UNA
NON TROPPO
BREVE INTRODUZIONE
ALL'USO DI
GNU/Linux
VIA CLI

Mariano E. Spadaccini

14 maggio 2025 – v0.2.3-alpha

I

"the possession of great power necessarily implies great responsibility"
William Lamb, 27 June 1817

"with great power comes great responsibility"
1962 Amazing Fantasy #15
written by Stan Lee
(narrative caption of the
comic book's last panel)

Copyright

This work is licensed under Attribution-NonCommercial-ShareAlike 4.0 International
To view a copy of this license, visit
http://creativecommons.org/licenses/by-nc-sa/4.0/

Indice

Copyright II

Indice XII

Lista delle figure XIII

Lista delle tabelle XVI

Elenco degli esempi XXXI

0 Premessa 1
0.1 Convenzioni tipografiche . . . 1

I Base 3

1 Introduzione 5
1.1 Storia . . . 5
1.1.1 Unix . . . 5
1.1.2 GNU . . . 6
1.1.3 Linux . . . 7
1.1.4 GNU/Linux . . . 9
1.2 Licenze . . . 10
1.3 Installazione . . . 11
1.4 Distribuzioni . . . 11
1.5 Approfondimenti . . . 11
1.5.1 Kernel monolitico vs microkernel . . . 11
1.6 Concetti chiave . . . 14

2 Shell 15
2.1 Login e logout . . . 15
2.1.1 Login . . . 15
2.1.2 Logout . . . 16
2.2 Cosa è una shell? . . . 16
2.2.1 Shell popolari . . . 16
2.2.2 Compiti della shell . . . 18
2.3 Comandi . . . 18
2.3.1 Struttura dei comandi . . . 18
2.3.2 Tipi di comando . . . 19

2.3.3 Metacaratteri ed escaping 20
2.4 Concetti chiave 22
2.5 Comandi introdotti nel capitolo 22

3 Documentazione **23**
3.1 Comando `help` e argomento `--help` 23
3.2 Manuale 24
3.2.1 Overview 24
3.2.2 Struttura 24
3.2.3 whatis e apropos 27
3.3 info 28
3.4 HowTo 28
3.5 Concetti chiave 29
3.6 Comandi introdotti nel capitolo 29

4 Gestione di file e directory **31**
4.1 File e percorsi 31
4.1.1 File 31
4.1.2 Esempi di filename 32
4.2 Directory 33
4.2.1 Directory tree 33
4.2.2 Home directory 33
4.2.3 Pathname assoluti e relativi 33
4.2.4 Shortcuts 34
4.3 Directory commands 35
4.3.1 Working directory 35
4.3.2 Change directory 35
4.4 List dei file e directory 36
4.5 Creazione e cancellazione di directory 38
4.6 Gestione dei file 40
4.6.1 Copiare, spostare e cancellare 40
4.6.2 Link 43
4.7 Espansione 44
4.7.1 Parentesi graffe 44
4.8 Search pattern 45
4.8.1 Simple pattern 45
4.8.2 Classi di carattere 46
4.8.3 Classi di carattere e internazionalizzazione 48
4.9 Ricerca di file 49
4.9.1 Ricerca rapida di file 49
4.9.2 Ricerca completa dei file 50
4.10 Approfondimenti 57
4.10.1 Rimozione di file 57
4.11 Concetti chiave 60
4.12 Comandi introdotti nel capitolo 61

5 Contenuto di file **63**
5.1 `cat` . . . 63
5.1.1 `tac` . . . 63
5.1.2 `rev` . . . 65
5.2 `more` . . . 66
5.3 `less` . . . 67
5.4 Inizio e fine . . . 68
5.4.1 `head` . . . 68
5.4.2 `tail` . . . 69
5.5 Concetti chiave . . . 72
5.6 Comandi introdotti nel capitolo . . . 72

6 Editor di file **73**
6.1 Pico . . . 73
6.2 Nano . . . 73
6.3 Vim . . . 74
6.4 Emacs . . . 75
6.5 La guerra degli editor . . . 76
6.6 Concetti chiave . . . 77
6.7 Comandi introdotti nel capitolo . . . 77

7 Ricerca tramite espressioni regolari **79**
7.1 Terminologia . . . 79
7.2 Regole di base . . . 79
7.3 Varianti delle espressioni regolari . . . 80
7.3.1 Basic regular expression . . . 80
7.3.2 Extended regular expression . . . 81
7.3.3 Classi di carattere e internazionalizzazione . . . 83
7.3.4 Posix BRE e Posix ERE . . . 83
7.3.5 Perl compatible regular expression . . . 83
7.4 Ricerca da comando . . . 84
7.5 Esempi . . . 84
7.6 Concetti chiave . . . 90

8 Standard I/O e redirezione **91**
8.1 Standard I/O . . . 91
8.2 Redirezione . . . 92
8.2.1 Redirezioni contemporanee . . . 93
8.2.2 Redirezione dello `stderr` . . . 93
8.2.3 Altre redirezioni . . . 94
8.3 Pipeline . . . 96
8.4 Command substitution . . . 97
8.5 Filtri . . . 98
8.6 Concetti chiave . . . 101
8.7 Comandi introdotti nel capitolo . . . 101

9 Data management **103**
9.1 Riepilogo . . . 103
9.2 sort . . . 103
9.3 uniq . . . 107
9.4 cut . . . 107
9.5 Concetti chiave . . . 110
9.6 Comandi introdotti nel capitolo . . . 110

10 Archiviazione e compressione **111**
10.1 Compressione file . . . 111
10.1.1 `gzip` e `gunzip` . . . 112
10.1.2 `bzip2` e `bunzip2` . . . 113
10.2 Archiviare file . . . 115
10.3 Spezzettare gli archivi . . . 119
10.4 Concetti chiave . . . 122
10.5 Comandi introdotti nel capitolo . . . 122

11 Variabili di shell e environment **123**
11.1 Variabili di shell . . . 123
11.2 Variabili di environment . . . 124
11.2.1 Stampa variabili d'ambiente^P . . . 125
11.3 `type` reloaded . . . 126
11.4 Concetti chiave . . . 129
11.5 Comandi introdotti nel capitolo . . . 129

12 Introduzione al filesystem **131**
12.1 Terminologia . . . 131
12.2 Tipi di file . . . 131
12.3 The Linux Directory Tree . . . 133
12.3.1 `/boot` . . . 134
12.3.2 `/bin` e `/usr/bin` . . . 134
12.3.3 `/sbin` e `/usr/sbin` . . . 134
12.3.4 `/lib` e `/usr/lib` . . . 134
12.3.5 `/dev` . . . 135
12.3.6 `/etc` . . . 138
12.3.7 `/opt` . . . 139
12.3.8 `/usr` . . . 139
12.3.9 `/proc` . . . 140
12.3.10 `/sys` . . . 141
12.3.11 `/var` . . . 141
12.3.12 `/tmp` . . . 142
12.3.13 `/srv` . . . 142
12.3.14 `/media` . . . 142
12.3.15 `/mnt` . . . 142
12.3.16 `/home` . . . 143
12.3.17 `/root` . . . 143
12.3.18 `/lost+found` . . . 143
12.4 Albero delle directory e filesystem . . . 143
12.5 Approfondimenti . . . 144
12.6 File di log . . . 144

12.6.1 Montaggio manuale di un filesystem . . . 144
12.6.2 Device via buffer . . . 148
12.6.3 Esempio di FIFO . . . 148
12.6.4 Esempio di Unix domain socket . . . 148
12.6.5 I file descriptor . . . 150
12.7 Concetti chiave . . . 153
12.8 Comandi introdotti nel capitolo . . . 154

13 La shell Bash **155**
13.1 Prima del comando . . . 155
13.2 Singola linea . . . 155
13.3 `alias` e `unalias` . . . 157
13.4 Storico . . . 159
13.5 Directory stack . . . 161
13.6 Impostazione all'avvio . . . 165
13.7 `source` . . . 166
13.8 Approfondimenti . . . 166
13.8.1 Stampa variabili d'ambiente $/2^P$ . . . 166
13.9 Concetti chiave . . . 167
13.10 Comandi introdotti nel capitolo . . . 167

14 Altri comandi utili **169**
14.1 watch . . . 169
14.2 xargs . . . 170
14.3 sleep . . . 171
14.4 `time` . . . 172
14.5 `timeout` . . . 172
14.6 Background e foreground . . . 174
14.7 bc . . . 176
14.8 convmv . . . 177
14.9 pr . . . 178
14.10 whereis . . . 178
14.11 locale . . . 179
14.12 tr . . . 179
14.13 figlet . . . 182
14.14 Concetti chiave . . . 185
14.15 Comandi introdotti nel capitolo . . . 185

15 Approfondimenti **187**
15.1 Procedura di *logout* . . . 187
15.1.1 logout . . . 187
15.1.2 exit . . . 187
15.1.3 Ctrl + D . . . 188
15.2 Concetti chiave . . . 190
15.3 Comandi introdotti nel capitolo . . . 190

II Principiante **191**

16 Introduzione all'amministrazione di sistema **193**
16.1 Il superutente . . . 193
16.2 Agire responsabilmente . . . 194
16.3 Diventare superutente . . . 194
16.3.1 switch user . . . 195
16.3.2 superuser do . . . 196
16.3.3 doas . . . 197
16.4 Configurazione di sistema . . . 198
16.5 Processi . . . 198
16.6 Memoria di sistema . . . 201
16.7 Istantanee dei processi e della memoria . . . 202
16.8 Package Management Intro . . . 204
16.9 Approfondimenti . . . 206
16.9.1 Comparazione tra `su`, `sudo` e `doas` . . . 206
16.9.2 Configurazione e utilizzo di `doas` . . . 207
16.9.3 system call e processi . . . 210
16.9.4 Rimuovere tutto, forse . . . 210
16.9.5 Pacchettizzazione delle librerie . . . 211
16.10 Concetti chiave . . . 212
16.11 Comandi introdotti nel capitolo . . . 212

17 Amministrazione degli utenti **213**
17.1 Utenza . . . 213
17.2 Utenti e gruppi . . . 214
17.2.1 Pseudoutenti e pseudogruppi . . . 214
17.3 Informazioni su utenti e gruppi . . . 215
17.3.1 Il file `/etc/passwd` . . . 215
17.3.2 Il file `/etc/shadow` . . . 216
17.3.3 Il file `/etc/group` . . . 217
17.3.4 Il file `/etc/gshadow` . . . 218
17.4 Gestire gli account utente e le informazioni del gruppo . . . 218
17.4.1 Creare account utente . . . 218
17.4.2 Il comando `passwd` . . . 220
17.4.3 Rimozione di un utente . . . 221
17.4.4 Modificare l'account utente e l'assegnazione dei gruppi . . . 222
17.4.5 Creazione, modifica e cancellazione di gruppi . . . 222
17.5 Approfondimenti . . . 224
17.5.1 Univocità di *UID* e *user* . . . 224
17.5.2 Comando `id` in un sistema Desktop . . . 224
17.6 Concetti chiave . . . 226
17.7 Comandi introdotti nel capitolo . . . 226

18 Access control **227**
18.1 Introduzione all'*access control system* . . . 227
18.2 Mappaggio dei permessi . . . 227
18.2.1 Rappresentazione numerica . . . 230
18.3 Accesso a file e directory . . . 230
18.3.1 Esempi su file e directory . . . 231

18.4 Visualizzazione dei permessi . . . 234
18.5 Cambio dei permessi . . . 235
18.5.1 Metodo simbolico . . . 236
18.5.2 Metodo numerico . . . 238
18.5.3 Esempio di cambio di permessi su file e directory . . . 239
18.6 Cambio di owner e group . . . 240
18.7 L'ownership dei processi e nuovi permessi . . . 241
18.8 Permessi speciali . . . 241
18.8.1 I permessi speciali per file . . . 241
18.8.2 Permessi speciali per directory . . . 243
18.8.3 Note aggiuntive per modifica dei permessi speciali . . . 246
18.9 Maschera dei permessi . . . 246
18.10 Approfondimenti . . . 248
18.10.1 id utente e gruppo nei processi^P . . . 248
18.11 Concetti chiave . . . 251
18.12 Comandi introdotti nel capitolo . . . 251

19 Installazione software **253**
19.1 Da sorgente . . . 253
19.1.1 Scaricamento sorgenti . . . 253
19.1.2 Installazione o pacchettizzazione della sorgente . . . 254
19.2 Installazione da pacchetto . . . 256
19.2.1 Concetti essenziali . . . 256
19.2.2 Breve comparazione tra package manager . . . 258
19.2.3 Esempi di package management . . . 260
19.3 Domare tutti i package manager . . . 266
19.4 Approfondimenti . . . 267
19.4.1 ./configure . . . 267
19.4.2 Tool Gdebi . . . 267
19.4.3 Stato dei pacchetti . . . 269
19.5 Concetti chiave . . . 271
19.6 Comandi introdotti nel capitolo . . . 271

20 Networking **273**
20.1 Background . . . 273
20.1.1 Protocolli . . . 273
20.1.2 Indirizzamento e routing . . . 278
20.1.3 Nomi e DNS . . . 281
20.1.4 IPv6 . . . 282
20.2 GNU/Linux come network client . . . 284
20.2.1 Indirizzamento . . . 284
20.2.2 DNS . . . 286
20.2.3 Troubleshooting . . . 287
20.3 Concetti chiave . . . 294
20.4 Comandi introdotti nel capitolo . . . 294

21 Hardware **295**
21.1 inxi . . . 295
21.2 hwinfo . . . 297
21.3 lshw . . . 299
21.4 lscpu . . . 300
21.5 dmidecode . . . 301
21.6 lspci . . . 303
21.7 hdparm . . . 305
21.8 SMART . . . 307
21.9 lsusb . . . 309
21.10 Network interface . . . 311
21.11 Comandi introdotti nel capitolo . . . 317

III Programmazione **319**

22 Introduzione al Bash scripting **321**
22.1 La shell come linguaggio di scripting . . . 321
22.2 Command line . . . 321
22.3 Creazione file . . . 322
22.3.1 Estensione . . . 322
22.4 Script eseguibile . . . 322
22.4.1 Intestazione . . . 322
22.4.2 Script eseguibile . . . 323
22.5 Contenuto . . . 323
22.5.1 Sequenza di comandi . . . 323
22.5.2 Commenti . . . 323
22.5.3 Variabili . . . 324
22.5.4 Opzioni . . . 326
22.5.5 Lettura . . . 327
22.5.6 Argomenti da linea di comando . . . 327
22.5.7 Gestione stringhe . . . 328
22.5.8 Valutazione aritmetica . . . 330
22.5.9 Espressioni . . . 330
22.5.10 Espressioni aritmetiche . . . 330
22.5.11 Strutture di controllo . . . 332
22.6 Array . . . 339
22.7 Funzioni . . . 340
22.8 Debugging . . . 341
22.9 Approfondimenti . . . 342
22.9.1 Shebang . . . 342
22.9.2 Portabilità . . . 342
22.9.3 Fork bomb . . . 343
22.10 Concetti chiave . . . 344
22.11 Comandi introdotti nel capitolo . . . 344

IV Intermedio **345**

23 Startup e shutdown **347**
23.1 Panoramica . . . 347
23.2 Avvio tradizionale . . . 348
23.3 Notifiche di sistema . . . 358
23.4 Evoluzioni nell'avvio . . . 359
23.4.1 Da BIOS a UEFI . . . 359
23.4.2 Da MBR a GPT . . . 360
23.5 Spegnimento e riavvio . . . 360
23.5.1 Da `System V` a `systemd` . . . 363
23.6 Approfondimenti . . . 363
23.6.1 Bootloader . . . 363
23.6.2 Identificazione dei dischi e delle partizioni . . . 363
23.7 Concetti chiave . . . 365
23.8 Comandi introdotti nel capitolo . . . 365

24 Logging **367**
24.1 Introduzione . . . 367
24.2 Kernel space . . . 367
24.2.1 Kernel log levels . . . 368
24.3 User space . . . 371
24.3.1 Syslog facilities . . . 372
24.3.2 Syslog priorities . . . 372
24.3.3 File di log . . . 373
24.3.4 Configurazione `rsyslogd` . . . 373
24.3.5 `logrotate` . . . 376
24.4 systemd-journald . . . 382
24.4.1 Report . . . 383
24.4.2 Manutenzione dello spazio . . . 390
24.5 Logger . . . 391
24.6 Approfondimenti . . . 392
24.6.1 Kernel ring buffer . . . 392
24.6.2 `rsyslog` o `systemd-journal` . . . 393
24.6.3 `printk`^{P} . . . 393
24.7 Concetti chiave . . . 395
24.8 Comandi introdotti nel capitolo . . . 395

V Complementi **397**

Tabella ASCII **399**

Tavola esadecimale, decimale, ottale e binario **403**

Multipli del byte **405**

Indice analitico **411**

Bibliografia e Webografia **415**

Elenco delle figure

1.1 Logo di BSD 6
1.2 HURD Live CD 7
1.3 Linus Torvalds nel 2002 8
1.4 Il messaggio di Linus Torvalds che annuncia la nascita di Linux 9
1.5 Tux, la mascotte di Linux 9
1.6 Logo della licenza ufficiale della *GNU General Public License* versione 3 10
1.7 Distinzione tra kernel monolitico vs microkernel 12

2.1 L'originale pagina di manuale della *Bourne shell* 17

4.1 Esempio di output del comando `tree` 34
4.2 Output del software ***S**team **L**ocomotive* richiamato attraverso il comando `sl` 38

5.1 Esempio del comando `more` 66

6.1 Esempio di editing tramite `nano` 74
6.2 Esempio di editing tramite `vim` 74
6.3 Esempi di modalità dell'editor *Vim* e passaggi tra le varie modalità 75
6.4 Esempio di editing tramite `Emacs` 76

12.1 Comando `tree -L 1` digitato sul pc dell'autore 133

14.1 Esempio di comando `watch -n 5`, la console si aggiornerà ogni 5 secondi 170
14.2 Esempio di comando `watch -d` 170
14.3 Esempio di comando `watch -t` 170

16.1 Esempio di comando `top`, la console si aggiornerà ogni 3 secondi 203

18.1 Esempio di output del comando `tree -p` 232

22.1 Versione moderna della *fork bomb* in *Bash* 343

23.1 Tipica schermata di configurazione del *BIOS* su PC 348
23.2 Uno screenshot datato di *GRUB Legacy*, essenzialmente rimasto inalterato nella configurazione di default per *GRUB 2* 350

Elenco delle tabelle

3.1 Manual page sections 25
3.2 Manual Page Topics 26

4.1 Comando `ls` 37
4.2 Opzioni del comando `ls` 37
4.3 Principali opzioni del comando `cp` 41
4.4 Principali opzioni del comando `rm` 43
4.5 Classi di carattere nominali *POSIX* 48
4.6 Alcuni test del comando `find` 53
4.7 I principali operatori del comando `find` 53
4.8 Le principali azioni disponibili per il comando `find` 54
4.9 Le principali opzioni globali disponibili per il comando `find` 55

5.1 Opzioni principali del comando `cat` 65
5.2 Opzioni principali del comando `head` 69
5.3 Suffissi e valore associato 70
5.4 Opzioni principali del comando `tail` 71

7.1 Metacaratteri utilizzabili nelle *BRE* 80
7.2 Metacaratteri utilizzabili nelle *ERE* 81
7.3 Equivalenze tra espressioni regolari nelle varianti Posix BRE e ERE . . . 83
7.4 Opzioni principali del comando `grep` 85

8.1 Gli standard channels sono i 3 standard *file descriptors* 91

9.1 Opzioni principali del comando `sort` 106
9.2 Opzioni principali del comando `uniq` 107
9.3 Opzioni principali del comando `cut` 108

10.1 Principali software di archiviazione e compressione 111
10.2 Opzioni principali di `gzip` 112
10.3 Opzioni principali di `bzip2` 114
10.4 Opzioni principali di `tar` 115
10.5 Principali opzioni del comando `split` 119

11.1 Descrizione delle principali variabili di environment 125

12.1 Descrizione delle varie tipologie di file (si è utilizzato `nome` come placeholder) 133

14.1 Opzioni principali per il comando watch . . . 169
14.2 Suffissi utilizzabili per il comando sleep . . . 172
14.4 Interpretazione delle sequenze nei SET . . . 180
14.3 Opzioni principali per il comando tr . . . 181

16.1 Opzioni principali per il comando su . . . 195

18.1 Conversione decimale di una singola cifra della maschera al permesso da attribuire . . . 246

19.1 Equivalenza delle azioni in alcuni dei package manager supportati . . . 268

20.1 Classi di indirizzi privati su IPv4 . . . 280

21.1 Flag principali per lspcilspci . . . 303
21.2 Opzioni principali per smartctl . . . 307

22.1 I principali operatori unari e binari utilizzabili per ottenere un booleano . 331
22.2 I principali operatori unari e binari aritmetici utilizzabili per ottenere un booleano . . . 332

23.1 Codici escape che possono essere utilizzati nel file /etc/issue . . . 358

24.1 I kernel log level. Se omesso sarà utilizzato KERN_DEFAULT . . . 368
24.2 Le log facilities gestite dalla printk() e dmesg . . . 370
24.3 I log levels (priorità) gestiti dalla printk() e dmesg . . . 371
24.4 Le log facilities . . . 372
24.5 Le log priority . . . 373
24.6 Extended ASCII - character codes 0 – 255 LATIN1/CP1252 . . . 401
24.7 Hexadecimal, Decimal, Octal and Binary . . . 403
24.8 Grandezza numerica, sigla e nome delle scale . . . 405

Lista degli esempi

2.1 Procedura di login con successo . . . 15
2.2 Procedura di login con credenziali errate . . . 16
2.3 Comando logout . . . 16
2.4 Comando exit . . . 16
2.5 Logout tramite la combinazione di tasti Ctrl + D . . . 16
2.6 Prompt dei comandi . . . 18
2.7 Prompt dei comandi semplificato . . . 18
2.8 Comando `type` . . . 19
2.9 Comando `which` . . . 20
2.10 Comando `which` . . . 20
2.11 Comando `echo` . . . 20
2.12 Comando `date` . . . 20
2.13 Comando `help` . . . 21
2.14 Caratteri speciali . . . 21
3.1 Comando `help exit` . . . 23
3.2 Comando `exit --help` . . . 23
3.3 Opzione `--help` . . . 24
3.4 Comando `man bash` . . . 25
3.5 Comando `man crontab` . . . 26
3.7 Comando `man 5 crontab` . . . 26
3.6 Comando `man crontab`, sezione `SEE ALSO` . . . 27
3.8 Comando `man -all crontab` . . . 27
3.9 Comando `man -a crontab` . . . 27
3.10 Comando `whatis` . . . 27
3.11 Comando `apropos` . . . 28
4.1 Creazione file tramite `touch` . . . 31
4.2 Caratteri consigliati . . . 31
4.3 Creazione file ed escaping . . . 32
4.4 Esempi di nomi di file con commento . . . 32
4.5 Comando `tree` . . . 33
4.6 Alcuni path name assoluti . . . 34
4.7 Alcuni path name relativi . . . 34
4.8 Shortcut ~ . . . 35
4.9 Esempio dello shortcut . . . 35
4.10 Esempio del comando `pwd` . . . 35
4.11 Esempio del comando `cd` . . . 35
4.12 Esempi dei comandi `cd` e `pwd` con prompt più comune . . . 36
4.13 Esempio del comando `cd -` . . . 36

4.15 Esempio di digitazione errata del comando `ls` . . . 37
4.14 Esempi con il comando `ls` . . . 38
4.18 Esempio con il comando `mkdir -p` . . . 39
4.19 Segmento del manuale di `rmdir` . . . 39
4.16 Esempio con il comando `mkdir` . . . 39
4.17 Segmento del manuale di `mkdir` . . . 39
4.20 Esempio con il comando `rmdir -p` . . . 40
4.21 Segmento del manuale di `cp` . . . 40
4.22 Esempio con il comando `cp` . . . 41
4.23 Segmento del manuale di `mv` . . . 42
4.24 Esempio con il comando `mv` . . . 42
4.26 Esempio con il comando `ln` (*hard link*) . . . 43
4.25 Esempio con il comando `rm` . . . 43
4.27 Esempio con il comando `ln -s` (*soft link*) . . . 44
4.28 Esempio di lista/1 . . . 45
4.29 Esempio di lista (2) . . . 45
4.30 Esempio di creazione di directory tramite . . . 45
4.31 Simple pattern . . . 45
4.32 Simple pattern con `echo` . . . 46
4.33 Esempio di pattern che non matcha . . . 46
4.34 Esempio di classe di carattere . . . 46
4.35 Esempio di classi di carattere con intervalli . . . 47
4.36 Esempio di classe di carattere con intervallo . . . 47
4.37 Esempio di classi di carattere con intervallo ampio . . . 47
4.38 Esempio di classi di carattere con intervallo ampio (2) . . . 47
4.39 Esempio di concatenazione di classi di concatenazione di pattern . . . 47
4.40 Esempio di negazione di classi di carattere . . . 48
4.41 Esempio di utilizzo di `locate` . . . 49
4.42 Esempio di `find` senza parametri (*path relativo* di default `.`) . . . 50
4.43 Segmento del manuale di`find` . . . 50
4.44 Esempio di `find` dalla *home* (*path assoluto* `/home/ms`) . . . 51
4.45 Esempio di `find` . . . 54
4.46 Esempio di `find` (2a) . . . 55
4.47 Esempio di `find` (2b) . . . 55
4.48 Esempio di `find` (3) . . . 55
4.49 Esempio di `find` (4) . . . 56
4.50 Esempio di `find` (5) . . . 56
4.51 Esempio di `find` (6) . . . 56
4.52 Esempio di `find` (7) . . . 56
4.53 Esempio di `find` (8) . . . 57
4.54 Esempio di `find` (9) . . . 57
4.55 Esempio di `find` (10) . . . 57
4.56 Elenco di file con sequenze antipatiche . . . 57
4.57 Cancellazione file precedenti tramite *escaping* . . . 58
4.58 Cancellazione file precedenti tramite *quoting* . . . 58
4.59 Cancellazione file precedenti/2 . . . 58
4.60 Elenco di file con sequenze molto antipatiche . . . 58
4.61 Tentativi singoli di cancellazione dei file precedenti . . . 58
4.62 Tentativo di cancellazione dei file precedenti tramite pattern . . . 58

4.63 Comprensione del pattern tramite `echo` . . . 58
4.64 Utilizzare la sequenza `--` per esplicitare la fine delle opzioni . . . 59
4.65 Workaround per la cancellazione dei file . . . 59
4.66 Workaround per la cancellazione dei file tramite `find` . . . 59
4.67 Cancellazione tramite *inode* . . . 59
5.1 Comando `cat` senza argomenti e input come nell'esempio . . . 63
5.2 Comando `cat` con argomenti . . . 63
5.3 Comandi `cat` senza e con opzioni . . . 64
5.4 Segmento del manuale di `tac` . . . 64
5.5 Comandi `cat` e `tac` a confronto . . . 64
5.6 Processamento dello standard input da `tac` . . . 64
5.7 Segmento del manuale di `rev` . . . 65
5.8 Comando `rev` a confronto con quanto mostrato nell'**Esempio 5.5** . . . 65
5.9 Processamento dello standard input da `tac` . . . 66
5.10 Paginazione tramite `more/1` . . . 67
5.11 Paginazione tramite `more` (2) . . . 67
5.12 Paginazione tramite `more` (3) . . . 67
5.13 Segmento del manuale di `less` . . . 67
5.14 Paginazione tramite `less` . . . 68
5.15 Comando `head` su file di testo . . . 69
5.16 Comando `head -n` su file di testo . . . 69
5.17 Comando `tail` su file di testo . . . 70
5.18 Comando `tail -n` su file di testo . . . 70
5.19 Comando `tail -f` su file di testo . . . 71
6.1 Edit di un file tramite `vi` . . . 74
6.2 `vi` al posto di `vim` . . . 75
6.3 Edit di un file tramite `emacs` . . . 76
7.1 Esempi di regexp variante BRE . . . 80
7.2 Esempi di regexp variante ERE . . . 82
7.3 Contenuto di file utilizzato nei prossimi esempi . . . 84
7.4 Contenuto di file utilizzato nei prossimi esempi . . . 84
7.5 Contenuto di file utilizzato nei prossimi esempi . . . 86
7.6 `grep` di stringa . . . 86
7.7 `grep` di stringa su più file . . . 86
7.8 `grep` di una stringa attivando il *case insensitive* . . . 86
7.9 `grep` di una regexp . . . 87
7.10 `grep` di una parola intera . . . 87
7.11 `grep` non della parola intera . . . 87
7.12 `grep` di una parola intera attivando il case insensitive . . . 87
7.13 `grep` con linee dopo il match . . . 88
7.14 `grep` con linee prima del match . . . 88
7.15 `grep` con linee intorno al match . . . 88
7.16 `grep` ricorsivo . . . 88
7.17 `grep -v` per invertire il match . . . 88
7.18 `grep` con più pattern in seguenza . . . 89
7.19 `grep -l` per solo i nomi dei file . . . 89
8.1 Redirezione output tramite `>` . . . 92
8.2 Redirezione output tramite `>` . . . 92
8.3 Redirezione tramite `<` . . . 92

8.4 Redirezione tramite < al comando wc . . . 93
8.5 Redirezione contemporanee . . . 93
8.6 Redirezione solo stdout su file . . . 93
8.7 Redirezione anche stderr su file . . . 94
8.8 Redirezione stderr nello stdout . . . 94
8.9 Sintassi di un *here document* . . . 94
8.10 Sintassi di un *here document* con il comando cat . . . 94
8.11 Sintassi di un *here document* con l'espansione delle variabili . . . 95
8.12 Sintassi di un *here document* conflusso letterale . . . 95
8.13 Un *here document* per creazione ed esecuzione di uno script . . . 95
8.14 Un *here strings* con il comando bc . . . 96
8.15 Esempio di pipeline . . . 96
8.16 Esempio di pipeline con tee . . . 97
8.17 Sostituzione tramite backtick . . . 97
8.18 Sostituzione tramite $() . . . 97
8.19 Esempio del comando grep su stdin immesso da tastiera . . . 98
8.20 Esempio del comando sort su stdin immesso da tastiera . . . 98
8.21 Esempio del comando sort in pipeline con grep su stdin immesso da tastiera . . . 99
9.1 Comando cat su file da ordinare . . . 104
9.2 Comando sort su file senza opzioni . . . 104
9.3 Comando sort con ridefinizione della variabile LC_ALL . . . 105
9.4 Comando sort -n . . . 106
9.5 Comando sort -n . . . 106
9.6 Comando uniq in pipeline con sort . . . 107
9.7 Comando cut su stdin . . . 108
9.8 Comando cut su byte . . . 108
9.9 Comando cut su /etc/passwd precedentemente filtrato tramite grep 109
10.1 Compressione con gzip . . . 112
10.2 Decompressione con gzip . . . 113
10.3 Decompressione con gunzip . . . 113
10.4 Compressione con gzip -v . . . 113
10.5 Decompressione con gunzip -v . . . 113
10.6 Decompressione su stdout . . . 113
10.7 Compressione con bzip2 . . . 113
10.8 Decompressione con bzip2 . . . 114
10.9 Decompressione con bunzip2 . . . 114
10.10 Compressione con bzip2 -v . . . 114
10.11 Decompressione con bunzip2 -v . . . 114
10.12 Decompressione su stdout . . . 114
10.13 Sintassi generica dello strumento di archiviazione tar . . . 115
10.14 Creazione di un archivio di file . . . 116
10.15 Creazione di un archivio di directory . . . 116
10.16 Visualizzazione indice del contenuto . . . 116
10.17 Visualizzazione indice del contenuto con -v . . . 116
10.18 Estrazione contenuto . . . 117
10.19 Estrazione contenuto . . . 117
10.20 Creazione di un file compresso con gzip da un archivio di file . . . 117
10.21 Creazione di un archivio di file e compressione con gzip . . . 117

10.22 Decompressione file di archivio `gzip` e estrazione del contenuto 117
10.23 Decompressione ed estrazione file di archivio `gzip` 118
10.24 Creazione di un file compresso con `bzip2` da un archivio di file 118
10.25 Creazione di un archivio di file e compressione con `bzip2` 118
10.26 Decompressione file di archivio `bzip2` e estrazione del contenuto 118
10.27 Decompressione ed estrazione file di archivio `bzip2` 118
10.28 Sintassi generica del comando `split` 119
10.29 `split` da $100M$ 119
10.30 `split` con numero di chunk definito 119
10.31 `split` da $100M$ 120
10.32 `split` da $100M$ con prefisso assegnato 120
10.33 `split` da $100M$ con prefisso assegnato e opzioni `-dx` 120
10.34 `cat` dei chunk 121
10.35 Calcolo del digest SHA dei due file 121
11.1 Assegnazione di una variabile e utilizzo 123
11.2 Cancellazione di una variabile 123
11.3 Assegnazione di una variabile e esportazione 124
11.4 Assegnazione di una variabile con esportazione 124
11.5 Visualizzazione delle variabili di ambiente 124
11.6 Rimozione della variabile dall'ambiente 125
11.7 Script Perl per visualizzare le variabili d'ambiente 125
11.8 Output dello script dell'***Esempio 11.7*** 126
11.9 Esempio di comando interno 126
11.10 Variabile `PATH` 126
11.11 Ricerca del comando `find` secondo quanto indicato nella variabile `PATH` 126
11.12 `type` di `find` per verificare l'*hashing* 127
11.13 Comando `hash` 127
11.14 Rimozione entry dell'hash 127
11.15 Comando `which` di `find` 128
11.16 Verifica del non uso dell'hashing del comando `which` 128
11.17 `type` vs `which` del comando `echo` 128
12.1 Determina il contenuto del file tramite il comando `file` 131
12.2 Alcuni file contenuti in `/dev` 135
12.3 Redirezione su `/dev/null` 137
12.4 `echo 1` su di `/dev/tty` 137
12.5 `cat` su `/dev/tty` 138
12.6 Uso di `dd` con fornitore `/dev/zero` 138
12.7 Informazioni sulla release tramite `lsb_release` 139
12.8 Informazioni sul carico sul sistema tramite `/proc/loadavg` e il comando `uptimed` 140
12.9 Elenco generico di home degli utenti 143
12.10 Elenco generico di home degli utenti 143
12.11 In ascolto del `syslog` 145
12.12 Lista dei dispositivi a blocchi prima dell'inserimento 145
12.13 Lista dei dispositivi a blocchi dopo l'inserimento 146
12.14 Individuazione dispositivo e partizione tramite `dmesg` 146
12.15 Individuazione dispositivo e partizione tramite `fdisk` 146
12.16 Mounting manuale di una USB Key attraverso il comando `mount` 147
12.17 Unmounting manuale di una USB Key attraverso il comando `umount` . . 147

12.18 Creazione di una named pipe tramite `mkfifo` . . . 148
12.19 Scrittura sulla named pipe . . . 148
12.20 Lettura dalla named pipe . . . 148
12.21 Creazione di una Unix domain socket tramite `nc` . . . 149
12.22 Verifica della socket tramite `lsof` . . . 149
12.23 Connessione alla Unix domain socket creata precedentemente . . . 150
12.24 Creazione di una Unix domain socket tramite `nc` ma non orientata alla connessione . . . 150
12.25 Verifica della socket non orientata alla connessione tramite `lsof` . . . 150
12.26 Connessione alla Unix domain socket creata precedentemente . . . 150
12.27 Esempio di uso di *file descriptor* . . . 151
13.1 Esecuzione di una sequenza di comandi . . . 155
13.2 Sequenza di variabili prima del comando . . . 155
13.3 Sequenza di comandi su singola linea . . . 156
13.4 Verifica dell'*exit code* . . . 156
13.5 Esecuzione di comandi separati da `&&` . . . 157
13.6 Esecuzione di comandi separati da `||` . . . 157
13.7 Generica impostazione di un alias . . . 157
13.8 Impostazione dell'alias `hi` . . . 157
13.9 Esecuzione dell'alias `hi` . . . 157
13.10 Impostazione di un alias `hi` . . . 158
13.11 Impostazione dell'alias `lm` . . . 158
13.12 Impostazione dell'alias `rm` . . . 158
13.13 Nuova impostazione dell'alias `hi` . . . 158
13.14 Alias che non funziona come si potrebbe ipotizzare . . . 158
13.15 Alias che non funziona come si potrebbe ipotizzare . . . 159
13.16 Cancellazione dell'alias `hi` . . . 159
13.17 Impostazione di un alias `hi` . . . 159
13.18 Esecuzione dell'alias `hi` appena cancellato . . . 159
13.19 Ricerca nello storico tramite Ctrl + r . . . 160
13.20 Stampa della cronologia . . . 160
13.21 Valore di $HISTFILE . . . 160
13.22 Valori di $HISTSIZE e $HISTFILESIZE . . . 161
13.23 Verifica del comportamento $HISTSIZE e $HISTFILESIZE . . . 161
13.24 Valore di $HISTFILE . . . 161
13.25 Stack iniziale . . . 161
13.26 Spostamento su altra directory e verifica che lo stack inizia ancora con la current directory . . . 162
13.27 Spostamento su altra directory tramite `pushd` . . . 162
13.28 Visualizzazione dello stack tramite `dirs -v` . . . 162
13.29 Visualizzazione dello stack tramite `dirs -v -l` . . . 162
13.30 Visualizzazione dello stack tramite `dirs -v` . . . 163
13.31 Aggiunta di directory allo stack tramite `pushd -n` . . . 163
13.32 Visualizzazione stack tramite `dirs -v` . . . 163
13.33 Eliminazione dallo stack di `/etc/defaults` . . . 163
13.34 Spostamento tramite lo stack . . . 164
13.35 Variabile d'ambiente `DIRSTACK` . . . 165
13.36 Variabile d'ambiente `DIRSTACK` . . . 165

13.37 Il comando `source` utilizzato per modificare aspetti dell'environment dell'utente . . . 166
13.38 Output dello script dell'***Esempio 11.7 che non mosrtrerà quanto indicato nella stessa sezione*** . . . 166
14.1 Esempio creazione di directory tramite *xargs* . . . 170
14.2 Esempio rimozione di file tramite *xargs* . . . 171
14.3 Esempio rimozione di file tramite *xargs -0* . . . 171
14.4 Esempio rimozione di file tramite *find -exec* . . . 171
14.5 Esempio rimozione di file tramite *xargs -0* . . . 171
14.6 Elenco compatto di utenti di sistema . . . 171
14.7 Sintassi generica del comando `sleep` . . . 171
14.8 Sequenza di comandi `sleep` . . . 172
14.9 Il comando `time` con il software `sleep` . . . 172
14.10 Sintassi generica del comando `timeout` . . . 172
14.11 Esempio del comando `timeout` con il software `sleep` . . . 173
14.12 Esempio del comando `timeout` con l'opzione `--preserve-status` 173
14.13 Esempio del comando `time` che esegue `timeout` il quale esegue `sleep` 174
14.14 Invio di un generico comando in background . . . 174
14.15 Invio di due `sleep` in background . . . 174
14.16 Invio di un nuovo `sleep` in background . . . 174
14.17 Attesa di job in background . . . 175
14.18 Check dei job in coda . . . 175
14.19 Wait dei job in coda . . . 175
14.20 Spostamento in foreground del job . . . 175
14.21 Esempi di *bc* . . . 176
14.22 Sintassi di un *bc* . . . 176
14.23 Sintassi di un *bc* . . . 177
14.24 Elenco *set di caratteri* utilizzabili . . . 177
14.25 Esempi di conversione da un *set di caratteri* ad un altro in *test mode* . . . 177
14.26 Esempi di conversione da un *set di caratteri* ad un altro in *test mode* dell'intera `/home` . . . 178
14.27 Elenco degli utenti di sistema su due colonne . . . 178
14.28 Esempi del comando `whereis` . . . 178
14.29 Esempi del comando `locale` . . . 179
14.30 Esempio di comando `tr` . . . 179
14.31 Sintassi del comando `tr` . . . 179
14.32 `tr` di `{}` in `()` . . . 180
14.33 `tr` di `{}` in `[]` . . . 180
14.34 `tr` di `{}` in `[]` corretta . . . 181
14.35 `tr` per sostituzione di maiuscole in minuscole tramite range . . . 181
14.36 `tr` per sostituzione di maiuscole in minuscole tramite classe . . . 181
14.37 `tr` per cancellare sequenze di numeri . . . 181
14.38 `tr` per cancellare sequenze non numeriche . . . 182
14.39 Altri esempi di `tr` . . . 182
14.40 Semplici esempi di `figlet` . . . 182
14.41 Esempio di `figlet -f` . . . 183
14.42 Esempio di `figlet -c` . . . 183
14.43 Esempio di `figlet -c` . . . 183
14.44 Esempio di `figlet -kc` il cui `stdin` proviene dal comando `date` . . . 183

14.45 Esempio di `figlet -kc` il cui `stdin` proviene dal comando `date` eseguito con `watch` . . . 184
15.1 Comandi `type logout` e `logout --help` . . . 187
15.2 Comandi type exit e exit –help . . . 187
15.3 Comando man exit . . . 188
15.4 stty -a . . . 188
16.1 `ls` di `/root` da utente ordinario . . . 193
16.2 `ls` di `/root` e dell'albero `/home` da amministratore . . . 194
16.3 Sintassi generale del comando `su` . . . 195
16.4 Alternativa sintattica principale del comando `su` con caricamento dell'environment associato all'utente target . . . 195
16.5 Alternativa sintattica principale del comando `su` senza caricamento dell'environment associato all'utente target . . . 195
16.6 Utilizzo di `su -` per diventare *root* . . . 196
16.7 Utilizzo di `su -` da *root* . . . 196
16.8 Esempio di `sudo command` . . . 196
16.9 Esempio di `sudo -i` . . . 196
16.10 Esempio di `sudo -i` non . . . 197
16.11 Esempio di `doas command` . . . 197
16.12 Esempio di `doas -s` . . . 197
16.13 `ps` senza opzioni aggiuntive . . . 198
16.14 `ps x` . . . 199
16.15 `ps $$` . . . 200
16.16 `ps l 1`, in cui `l` mostra un formato lungo . . . 200
16.17 `ps aux`, in cui `a` mostra tutti i processi, `x` elimina il vincolo di una `tty` associata, `u` l'user associato . . . 201
16.18 `free` . . . 201
16.19 `free` con l'opzione `--human` o `-h` . . . 202
16.20 `free` con l'opzione `--human` o `-h` aggiungendo `--si` . . . 202
16.21 Packege sul sistema relativi all'editor *Vim* . . . 204
16.22 Lista dei pacchetti installati in una *Debian-based* . . . 205
16.23 Lista dei pacchetti installati in una *RPM-based* . . . 205
16.24 `doas` installato ma non configurato . . . 207
16.25 Semplice linea di configurazione per `doas` . . . 207
16.26 Sintassi di una linea di configurazione per `doas` . . . 207
16.27 Linea di configurazione per `doas` per non richiedere la password all'utente 208
16.28 Linea di configurazione per `doas` con vincolo di comando e argomento . 208
16.29 Linea di configurazione per `doas` . . . 208
16.30 Uso di `doas` con indicazione di utente target . . . 209
16.31 Uso di `doas` per lanciare una shell . . . 209
16.32 Uso di `doas` per lanciare una shell con definizione di utente target . . . 209
16.33 Check di `doas.conf` . . . 209
16.34 Cancellazione tramite `rm -rf` di un path . . . 210
16.35 Cancellazione tramite `rm -rf` con digitazione errata di path . . . 210
17.1 *UID* e *GID* associati all'utente . . . 214
17.2 `groups` . . . 214
17.3 Sintassi generica di *useradd* . . . 218
17.4 Creazione utente tramite `useradd` . . . 219
17.5 Creazione utente tramite `adduser` . . . 219

17.6 Cambio password tramite `passwd` . . . 220
17.7 Cambio della propria password tramite `passwd` . . . 220
17.8 Verifica dello stato della propria password tramite `passwd -S` . . . 220
17.9 Verifica dello stato della propria password tramite `passwd -S` da `root` 220
17.10 *Lock* e *unlock* della password dell'account `ms` tramite `passwd -l` e `passswd -u` da `root` . . . 221
17.11 Sintassi generica di *userdel* . . . 221
17.12 Rimozione di file tramite *UID* . . . 221
17.13 Rimozione di utente tramite `deluser` con rimozione della home directory associata . . . 222
17.14 Rimozione completa dell'utente tramite `deluser` . . . 222
17.15 Cambiare `GID` tramite `usermod` . . . 222
17.16 Aggiungere un utente ad un gruppo tramite `usermod` . . . 222
17.17 Sintassi generica di *groupadd* . . . 223
17.18 Aggiunta di un gruppo tramite `addgroup` . . . 223
17.19 Sintassi generica di *groupmod* . . . 223
17.20 Sintassi generica di *groupdel* . . . 223
17.21 Sintassi generica di *gpasswd* . . . 223
17.22 `ls -ld .*` della home . . . 224
17.23 `ls -nd .*` della home . . . 224
17.24 *UID* e *GID* tipicamente ad un utente desktop . . . 225
18.1 Esecuzione di `ls -l` per l'interpretazione dei permessi associati alle risorse 227
18.2 Prima colonna precedente suddivisa in due blocchi . . . 228
18.3 Blocco dei permessi suddiviso per la categoria di utenti a cui si riferiscono 228
18.4 Lista dei file tramite `find` senza scendere di livello . . . 231
18.5 Esempi con il file `noread.txt` . . . 232
18.6 Esempi con il file `nowrite.txt` . . . 232
18.7 Esempi con il file `noexecute.sh` . . . 232
18.8 Lista delle directory tramite `find` senza scendere di livello . . . 233
18.9 Esempi con la directory `noread` . . . 233
18.10 Esempi con la directory `nowrite` . . . 233
18.11 Esempi con la directory `noexecute` . . . 233
18.12 Esecuzione di `ls -ld noexecute*` e `stat noexecute*` . . . 234
18.13 Sintassi generica del comando `chmod` . . . 235
18.14 Creazione di `script.sh` e `scripts` . . . 239
18.15 Eliminazione del permesso di scrittura al gruppo . . . 239
18.16 Eliminazione del permesso di scrittura al gruppo e aggiunta dell'esecuzione sia al proprietario sia al gruppo . . . 240
18.17 Sintassi generica del comando `chown` . . . 240
18.18 Esempi di `chown` . . . 240
18.19 Esempi di `ps` con output personalizzato . . . 241
18.20 Verifica del tool `passwd` . . . 241
18.21 `chmod` per *SUID* e *SGID* in modalità simbolica . . . 242
18.22 `ls -l` per verificare *SUID* e *SGID* con tipico carattere `s` . . . 242
18.23 `ls -l` per verificare *SUID* e *SGID* con carattere `S` . . . 243
18.24 `chmod` per *SUID* e *SGID* in modalità numerica . . . 243
18.25 Creazione del gruppo `progetto_top` . . . 244
18.26 Creazione della directory assegnandola al gruppo `project_top` . . . 244
18.27 Directory di sistema con lo *sticky bit* . . . 244

18.28 `chmod` per *SUID* e *SGID* in modalità simbolica . . . 245
18.29 `chmod` per *SUID* e *SGID* e *stickybit* in modalità numerica . . . 245
18.30 Eliminazione dei permessi speciali per le directory in modalità numerica . 246
18.31 Visualizzazione della maschera dei permessi correnti . . . 246
18.32 Visualizzazione dei permessi consentiti . . . 247
18.33 Modifica maschera dei permessi . . . 247
18.34 Creazione di directory e file e visualizzazione dei permessi associati . . . 247
18.35 Creazione di directory e file con maschera modificata . . . 248
18.36 Verifica del binario `passwd` . . . 250
18.37 Verifica dei processi attivi con le colonne di interesse . . . 250
19.1 Scaricare il sorgente . . . 253
19.2 Apertura dell'archivio . . . 254
19.3 Comandi per la compilazione del sorgente . . . 254
19.4 Comandi per la compilazione del sorgente e l'installazione tramite pacchettizzazione . . . 255
19.5 Alternative maggiori del comando `checkinstall` . . . 255
19.6 Comandi per l'aggiornamento della lista dei pacchetti `.deb` disponibili . 260
19.7 Comandi per l'aggiornamento della lista dei pacchetti `.rpm` disponibili . 260
19.8 Comandi per l'aggiornamento del sistema `.deb` . . . 261
19.9 Comandi per l'aggiornamento del sistema `.rpm` . . . 261
19.10 Ricerca di pacchetti `.deb` . . . 261
19.11 Informazioni su pacchetto `.deb` . . . 261
19.12 Ricerca di un editor e check di pacchetto tramite `apt-cache` . . . 261
19.13 Ricerca di un editor e check di pacchetto tramite `apt` . . . 262
19.14 Lista delle dipendenze tramite `apt-cache` e `apt` . . . 263
19.15 Ricerca tramite `dnf` di pacchetti `.rpm` . . . 263
19.16 Informazioni tramite `dnf` su pacchetto `.rpm` . . . 263
19.17 Lista delle dipendenze tramite `dnf` . . . 263
19.18 Installazione pacchetti `.deb` da repository . . . 264
19.19 Installazione pacchetti `.rpm` da repository . . . 264
19.20 Installazione pacchetti `.deb` da locale . . . 264
19.21 Installazione pacchetti `.rpm` da locale a basso livello . . . 264
19.22 Installazione pacchetti `.rpm` da locale . . . 265
19.23 Rimozione di un generico pacchetto `.deb` . . . 265
19.24 Rimozione di un generico pacchetto `.deb` con rimozione dei relativi file di configurazione . . . 265
19.25 Rimozione di un generico pacchetto `.rpm` . . . 265
19.26 Elenco dei pacchetti `.deb` installati . . . 265
19.27 Elenco dei pacchetti `.rpm` installati . . . 265
19.28 Opzioni del config . . . 267
19.29 Opzioni del config . . . 267
19.30 Installazione pacchetto locale con successiva risoluzione delle dipendenze 267
19.31 Installazione pacchetto locale con risoluzione delle dipendenze . . . 269
19.32 Installazione pacchetto locale con risoluzione delle dipendenze via *APT* . 269
19.33 Simulazione dell'installazione pacchetto locale con risoluzione delle dipendenze via *APT* . . . 269
19.34 Elencare i pacchetti `rc` . . . 270
19.35 Eliminare i pacchetti `rc` . . . 270
20.1 Esempio di utilizzo di `ifconfig` . . . 285

20.2 Esempio di utilizzo di `route` per aggiungere il *default gateway* 285
20.3 Esempio di utilizzo di `ip` 286
20.4 Esempio di utilizzo di `ip -6 route` per aggiungere il *default gateway* 286
20.5 Contenuto del file `/etc/resolv.conf` 286
20.6 Esempio di utilizzo di `ifconfig interfaccia` 287
20.7 Esempio di utilizzo di `ping` su rete locale 288
20.8 Esempio di utilizzo di `ping` remoto 288
20.9 Esempio di utilizzo di `ping` con invio intervallato da 0.2 secondi 289
20.10 Esempio di utilizzo di `ping` con flood 290
20.11 Esempio di utilizzo di `ping6` su loopback 290
20.12 Esempio di utilizzo di `dig` su dominio 291
20.13 `HEADER` relativa all'***Esempio 20.12*** 291
20.14 `QUESTION SECTION` relativa all'***Esempio 20.12*** 291
20.15 `ANSWER SECTION` relativa all'***Esempio 20.12*** 292
20.16 Sommario relativo all'***Esempio 20.12*** 292
20.17 Esempio di utilizzo di `dig` su dominio con type `any` 292
20.18 Query `dig` diretta ad un name server 293
20.19 Query diretta e inversa con `+short` 293
21.1 Esecuzione di `inxi` 295
21.2 Esecuzione di `inxi -B` 296
21.3 Esecuzione di `inxi -C` 296
21.4 Esecuzione di `inxi --disk --memory` 296
21.5 Esecuzione di `inxi --weather` 296
21.6 Esecuzione di `inxi -W pescara,italy` 296
21.7 Esecuzione di `inxi -iz` 297
21.8 Esecuzione di `hwinfo --short` 297
21.9 Esecuzione di `hwinfo --memory` 298
21.10 Esecuzione di `lshw -short -sanitize` 299
21.11 Esecuzione di `lshw -class volume` 300
21.12 Esecuzione di `lscpu` 301
21.13 Esecuzione di `lscpu` 302
21.14 Esecuzione di `dmideconde --string processor-frequency` 303
21.15 Esecuzione di `lspci` 303
21.16 Esecuzione di `lspci -v` 303
21.17 Esecuzione di `lspci -vvx` 304
21.18 Esecuzione di `hdparm -I /dev/sda` 305
21.19 Esecuzione di `hdparm -tT /dev/sda6` 306
21.20 Esecuzione di `smartctl -i /dev/sda` 307
21.21 Esecuzione di `smartctl -s on /dev/sda` 308
21.22 Esecuzione di `smartctl -s off /dev/sda` 308
21.23 Esecuzione di `smartctl -H /dev/sda` 308
21.24 Esecuzione di `smartctl -t short /dev/sda` 308
21.25 Esecuzione di `smartctl -l selftest /dev/sda` 309
21.26 Esecuzione di `lsusb` 309
21.27 Esecuzione di `lsusb -s` 309
21.28 Esecuzione di `lsusb` con storage messo su hub con potenzialità minori . 310
21.29 Esecuzione di `lsusb -s` 310
21.30 Esecuzione di `inxi --network` 311
21.31 Esecuzione di `inxi --network-advanced` 311

21.32Esecuzione di `hwinfo --short --netcard` 311
21.33Esecuzione di `lshw -class -network -short` 311
21.34Esecuzione di `lshw -class -network -sanitize` 312
21.35Esecuzione di `lspci | grep -Ei 'eth|network|ethernet|wireless|wifi'` 312
21.36Esecuzione di `lspci -vvs 02:00.0` 312
21.37Esecuzione di `ip link` 313
21.38Esecuzione di `ifconfig -a` 314
21.39Esecuzione di `nmcli device status` 314
21.40Esecuzione di `netstat -i` 315
21.41Esecuzione di `tcpdump --list-interfaces` 315
21.42Esecuzione di `cat /proc/net/dev` 316
22.1 Esecuzione di una sequenza di comandi 321
22.2 Eseguire lo script lanciando l'interprete ed indicando come argomento il testo creato 322
22.3 Eseguire lo script come argomento dell'interprete 322
22.4 Intestazione del Bash 322
22.5 Intestazione del Bash tramite `env` 323
22.6 Intestazione del Bash tramite `env` 323
22.7 Script con sequenza di comandi 323
22.8 Script con commento 323
22.9 Script con commento multiline 324
22.10Uso di variabili 324
22.11Uso di variabili tramite espansione 324
22.12Nome dello script e opzioni correntemente impostate 325
22.13Esecuzione dello script 22.12 325
22.14Valutazione interattiva di `$-` 326
22.15Attivazione di un flag 326
22.16Immissione da tastiera 327
22.17Lettura da file 327
22.18Mostra gli argomenti 327
22.19Esecuzione dello script 22.18 327
22.20Script per concatenazione di stringhe 328
22.21Concatenazione di stringhe 328
22.22Script per ottenere lunghezza di stringhe 328
22.23Esecuzione di 22.22 328
22.24Replacement di stringhe 329
22.25Esecuzione di 22.24 329
22.26Esempi di substring 329
22.27Esecuzione di 22.26 329
22.28Valutazione aritmetica 330
22.29Esempio di valutazione aritmetica 330
22.30Esecuzione aritmetica 330
22.31Alcune operazioni aritmetiche 330
22.32Esecuzione di operazioni aritmetiche 331
22.33Costrutto generico *if-then-eif-else* 332
22.34Costrutto generico *if-then-eif-else* con `test` 333
22.35Costrutto generico *if-then-eif-else* con doppia parentesi quadra 333
22.36Costrutto generico *if-then-eif-else* con doppia parentesi quadra 333
22.37Esempio con costrutto *if-then-else* per verifica dell'interattività della shell 334

22.38 Verifica se la modalità è interattiva . . . 334
22.39 Esempio con costrutto *if-then-eif-else* . . . 334
22.40 Controlla il numero di argomenti . . . 334
22.41 Esecuzione dello script 22.40 . . . 335
22.42 Controlla l'*exit status* . . . 335
22.43 Costrutto generico *while* . . . 335
22.44 Esempio con costrutto *while* . . . 335
22.45 Esecuzione del *while* . . . 336
22.46 Costrutto generico *until* . . . 336
22.47 Esempio con costrutto *until* . . . 336
22.48 Esecuzione del *until* . . . 336
22.49 Costrutto generico *for* . . . 337
22.50 Esempio con costrutto *for* moderno . . . 337
22.51 Esecuzione del *for* . . . 337
22.52 Esecuzione del *for* infinito . . . 337
22.53 Costrutto generico *for* . . . 338
22.54 Esempio di *for* con sequenza . . . 338
22.55 Esempio di *for* con sequenza generata da un altro comando . . . 338
22.56 Esempio con costrutto *for* utilizzando `seq`† . . . 338
22.57 Esecuzione del *until* . . . 339
22.58 Stampa di array . . . 339
22.59 Stampa di array . . . 339
22.60 Stampa di array . . . 340
22.61 Array con i parametri passati . . . 340
22.62 Esecuzione di 22.60 . . . 340
22.63 Costrutto generico di una funzione . . . 340
22.64 Funzione per la somma dei parametri . . . 341
22.65 Esecuzione di **??** . . . 341
22.66 Esempio con costrutto *while* in modalità debug . . . 341
22.67 Esecuzione del *while* . . . 341
22.68 Intestazione generica tramite `env` . . . 342
22.69 Intestazione errata tramite `env` con parametri aggiuntivi . . . 342
22.70 Intestazione corretta tramite `env` con parametri aggiuntivi . . . 342
22.71 Script tradizionale del *fork bomb* in Bash. Nota: non eseguire!! . . . 343
22.72 Trascrizione del *fork bomb* in Bash. Nota: non eseguire!! . . . 343
23.1 Spezzone del file `/boot/grub/menu.lst` . . . 350
23.2 Spezzone del file `/boot/grub/grub.cfg` . . . 353
23.3 Esempio di variabili globali nel file `/boot/default/grub` . . . 354
23.4 `grep` su `/etc/inittab` . . . 357
23.5 Esecuzione di `runlevel` . . . 357
23.6 Switch di runlevel tramite `init 0` . . . 357
23.7 Switch di runlevel tramite `init 6` . . . 358
23.8 Esempio con il comando `shutdown` . . . 359
23.9 Sintassi generica del comando `shutdown` . . . 361
23.10 `shutdown` di default . . . 362
23.11 Notifica di `shutdown` . . . 362
23.12 Notifica di `shutdown` . . . 362
23.13 Cancellazione `shutdown` pendente . . . 362
23.14 Avviso di cancellazione `shutdown` pendente . . . 362

23.15 `shutdown` programmato con messaggio di broadcast 362
23.16 Esecuzione del comando `lsblk` 364
23.17 Esecuzione del comando `lsblk` 364
24.1 `dmesg` 368
24.2 `dmesg` con buffer sovrascritto 368
24.3 `dmesg -H` 369
24.4 `dmesg -T` 369
24.5 `dmesg --time-format iso` 369
24.6 `dmesg --decode` 370
24.7 `dmesg --facility kern` 370
24.8 `dmesg --level info` 370
24.9 `dmesg -C` 371
24.10 Spezzone del file `/etc/rsyslog.conf` 373
24.11 Spezzone del file `/etc/rsyslog.d/50-default.conf` 373
24.12 Spezzone di configurazione per inotrare i messaggi di log via *UDP* 375
24.13 Spezzone di configurazione per inotrare i messaggi di log via *TCP* 375
24.14 Spezzone di configurazione per inotrare i messaggi di log via *REPL* . . . 376
24.15 Effetto immediato del *logrotate* tramite `ls -l /var/log` 376
24.16 Spezzone del file `/etc/logrotate.conf` 377
24.17 Generico file di log 377
24.18 Generico file di log ruotato 377
24.19 Generico file di log ruotato due volte 377
24.20 Generico file di log ruotato due volte 377
24.21 File `/etc/logrotate.conf` 378
24.22 Alcuni file nella directory `/etc/logrotate.d` 379
24.23 File `/etc/logrotate.d/dpkg` 379
24.24 Esempi di `postrotate/endscript` 381
24.25 `journalctl` 383
24.26 `journalctl -r` 383
24.27 `journalctl -n 5` 383
24.28 `journalctl -n 5` 383
24.29 `journalctl --grep` 384
24.30 `journalctl --list-boots` 384
24.31 `journalctl --list-boots` 384
24.32 `journalctl --list-boots` 385
24.33 `journalctl --since` 385
24.34 `journalctl --since --until` 385
24.35 `journalctl -r -u ssh` 386
24.36 `journalctl -b _PID=1` 386
24.37 `journalctl -b _UID=1` 386
24.38 `journalctl --facility=help` 387
24.39 Lista delle *priorities* utilizzabili con `journalctl` 387
24.40 `journalctl --facility --priority` 387
24.41 `journalctl --priority range` 388
24.42 `journalctl -b -o short-iso` e `journalctl -o json-pretty` 389
24.43 `journalctl --disk-usage` e `journalctl -o json-pretty` 390
24.44 `journalctl --vacuum-size` e `journalctl --vacuum-time` 390
24.45 Invio di un messaggio di log tramite `logger` 391
24.46 Aggiunta nel file `/etc/rsyslog.d/50-default.conf` 391

24.47 Invio di un messaggio di log su una facility appena configurata 391
24.48 Invio di un messaggio di log su una facility appena configurata 392
24.49 Invio di un un messaggio di log su una facility non configurata 392
24.50 `sysctl` per utilizzare `dmesg` senza essere superutente 392
24.51 Verifica su `/dev/log` tramite `ls` 393
24.52 Verifica su `/dev/log` tramite `fuser` 393
24.53 Verifica su `/dev/kmsg` tramite `fuser` 393
24.54 Tipica chiamata della funzione `printk()` 393
24.55 Script Perl per la valutazione degli scarti tra prefissi omogenei 406
24.56 Output dello script dell'***Esempio 24.55*** 407

Capitolo 0

Premessa

QUESTO testo ha come condizione necessaria l'avere già installato una distribuzione GNU/Linux. Essendo lo strumento di lavoro la **CLI** (***c**ommand **l**ine **i**nterface* non ci saranno differenze nella trattazione per versioni server o desktop in quanto per quest'ultimo sarà completamente trascurato l'ambiente grafico, per cui per rapportarsi alla trattazione si dovrà aprire una *console*.

0.1 Convenzioni tipografiche

Nel testo non si è seguita una linea rigorosa, ma sommariamente le convenzioni tipografiche utilizzate sono riportate di seguito:

Grassetto: utilizzato per i titoli dei capitoli, sezioni, paragrafi e delle altre suddivisioni del documento, poco utilizzato per enfatizzare passaggi nel testo ad eccezione di termini chiave;

enfatizzato: diffusamente utilizzato per enfatizzare:

- le parole importanti o comunque da evidenziare rispetto al resto del testo (tendenzialmente parole o concetti nuovi);
- per i titoli di documenti o relativi autori o comunque personaggi da evidenziare rispetto al resto del documento;
- altre citazioni;

italics: anche *stile corsivo*, poco utilizzato come alternativa all'*enfatizzazione* precedente;

`typewriter`: anche `monospace`, utilizzato per i nomi di file o path, per comandi, opzioni, variabili o codice;

MAIUSCOLETTO: utilizzato quasi esclusivamente nelle tabelle per i nomi delle colonne;

$Sezione^{P}$: solitamente il simbolo ^{P} è utilizzato in corrispondenza di sezioni[1] le quali sono fortemente caratterizzate da passaggi di programmazione;

$Sezione^{\dagger}$: solitamente il simbolo † è utilizzato in corrispondenza di sezioni[2] le quali riportano passaggi che rientrano nelle seguenti categorie:

[1] o comunque segmenti di testo quali capitoli o paragrafi ben isolati

[2] v. nota precedente

- ancora utilizzati ma non più di uso comune;
- tradizionalmente utilizzati ma la tendenza è verso soluzioni alternative;
- deprecati ma è possibile trovare tracce storiche.

Talvolta sono sommati due effetti tipografici, ad esempio la sigla inventata **QEUA** è l'acronimo di ***q****uesto* ***è*** ***u****n* ***a****cronimo*.
Per ulteriori convenzioni si consulti [Cev06].

Parte I

Base

Capitolo 1

Introduzione

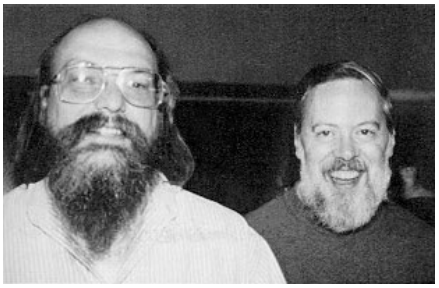
QUESTA sarà una brevissima introduzione per affrontare in maniera estremamente semplificata la nascita, il tema delle licenze e dell'installazione del sistema. In una futura revisione le sezioni saranno sicuramente ampliate.

1.1 Storia

La storia dietro il SO GNU/Linux è formata da tantissimi passaggi, importanti contributi e personaggi eccezionali. Di seguito ne riporto solo alcuni, ma invito il lettore ad approfondire da ulteriori fonti alcune saranno citate nel capitolo.

1.1.1 Unix

Dopo l'abbandono da parte di *AT&T*[1] del progetto *Multics*[2], fu creato il SO Unix[3] da un gruppo di ricerca presso i laboratori *AT&T* e *Bell Laboratories*. Il team includeva *Ken Thompson* e *Dennis Ritchie* che lo rilasciarono nel 1970. Per renderlo portabile su più



(a) Ken Thompson e Dennis Ritchie nel 1973

(b) Logo del SO Unix

architetture, successivamente riscrissero il sistema nel linguaggio *C*.

[1] società statunitense è una multinazionale colosso delle telecomunicazioni

[2] ***MULT****iplexed* ***I****nformation and* ***C****omputing* ***S****ervice* è stato un SO che introdusse molti concetti ed influenzò lo sviluppo di SO time-sharing

[3] ufficialmente registrato come **UNIX**

Basato sulla sesta edizione di Unix di AT&T, nel 1977 fu sviluppato il **BSD**[4] dal **CSRG**[5] dell'*UC Berkeley*.
Poiché *BSD* conteneva codice dello Unix di proprietà di AT&T, all'inizio degli anni '90

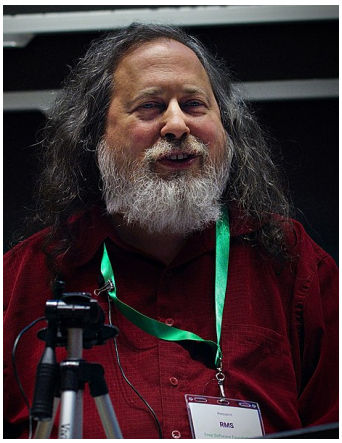


Figura 1.1: *Logo di BSD*

quest'ultima intentò una causa contro l'Università della California e ciò ha certamente rallentato lo sviluppo e l'adozione di BSD.

1.1.2 GNU

Lo *GNU project* è un progetto collaborativo per lo sviluppo di **free software**, iniziativa nata il 27 settembre 1983 da *Richard Stallman* con l'obiettivo della creazione di **GNU**[6], un



(a) Richard Matthew Stallman nel 2019



(b) Logo del progetto GNU

sistema operativo *Unix-like* completo utilizzando esclusivamente *software libero*.
Il free software[7] ha proprio l'obiettivo indicato dall'espressione, assegnare agli utilizzatori la libertà e il controllo del propri computer e device attraverso lo sviluppo collaborativo e la pubblicazione di software che assegni a ciascuno i diritti[8] di eseguire, copiare e distribuire, studiare e modificare liberamente il software medesimo.

[4] ***B**erkeley **S**oftware **D**istribution*
[5] ***C**omputer **S**ystems **R**esearch **G**roup*
[6] ***G**NU's **N**ot **U**nix*, acronimo ricorsivo
[7] una delle frasi più ricorrenti tra i sostenitori del *software libero* è *"Free as in Freedom, not as in Beer"*
[8] ottimo testo divulgativo per approfondire tali tematiche è [DM99]

Tornando allo sviluppo del SO GNU, poiché in quel periodo Unix era diffuso e certamente più apprezzato rispetto alle concorrenza[9], si decise che sarebbe stato un sistema operativo simile a Unix.
Nel 1992 il progetto GNU aveva già creato le principali utilità del SO[10], ma non aveva completato il *kernel*.
Il progetto software denominato *GNU Hurd* è un insieme di componenti software che comprende un *microkernel* denominato *GNU Mach*, in riferimento all'omonimo microkernel *Mach* per sistemi paralleli e distribuiti sviluppato dalla *Carnegie Mellon University*. GNU Hurd si rivelò un progetto molto complesso e si dovette attendere qualche decennio prima di ottenere un'implementazione, anche se va tenuto presente che le priorità di sviluppo del software era minore poiché sono emerse altre soluzioni. Evento importante va segnalato

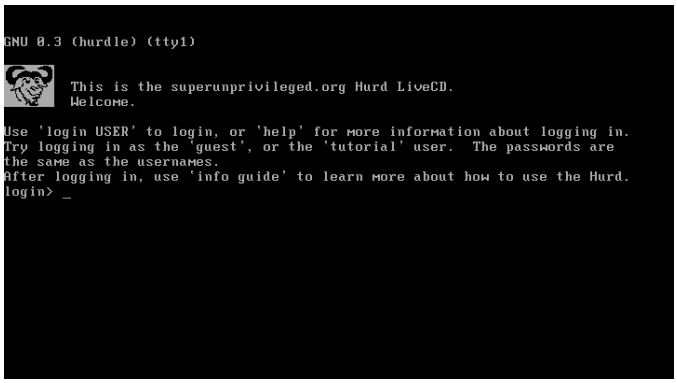


Figura 1.2: *HURD Live CD*

che nel 1985 fu creata la *Free Software Foundation*, un ente di supporto al progetto GNU finalizzato a supportare lo sviluppo di software libero, promuovendo diverse iniziative tra le quali l'indicazione aggiornata di una lista dei *high-priority projects*, dichiarando che *"there is a vital need to draw the free software community's attention"*, poiché li considera *"important because computer users are continually being seduced into using non-free software, because there is no adequate free replacement"*.

1.1.3 Linux

Nel 1991 fu pubblicata la prima versione funzionante del *kernel monolitico* ideato da Linus Torvalds che denominò *Freax*, in assonanza con *free* e la *x* di Unix.
In particolare lo studente universitario in *computer science* all'*University of Helsinki* era insoddisfatto di *MINIX*, il SO Unix-like sviluppato da *Andrew S. Tanenbaum* per insegnamenti di Unix ai corsi di *Computer networks* e *Operating systems* presso la *Vrije Universiteit Amsterdam*, proprio per esemplificare i principi esaminati nel suo libro di testo *Operating Systems: Design and Implementation*. In particolare era disponibile il codice sorgente del sistema, ma la licenza non permetteva modifiche o redistribuzioni. Inoltre, il MINIX era progettato per un'architettura a 16 bit, per cui non si adattava alla nuova architettura a 32 bit che si stavano diffondendo con i pc Intel 386.
Torvalds decide quindi di realizzare un proprio kernel con l'obiettivo di esplorare tutte le

[9] ad esempio, lo stesso Richard Stallman commentò che considerava l'*MS-DOS "a toy"*
[10] ad esempio il *compilatore C* denominato `gcc`, la shell, editor di testo

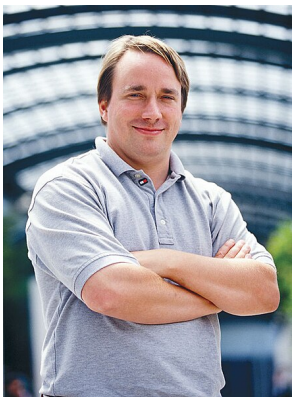
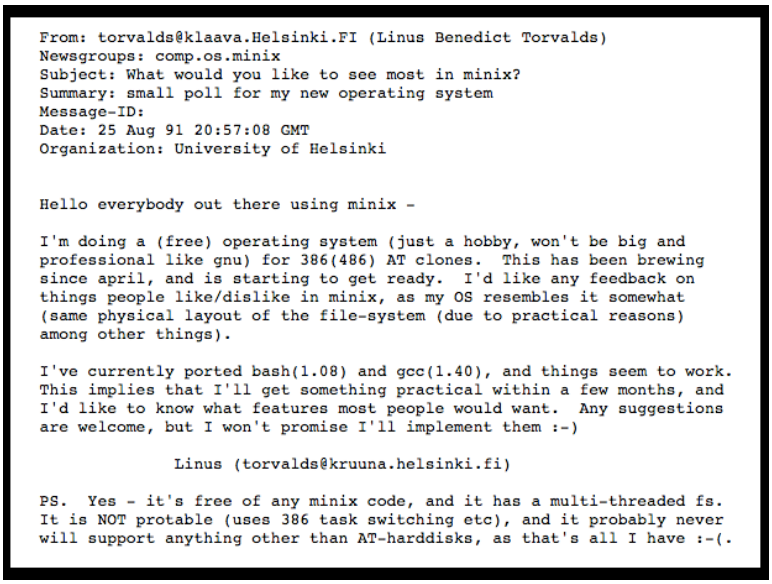


Figura 1.3: *Linus Torvalds nel 2002*

potenzialità offerte dall'hardware a sua disposizione e realizza un emulatore di terminale[11] eseguibile senza SO che permetteva due operazioni: inviare alla porta seriale e ricevere dalla stessa segnali elettrici attraverso due processi separati.
Il 3 luglio 1991, cercando di implementare le *Unix system calls* nel progetto, prova ad ottenere una copia digitale della documentazione *POSIX standards* tramite una richiesta al newsgroup `comp.os.minix`. Non avendo successo nella ricerca di tale documentazione, Torvalds inizialmente utilizzò la documentazione SunOS disponibile nella propria università, documentazione per l'OS Sun Microsystems. Altre system calls le apprese dal precedente testo citato di Tanenbaum adottato nel corso di Unix. Da un emulatore, i contorni del nuovo software si stavano trasformando proprio in quelli di un kernel[12].
Il 25 agosto 1991 Torvalds rende pubblico il progetto sul già citato newsgroup dedicato a MINIX, `comp.os.minix`, scrivendo il famoso messaggio del 25 agosto 1991 riportato in ***Figura 1.4*** che segna ufficialmente la nascita di Linux.
La prima versione *v. 0.01* è resa disponibile dal 17 settembre 1991, insieme alla shell *GNU Bash*. Il software realizzato è però ancora sviluppato su MINIX e compilabile attraverso il compilatore *GNU C*. Torvalds continua a sviluppare il progetto e dalla *v. 0.11* non ha più necessità esterne, per cui il software diviene propriamente un kernel sviluppato sulle specifiche POSIX.
Con i successivi rilasci, anche il nome ufficiale del progetto cambia nell'attuale. Infatti, per facilitare lo sviluppo, nel settembre 1991 i file furono spostati su un server FTP[13] della propria università (*Helsinki University of Technology*) e *Ari Lemke*, assistente e amministratore del server, rinominò (senza preventivamente chiederlo) la directory in *Linux*[14] pensando che l'originale non fosse un buon nome[15] per il progetto.
Successivamente all'annuncio di intenzione del cambio di licenza in una nota della release *v. 0.12*, l'avvenimento avvenne a metà dicembre 1992 in cui Torvalds pubblicò la *v. 0.99* utilizzando la GNU GPL la quale non poneva come la precedente restrizioni per uso commerciale, e questa nuova caratteristica condusse ad un'inaspettata evoluzione che

[11] scritto in C e assembler
[12] v. [TD02]
[13] `ftp.funet.fi`
[14] ***Linus Unix***
[15] Torvalds aveva già considerato tale nome, ma inizialmente lo scartò poiché proiettava un'immagine di forte egocentrismo. Successivamente all'operazione di *rename*, il nome fu accettato



```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID:
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki


Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones.  This has been brewing
since april, and is starting to get ready.  I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
This implies that I'll get something practical within a few months, and
I'd like to know what features most people would want.  Any suggestions
are welcome, but I won't promise I'll implement them :-)

                Linus (torvalds@kruuna.helsinki.fi)

PS.  Yes - it's free of any minix code, and it has a multi-threaded fs.
It is NOT protable (uses 386 task switching etc), and it probably never
will support anything other than AT-harddisks, as that's all I have :-(.
```

Figura 1.4: *Il messaggio di Linus Torvalds che annuncia la nascita di Linux*

successivamente Torvalds definì *"making Linux GPLed was definitely the best thing I ever did"*, evoluzione che sarà ripresa nella prossima sezione.
Gli sviluppi si susseguirono, ma è interessante un altro passaggio non tecnico ma di immagine: nel 1996 Torvalds annuncia una mascotte per Linux: un pinguino[16] denominato *Tux*[17] visibile in ***Figura 1.5***.



Figura 1.5: *Tux, la mascotte di Linux*

1.1.4 GNU/Linux

Da 1.1.2 e 1.1.3 è facile intuire una convergenza tra il progetto GNU e il progetto Linux. Ripercorrendo brevemente gli avvenimenti, nei primi anni '90 al *progetto GNU* mancava un tassello fondamentale per la realizzazione di un SO completo secondo il vincolo autoimposto che *tutti i software fossero liberi*. Parallelamente allo sviluppo dell'incompleto

[16] nel periodo in cui si cercava una mascotte, Torvalds raccontò di essere stato morso da un piccolo pinguino (*Eudyptula minor*) durante una visita allo zoo e acquario nazionale di Canberra, in Australia. Larry Ewing fornì la bozza originale

[17] il nome fu suggerito da James Hughes derivante da ***T**orvalds' **UniX***

kernel del progetto GNU, il *progetto Linux* aveva rilasciato nel dicembre '92 la release *v. 0.99* con la nuova licenza *GNU GPL*, per cui la lacuna precedente si poteva *facilmente* colmare.
Gli sviluppatori dei relativi progetti Linux e GNU lavorarono per integrare i componenti GNU sul kernel Linux per creare un OS pienamente funzionante e libero con il nome combinato GNU/Linux.

Ritengo doverosa una precisazione: il nome combinato *GNU/Linux* non è spesso utilizzato in favore del semplice *Linux*, ma personalmente lo considero non corretto e preferisco adottare il nome *Linux* solo per l'identificazione esclusiva del kernel, quindi per riferirmi all'intero SO utilizzo GNU/Linux conformandomo a parte della community ma soprattutto da quanto scritto nei software e nella documentazione prodotti dai progetti *GNU* e *Debian*.

1.2 Licenze

Senza entrare nel dettaglio che ha scatenato/scatena/scatenerà lunghi dibattiti, il software alla base del mondo GNU/Linux è spesso software *FOSS*, acronimo di ***F**ree and **o**pen-**s**ource **s**oftware*, termine che si riferisce ad una classe di software sia *free software* sia *open-source software*, in cui ciascuno è libero di usare, copiare, studiare a modificare il software in ogni sua parte, e il codice sorgente è aperto per la condivisione cosicché altri sviluppatori siano incoraggiati a migliorare lo stesso.

Il software GNU permette tali diritti esplicitandoli nella licenza e, tra le molte licenze variamente compatibili con tale filosofia, quella più conosciuta è proprio la *GNU GPL*[18], spesso semplicemente ***GPL***, una licenza libera ed estremamente *copyleft*[19] la quale protegge le *libertà fondamentali*. Originariamente stesa nel 1989 da Richard Stallman per patrocinare i programmi creati per il sistema operativo GNU, in maniera un po' sommaria, a differenza di altre licenze libere non-copyleft, un'opera protetta da GNU GPL deve rimanere libera.
L'ultima versione della GPL è la 3, il cui logo è visibile nella ***Figura 1.6***.



Figura 1.6: *Logo della licenza ufficiale della* GNU General Public License *versione 3*

Oltre la precedente *GNU GPL*, sono state introdotte altre licenze divenute famose:

[18] ***GNU General Public License***
[19] oltre il precedente, ottimo testo relativo alla trattazione delle tematiche è [Les05]

GNU FDL, la ***GNU Free Documentation License***, scritta per essere adottata dalla documentazione prodotta;

GNU LGPL, la ***GNU Lesser General Public License***, adottata in particolar modo dalle librerie software, creata per ragioni di compatibilità e portabilità in quanto permette una maggior integrazione del software libero all'interno di software proprietario.

Da queste licenze traspare un principio filosofico con il quale sono fortemente identificato: sono estremamente convinto che il mondo sia un posto ***molto migliore*** *(più efficiente, più sicuro, più interessante, ...)* se si ha a disposizione il codice sorgente di tutto il software installato nel proprio computer, il diritto di poterlo usare e gestire totalmente in maniera costruttiva.

1.3 Installazione

Non ci sono particolari indicazioni, ma alcuni consigli:

- non avere un sistema GNU/Linux installato in *dual boot*;
- non eseguire un environment quale *Cygwin*, *WSL* o altro.

Se pensate siano condizioni da integralista, probabilmente avete ragione, ma per apprezzare il sistema nella sua completezza, ci si dovrà immergere totalmente, le mezze misure non sono accettabili a meno che non vogliate al più intereagire rispettivamente saltuariamente o con una installazione surrogata.

1.4 Distribuzioni

Non ci sono indicazioni al riguardo, ma il testo avrà come riferimenti principali la *Debian* (equivalentemente le derivate quali *Ubuntu*, *Kali Linux*, . . .) e la *Red Hat Enterprise Linux* (equivalentemente le derivate quali *AlmaLinux*, *ClearOS*, *Rocky Linux*, . . .).

Simpatie dell'autore Ho forte simpatie per le distribuzioni *Debian/Ubuntu* e, se non vincolato da questioni lavorative, utilizzo *Debian* e *Ubuntu LTS*[20].

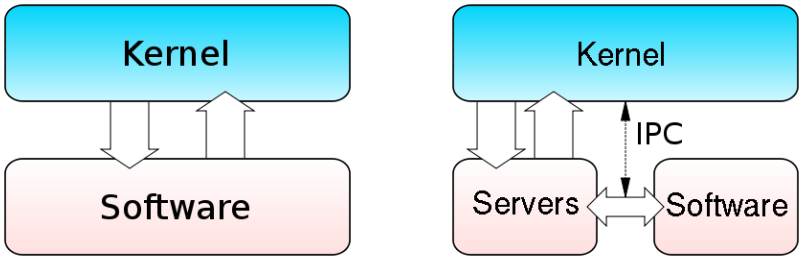
1.5 Approfondimenti

1.5.1 Kernel monolitico vs microkernel

Dal punto di vista storico è interessante il dibattito sviluppato da *Andrew S. Tanenbaum* e *Linus Torvalds* riguardanti il kernel di Linux e in generale l'architettura di un kernel.

[20] **LTS** è l'acronimo di ***l**ong **t**erm **s**upport* per le quali la società *Canonical Ltd.* promette per le LTS 5 anni di supporto per le versioni server dal rilascio delle stesse (frequenza di rilascio delle versioni LTS è indicativamente ogni 2 anni nel mese di aprile degli anni pari, per i rilasci in generale uno ogni 6 mesi), quindi la sequenza è stata *6.06 LTS Dapper Drake* la prima, ma poi regolarmente *8.04 LTS Hardy Heron*, *10.04 LTS Lucid Lynx*, *12.04 LTS Precise Pangolin*, *14.04 LTS Trusty Tahr*, *16.04 LTS Xenial Xerus*, *18.04 LTS Bionic Beaver*, *20.04 LTS Focal Fossa*, *22.04 LTS Jammy Jellyfish* e *24.04 LTS Noble Numbat*

Il prof. Tanenbaum creatore di MINIX iniziò il dibattito[21] nel 1992 su *Usenet*, sul già citato gruppo `comp.os.minix`, sostenendo che i microkernel fossero superiori ai kernel monolitici, e quindi Linux già era nel 1992 obsoleto. Il dibattito è stato considerato da molti una *flame war*[22].
Il dibattito iniziò il 29 gennaio 1992 quando Tanenbaum pubblicò le sue critiche al kernel Linux in un post intitolato *"LINUX è obsoleto"*, sottolineando come il design monolitico fosse dannoso. Inizialmente non spiegò il motivo per cui ritenesse il microkernel migliore, ma suggerì questioni di portabilità in quanto il kernel Linux fosse troppo legato all'architettura dei processori x86 per essere utilizzabile nel futuro, quindi l'evoluzione hardware ne suggeriva la sostituzione. Per chiarire la prospettiva, affermò che scrivere un kernel monolitico nel 1991 è *"a giant step back into the 1970s"*.
Essendo la critica postata in un newsgroup pubblico, Torvalds replicò direttamente qualche giorno dopo, sostenendo che MINIX aveva delle lacune di progettazione intrinseche (citando la mancanza del *multithreading* come esempio specifico), ma riconobbe che il design di un microkernel è superiore *"from a theoretical and aesthetical"* punto di vista. Inoltre affermò che, poiché sviluppava il kernel Linux nel tempo libero e lo distribuiva gratuitamente (al tempo MINIX non era gratuito), Tanenbaum non avrebbe dovuto effettuare simili obiezioni. In ogni caso menzionò che il suo sviluppo era intenzionalmente per l'Intel 80386 per esercizio di apprendimento per se stesso; ammettendo che ciò rendeva il kernel sviluppato meno portabile di MINIX, il principio progettuale era accettabile poiché rendeva l'API più semplice e più portabile. Per tale ragione, affermò *"linux è più portabile di minix"*.



(a) Un kernel monolitico definisce un'interfaccia virtuale di alto livello sull'hardware con un set di primitive o chiamate di sistema per implementare servizi di SO

(b) Un microkernel è un kernel ridotto che fornisce un analogo set di primitive o chiamate di sistema ma implementa servizi minimali del SO; le altre funzionalità del kernel saranno svolte da software separati dal microkernel, e tali server saranno eseguiti sul microkernel comunicando con quest'ultimo tramite chiamata

Figura 1.7: *Distinzione tra kernel monolitico vs microkernel*

A seguito della replica di Torvalds, Tanenbaum ha sostenuto che i limiti di MINIX sono dovuti alla considerazione che, essendo un professore, il sistema concepito sarebbe dovuto

[21] è riportato in appendice A di [DM99], consultabile anche online [DM]
[22] discussione oltre i confini oggettivi e spesso con toni da *tifoseria* per determinate opinioni

essere possibile eseguirlo su hardware modesto per uno studente medio, quindi un computer basato su Intel 8088 talvolta senza hard disk, invece Linux era stato sviluppato per l'Intel 386, un processore significativamente più potente e costoso. Inoltre affermò: *"... as of about 1 year ago, there were two versions, one for the PC (360K diskettes) and one for the 286/386 (1.2M). The PC version was outselling the 286/386 version by 2 to 1"*.
Aggiunse che, anche se Linux era gratuito, non sarebbe stata un'alternativa praticabile per i suo studenti poiché non avrebbero potuto sostenere i costi relativi all'hardware richiesto, invece MINIX poteva essere eseguito su *"a regular 4.77 MHz PC with no hard disk"*.

Inoltre di seguito riporto alcuni passaggi di Tanenbaum che a distanza di anni fanno sorridere:

> *I still maintain the point that designing a monolithic kernel in 1991 is a fundamental error. Be thankful you are not my student. You would not get a high grade for such a design :-)*

e

> *Writing a new OS only for the 386 in 1991 gets you your second 'F' for this term. But if you do real well on the final exam, you can still pass the course.*

Intervenne un altro utente del gruppo per replicare a Tanenbaum sostenendo *"... an explicit design goal of Linux was to take advantage of the special features of the 386 architecture. So what exactly is your point? Different design goals get you different designs"*.
In ogni caso Torvalds tentò di terminare la discussione affermando che non avrebbe dovuto reagire in modo eccessivo alle dichiarazioni iniziali di Tanenbaum, e che gli stava già scrivendo una email per scusarsi.

1.6 Concetti chiave

copyleft: modello di gestione dei diritti d'autore basato su un insieme di licenze attraverso le quali l'autore indica, ai fruitori dell'opera, che essa può essere utilizzata, diffusa e talvolta anche modificata liberamente, nel rispetto delle condizioni specificate nella licenza stessa

distribuzione: è un sistema GNU/Linux (completo di librerie e software vari) supportato da un fornitore o da una community, ed evolve nel tempo di seguito all'evoluzione dei vari software supportati

FOSS: ***F****ree and* ***o****pen-****s****ource* ***s****oftware* è un termine che si riferisce ad una classe di software che è sia *free software* sia *open-source software*, in cui ciascuno è libero di usare, copiare, studiare a modificare il software in ogni sua parte, e il codice sorgente è aperto per la condivisione

free software: software che assegnare agli utilizzatori la libertà di eseguire, copiare e distribuire, studiare e modificare liberamente il software

GNU GPL: è una licenza copyleft per software libero, inizialmente creato per patrocinare i programmi sviluppati per il sistema operativo GNU

kernel monolitico: software che definisce un'interfaccia virtuale di alto livello sull'hardware con un set di primitive o chiamate di sistema per implementare servizi di SO

microkernel: kernel ridotto che fornisce un analogo set di primitive o chiamate di sistema ma implementa servizi minimali del SO; le altre funzionalità del kernel saranno svolte da software separati dal microkernel, e tali server saranno eseguiti sul microkernel comunicando con quest'ultimo tramite chiamata

multithreading: nel contesto dei SO è una tecnica utilizzata per permettere ad un singolo processo di avere più thread di esecuzione

POSIX: o **Posix**, acronimo di ***P****ortable* ***O****perating* ***S****ystem* ***I****nterface for Uni****x***, nome di successo coniato da Richard Stallman per indicare la famiglia degli standard definiti nell'*IEEE 1003* la quale formalizza alcuni standard delle API per i software sviluppati per le diverse varianti del SO Unix

software libero: v. **free software**

Capitolo 2

Shell

PRIMA di accedere ad un sistema GNU/Linux è necessario eseguire il *login*, dopodiché *solitamente*[1] si verrà proiettati in un *environment* denominato *shell*.

2.1 Login e logout

2.1.1 Login

Tramite la procedura di *login*[2] il sistema permette ad un utente[3], qualora la procedura di autenticazione sia andata a buon fine, di interagire con il medesimo sistema[4]. L'esito positivo è sintetizzato con l'espressione *autenticazione avvenuta con successo*, e successivamente alla stessa sarà aperta una *sessione di sistema*. Nella casistica più comune, la procedura di autenticazione provede la richiesta dal sistema delle credenziali quali una coppia di *username* e *password* come nell'***Esempio 2.1***.

```
1 login: ms
2 password:
3 $
```

Esempio 2.1: *Procedura di login con successo*

in cui il sistema risponderà con un prompt di attesa comandi, genericamente usato il carattere `$`[5].
Nel caso l'autenticazione non avvenga con successo, generalmente si potrà riprovare con un'eventuale *penalità*[6] per un numero di volte che dipenderà dalla rigidità delle politiche di autenticazione.
Nell'esempio di seguito, dopo il tentativo di immissione credenziali errato, il sistema attenderà un paio di secondi per restituire il messaggio di errore `Authentication`

[1] per quantificare l'avverbio *solitamente* sarà necessario attendere fino alla sezione 17.4.1 a pag. 218

[2] attuata dal software `login`

[3] più precisamente ad un'entità generica, quindi anche un software

[4] o con altri sistemi ai quali solitamente fornirà la prova tramite un *token*

[5] carattere *dollaro*

[6] spesso, dopo alcuni tentativi, il sistema stabilisce un tempo da dover attenedere prima di poter tentare nuovamente la procedura di autenticazione

```
1 login: ms
2 password:
3 Authentication failure
4
5 login:
```

Esempio 2.2: *Procedura di login con credenziali errate*

`failure`, cosicché solo successivamente si potrà procedere con nuovo un nuovo tentativo di login: come nell'***Esempio 2.2***.

2.1.2 Logout

Dopo una procedura di login corretta, si potrà procedere al logout con diverse alterative[7]:

```
1 $ logout
```

Esempio 2.3: *Comando logout*

oppure

```
1 $ exit
```

Esempio 2.4: *Comando exit*

oppure

```
1 $  Ctrl + D
```

Esempio 2.5: *Logout tramite la combinazione di tasti* Ctrl + D

Simpatie dell'autore Quasi sempre utilizzo la combinazione di tasti Ctrl + D .

2.2 Cosa è una shell?

La shell è la componente software che fornisce l'interfaccia all'utente per l'interazione con il SO. Essa è l'*interprete dei comandi*, quindi è possibile impartire comandi e richiedere l'avvio di programmii. Per ovvi motivi di seguito saranno esaminate solo le shell via CLI.

2.2.1 Shell popolari

Si differenziano tra loro essenzialmente per espressività, quindi ricchezza di comandi interni e flessibilità di interpretazione.

[7] sono alternative in quanto conducono tutte al *logout*, ma la sequenza operativa è differente e sarà approfondita nella sezione 15.1 a pag. 187

Bourne shell La *Bourne shell* (`sh`) è probabilmente la più popolare shell tra i sistemi *Unix*, nata per sostituire la precedente *Thompson shell*[8], la prima shell Unix introdotta nel 1971. L'autore *Stephen L. Bourne*[9] inizio lo sviluppo nel 1976 per poi rilasciarla nel 1979, fornita come shell di default della versione 7 di Unix. Anche se più primitiva delle varianti moderne, ancora oggi molti sistemi la popongono tra le varie shell disponibili oppure, come nel caso del sistema in cui scrivo, per retrocompatibilità c'è un *link simbolico*[10].

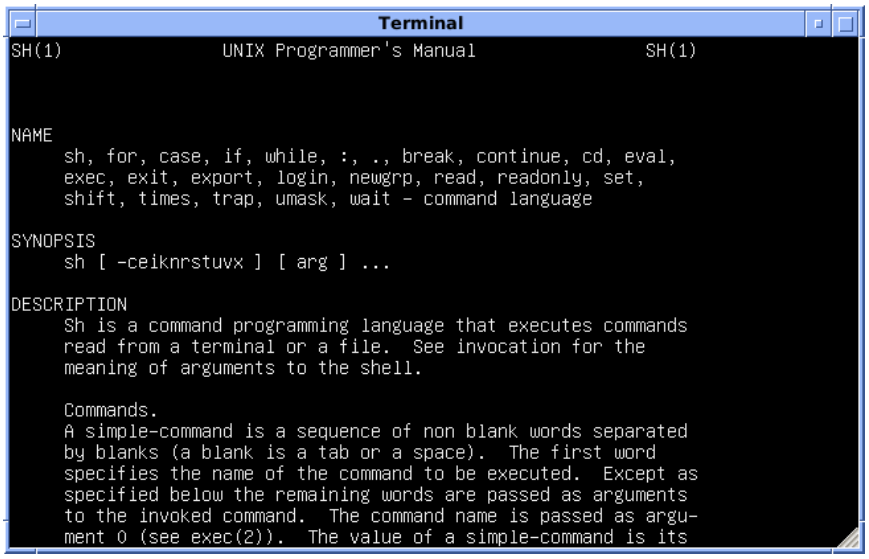


Figura 2.1: *L'originale pagina di manuale della* Bourne shell

C shell e Korn shell La *C shell* (`csh`), creata all'University of California in Berkeley da *Bill Joy* nel 1978 sotto *BSD license*, ha qualche espressiviò del linguaggio di programmazione *C*.
La *Korn shell* (`ksh`) è un'altra Unix shell sviluppata da *David Korn* ai Bell Labs inizi anni '80 e rilasciata nel 1983. Inizialmente il progetto si basava sul codice sorgente della Bourne shell, successivamente ha mantenuto la retrocompatibiltà con la Bourne Shell e si è arricchito di molte caratteristiche della C shell.

Bourne-again shell La ***B**ourne **A**gain **SH**ell*, popolarmente conosciuta con l'aconimo `bash`, è una shell sviluppata da *Brian Fox* and *Chet Ramey* nell'ambito del progetto *GNU* della *Free Software Foundation*[11] come alternativa libera della Bourne shell.
Dal punto di vista tecnico è *superset* della *Bourn shell*, fonde molte funzioni della C shell e della Korn shell. Essa è presente im molti sistemi Unix-like[12] e nell'ambiente *WSL*[13].
Aneddoticamente ha anche un nome ha successo, soprattutto per l'omofono a *born again*[14],

[8] il cui autore è stato Ken Thompson
[9] ovviamente da lì il nome
[10] i link saranno analizzati nella sezione 4.6.2 a pag. 43
[11] quindi sotto licenza GNU GPL
[12] *GNU/Linux*, *FreeBSD*, *NetBSD*, *macOS*, *Solaris*, *MINIX*, ...
[13] *Windows Subsystem for Linux*
[14] traducibile con il termine *rinascita*

Si può considerare sicuramente la più utilizzata nei sistemi GNU/Linux, è la shell utilizzata per la stesura di questo documento.

2.2.2 Compiti della shell

TODO

2.3 Comandi

Indipendentemente dal SO, sommariamente le fasi di interazione con una shell si possono riassumere di seguito:

1. attende l'input dell'utente collegato;
2. la shell esegue il comando.

Generalmente una shell mostra un *prompt* per indicare che è in attesa di un comando da parte dell'utente. Il prompt può essere variamente esplicativo, ma solitamente mostra il nome dell'utente collegato, il nome dell'*host*, la directory corrente e un carattere finale come di seguito:

```
1 ms@nb-m:/home$
```

Esempio 2.6: *Prompt dei comandi*

Nell'esempio, l'utente `ms` è sull'host `nb-m` nella directory `/home`. Di seguito sarà solitamente utilizzato un prompt semplificato come nel'esempio che segue:

```
1 $
```

Esempio 2.7: *Prompt dei comandi semplificato*

2.3.1 Struttura dei comandi

Ad esclusione di sequenze particolari, un comando è una sequenza di caratteri che terminano con la pressione del tasto Enter [15]. Affinché la shell possa interpretare correttamente quanto immesso dall'utente, la linea di comando fornita dovrà rispettare una sintassi. In particolare la shell segmenterà la linea in *parole* utilizzando come carattere di separazione tra esse gli spazi. Solitamente la prima parola sarà il comando[16], le successive saranno i *parametri* che ne determineranno in dettaglio il compito.
I parametri si possono suddividere in due tipi:

opzioni:
- iniziano con il carattere `-` nella versione breve[17] oppure con i due caratteri in sequenza `--` nella versione estesa ed esplicativa[18];
- sono solitamente opzionali;

[15] per linee di comando lunghe, spesso si preferisce spezzarle inserendo al termine della linea il token `\` e premendo Enter, solo all'ultima linea semplicemente il comando Enter. Dopo l'inserimento di quest'ultima linea e del carattere Enter (quindi linea che non termina con il token `\`), la shell concatenerà tutte le linee precedenti e valuterà il comando

[16] le eccezioni saranno fornite nella sezione 13.1 a pag. 155

[17] ad esempio `-a` oppure `-l` oppure `-R`

[18] ad esempio `-all` oppure `--no-group` oppure `--human-readable`

- spesso con valore *on* oppure *off*;
- se si utilizza la versione breve di un solo carattere, sono opzionalmente cumulabili in sequenza tutti dietro il carattere –[19];
- solitamente talune combinazioni sono per convenienza mappate in una versione alternativa[20].

argomenti: sono i parametri senza l'iniziale carattere –, spesso sono obbligatori o comunque se non passati è assegnato un valore di default

2.3.2 Tipi di comando

I comandi si suddividono in due categorie:

interni: sono forniti dalla stessa shell

esterni: non sono forniti dalla shell, per cui essa dovrà consultare l'environment esterno (tipicamente il comando sarà eseguito lanciando un omonimo software

Ovviamente la shell cercherà di interpretare il comando immesso dall'utente prima tramite i comandi interni e, se non riesce, con i comandi esterni. Ciò implica che, qualora dovesse esistere un comando esterno omonimo di un comando interno, per utilizzarlo l'utente dovrà fornire un contesto ulteriore affinché la shell non interpreti quanto immesso come interno, ma consulti l'environment esterno[21].
Ad ogni modo, è possibile richiede a `bash` se un comando sarà interpretetato internamente o esternamente attraverso il comando `type` passando come argomento proprio l'eventuale comando del quale l'utente è dubbioso:

```
1 $ type cd
2 cd is a shell builtin
3 $ type date
4 date is /usr/bin/date
5 $ type ciao
6 bash: type: ciao: non trovata
```

Esempio 2.8: *Comando* `type`

in cui è evidente che:

`cd`[22] è interno alla shell;

`date`
è esterno per cui sarà chiamato il software indicato nel percorso;

`ciao` non è un comando riconosciuto.

Ripetendo equivalentemente l'***Esempio 2.8*** nell'***Esempio 2.9***, si può notare che, per i comandi esterni, il comando `type` restituiste l'equivalente del comando `which`, il quale ha esattamente lo scopo di indicare la localizzazione di un comando all'interno del sistema.

Ora si esamini l'***Esempio 2.10***.

[19] per esempio è possibile sostituire la sequenza di opzioni `-a -l -R` con `-alR`

[20] per esempio potrebbe essere possibile sostituire una sequenza opzionale del tipo `-arTS` con la sola opzione `-o` oppure `--opt` che sarebbe certamente un conveniente mappaggio

[21] questo passaggio sarà esemplificato nell'***Esempio 2.10***

[22] il comando `cd`
sarà esaminato alla sezione 4.3.2 a pag. 35

```
1 $ which cd
2 $ which date
3 /usr/bin/date
4 $ which ciao
5 $
```

Esempio 2.9: *Comando* `which`

```
1 $ type echo
2 echo is a shell builtin
3 $ which echo
4 /usr/bin/echo
```

Esempio 2.10: *Comando* `which`

Chiaramente si nota l'output discordante ma, per quanto scritto ad inizio sezione, appare chiaro che si tratta di un'omonimia tra comandi interni ed esterni, per cui sarà richiamata la funzionalità interna. Qualora l'utente voglia richiamare l'omonimo comando indicato da `which`, potrà richiarmare tale software indicando esplicitamente il percorso[23].

Può essere interessante esaminare i comandi precedenti `echo` e `date`.

Comando `echo` In sintesi semplicemente replica quanto scritto:

```
1 $ echo sabato e domenica
2 sabato e domenica
```

Esempio 2.11: *Comando* `echo`

Si esaminerà ulteriormente questo comando alla sezione **??** a pag. **??**

Comando `date` Mostra la data, orario e time zone correnti:

```
1 $ date
2 dom 22 ott 2023, 00:34:19, CEST
```

Esempio 2.12: *Comando* `date`

Si esaminerà ulteriormente questo comando alla sezione **??** a pag. **??**

Tornando al comando `type`, è utile consultare la guida fornita dal comando `help`[24] come mostrato nell'***Esempio 2.13***.

2.3.3 Metacaratteri ed escaping

Le shell caratterizzano alcuni caratteri immessi in modo particolare, ad esempio uno di questi è il carattere [ ][25], cui si assegna il nome *metacaratteri* per indicare che per loro l'interpretazione non sarà letterale. Nell'***Esempio 2.14*** sono mostrati altri caratteri che `bash` tratta come caratteri speciali.

[23] ovviamente in altre shell il comando potrebbe non essere interno, per cui per quella shell sarà sicuramente utilizzata la variante esterna

[24] questo comando sarà esaminato nella sezione 3.1 a pag. 23

[25] carattere spazio

```
1 $ help type
2 type: type [-afptP] nome [nome ...]
3     Visualizza le informazioni sul tipo di comando.
4
5     Per ogni NOME, indicare come verrebbe interpretato se utilizzato come
      a
6     nome del comando.
7
8     Opzioni:
9 ...
```

Esempio 2.13: *Comando* `help`

```
1 $&;(){}[]*?!<>"'
```

Esempio 2.14: *Caratteri speciali*

Per utilizzarli è possibile:

- farli precedere singolarmente dal carattere \[26] che ha esattamente lo scopo di eseguirà l'*escape* del carattere speciale che segue;
- racchiudere tutta la sequenza con due caratteri `'`[27] oppure `"`[28], come ad esempio nelle sequenze `'&&!'` oppure `"%&/"`. Ulteriori esempi ci saranno nella sezione 4.1.1 a pag. 31.

È utile sapere che c'è una differenza[29] tra i caratteri `'` e `"`, in particolare se la stringa è tra `'` sarà trattata letteralmente, se tra `"` saranno effettuate eventuali *interpolazioni*, in particolare

- permette il *command substitution*[30];
- l'esplosione di eventuali variabili[31];
- espansione dello storico qualora si utilizzi il carattere `!`[32];
- ulteriori verifiche[33].

Quindi è utile sintetizzare che il quoting può essere:

forte: per trattare letteralmente la stringa, utilizzando i singoli apici `'`;

debole: per effettuare eventuali interpolazioni, utilizzando i doppi apici `"`.

[26] carattere *backslash*
[27] *single quote*
[28] *double quote*
[29] tale differenza si riscontra anche in molti linguaggi di programmazione
[30] v. 8.4 a pag. 97
[31] v. 11.1 a pag. 123
[32] v. 13.4 a pag. 159
[33] eventuali caratteri `@` per esplosioni di array, ...

2.4 Concetti chiave

comandi: immessi dall'utente e diretti alla shell. Se quest'ultima sa interpretarli (*comandi interni*) li esegue direttamente, altrimenti lancia software (*comandi esterni*)

escaping: è un metodo per effettuare il quoting di un singolo metacarattere, quindi per trattare letteralmente il metacarattere che segue

exit: identifica una funzione o una procedura che determina il *normale* termine di un processo. Nel caso di una funzione, è accompagnato da uno stato di uscita che, se non espresso, di default è 0

interpolazione: valutazione di una stringa per eventuali esplosioni di parti della stessa

login: procedura tramite la quale l'utente è autenticato e, se l'autenticazione è avvenuta con successo, sarà aperta una *sessione di sistema*, quindi sarà proiettato all'environment assegnato per poter interagire con il SO (secondo le autorizzazioni a lui concesse)

logout: procedura opposta al *login*, tramite la quale il SO termina la *sessione di sistema* precedentemente assegnata

metacarattere: è un carattere speciale al quale è assegnato un'interpretazione non letterale (a meno che non sia stato effettuato il quoting dello stesso)

prompt: indica l'attesa da parte della shell di un comando

quoting: è il metodo affinché una sequenza contenente uno o più metacaratteri sia trattata letteralmente, quindi anche i metacaratteri saranno trattati letteralmente

shell: è la componente di un SO che permette all'utente l'interazione con lo stesso tramite l'immissione di comandi

2.5 Comandi introdotti nel capitolo

Ctrl + D ⇒ è la sequenza che trasmette il carattere di controllo `EOF`

`bash` ⇒ è la ***B**ourne-**A**gain **SH**ell*

`cd` ⇒ comando builtin, acronimo di ***c**hange **d**irectory*

`date` ⇒ scrive o modifica la data e orario di sistema

`echo` ⇒ comando builtin, mostra una linea di testo

`exit` ⇒ comando builtin. Genericamente provoca la terminazione di un processo

`help` ⇒ visualizza informazioni sui comandi integrati

`login` ⇒ apre una sessione sul sistema

`logout` ⇒ esce dalla sessione del sistema

`sh` ⇒ è la *Bourne **SH**ell*

`type` ⇒ visualizza le informazioni sul tipo di comando

`which` ⇒ localizza un comando

Capitolo 3

Documentazione

FORSE è l'aspetto a cui si presta meno attenzione, ma è veramente fondamentale avere a disposizione una ricca e coerente documentazione affinché si possa instaurare un buon rapporto con un software complesso.

3.1 Comando `help` e argomento `--help`

In `bash` i comandi interni sono descritti in dettaglio dal comando `help` mettendo quello da esaminare come argomento. Riprendendo l'***Esempio 2.4***:

```
1 $ help exit
2 exit: exit [n]
3     Esce dalla shell.
4
5     Esce dalla shell con uno stato N. Se N è omesso lo stato
6     di uscita è quello dell'ultimo comando eseguito.
```

Esempio 3.1: *Comando* `help exit`

Per i comandi interni si può equivalentemente far seguire al comando l'opzione `--help`:

```
1 $ exit --help
2 exit: exit [n]
3     Esce dalla shell.
4
5     Esce dalla shell con uno stato N. Se N è omesso lo stato
6     di uscita è quello dell'ultimo comando eseguito.
```

Esempio 3.2: *Comando* `exit --help`

Nell'***Esempio 3.3*** si è utilizzato il comando interno `type` con l'opzione `--help`.
e il risultato è identico all'***Esempio 2.13*** in cui si era utilizzato il comando `help` con argomento `type`.

Per quanto riguarda comandi esterni, la maggior parte supporta l'opzione `--help`, solitamente riportando la sintassi e i parametri supportati.

```
1 $ type --help
2 type: type [-afptP] nome [nome ...]
3     Visualizza le informazioni sul tipo di comando.
4
5     Per ogni NOME, indicare come verrebbe interpretato se utilizzato come
      a
6     nome del comando.
7
8     Opzioni:
9 ...
```

Esempio 3.3: *Opzione* `--help`

3.2 Manuale

3.2.1 Overview

Quasi tutti i programmi a linea di comando sono accompagnati da una *pagina di manuale* (*man page*). Più in generale, molte distribuzioni GNU/Linux seguono la filosofia che ogni comando a linea di comando dovrebbe avere una pagina di manuale, ed essendo estremamente precise e aggiornate, personalmente le ritengo le miglior risorsa documentale delle distribuzioni.
In ogni caso, dal passaggio precedente consegue che spesso per i neofiti[1] uno svantaggio è da evidenziare: si rischia di non trovare facilmente l'opzione interessata, o ancora peggio si rischia di perdersi.
Per richiamare la pagina di manuale del comando, è sufficiente digitare il comando `man comando`. Nell'***Esempio 3.4*** è mostrato l'uso del comando `man bash`, in cui per uscire si digiterà il comando `q`[2].
Interessante confrontare lo stesso ***Esempio 3.4*** con la ***Figura 2.1***.

3.2.2 Struttura

Indicativamente la struttura delle pagine di manuale segue quanto riportato nella ***Tabella 3.1***, frequente e molto utile la sezione `EXAMPLES`.
Ciascuna pagina di manuale appartiene ad un TOPIC come indicato in ***Tabella 3.2***.
Le pagine di manuale sono scritte in un formato per essere processato dal programma `groff` e sono memorizzate nella directory `/usr/share/man` nelle subdirectory:

`man`*n*
: in cui *n* è un numero di capitolo riportato in ***Tabella 3.2*** a seconda della tematica di appartenenza;

`lang`
: in cui *lang* rappresenta l'abbreviazione delle traduzioni disponibili, a loro volta contenenti subdirectory `man`*n* secondo lo schema descritto nel precedente punto.

Molti comandi hanno contemporaneamente pagine di manuale appartenenti a più TOPIC, per cui talvolta è importante notare il numero tra parentesi di fianco il nome del comando,

[1] ma vale anche più in generale per tutti gli utenti che interagiscono con comandi particolarmente complessi, per cui le pagine di manuale sono molto lunghe e ricche di opzioni

[2] per conoscere meglio il manuale, ovviamente si potrà consultare la pagina del manuale del manuale, quindi `man man`

```
1 $ man bash
2 BASH(1)                    General Commands Manual
      BASH(1)
3
4 NAME
5        bash - GNU Bourne-Again SHell
6
7 SYNOPSIS
8        bash [options] [command_string | file]
9
10 COPYRIGHT
11       Bash is Copyright (C) 1989-2018 by the Free Software Foundation,
      Inc.
12
13 DESCRIPTION
14       Bash  is  an  sh-compatible  command language interpreter that
      executes
15       commands read from the standard input or from a file.  Bash also
      incor-
16       porates useful features from the Korn and C shells (ksh and csh).
17
18       Bash  is  intended  to  be a conformant implementation of the Shell
       and
19       Utilities portion  of  the  IEEE  POSIX  specification  (IEEE
      Standard
20       1003.1).  Bash can be configured to be POSIX-conformant by default.
21 ...
```

Esempio 3.4: *Comando* `man bash`

SECTION	CONTENT
NAME	Command name and brief description
SYNOPSIS	Description of the command syntax
DESCRIPTION	Verbose description of the command's effects
OPTIONS	Available options
ARGUMENTS	Available Arguments
FILES	Auxiliary files
EXAMPLES	Sample command lines
SEE ALSO	Cross-references to related topics
DIAGNOSTICS	Error and warning messages
COPYRIGHT	Authors of the command
BUGS	Known limitations of the command

Tabella 3.1: *Manual page sections*

come evidenziato nell'***Esempio 3.5***[34].
Scorrendo questa pagina di manuale si arriva alla sezione `SEE ALSO` in cui si può notare quanto riportato nell'***Esempio 3.6***.
Quindi si rimanda ad un'altra pagina dello stesso comando, ma questa volta al topic *Con-*

[3] un identico output l'avremmo avuto digitando il comando `man 1 crontab`
[4] il `crontab` sarà esaminato alla sezione **??** a pag. **??**

N°	TOPIC
1	User commands
2	System calls
3	C language library functions
4	Device files and drivers
5	Configuration files and file formats
6	Games
7	Miscellaneous (e. g. groff macros, ASCII tables, . . .)
8	Administrator commands
9	Kernel functions
n	New commands

Tabella 3.2: *Manual Page Topics*

```
1 $ man crontab
2 CRONTAB(1)                  General Commands Manual
      CRONTAB(1)
3
4 NAME
5    crontab - maintain crontab files for individual users (Vixie Cron)
6
7 SYNOPSIS
8    crontab [ -u user ] file
9    crontab [ -u user ] [ -i ] { -e | -l | -r }
10
11 DESCRIPTION
12    crontab  is  the  program used to install, deinstall or list the tables
13    used to drive the cron(8) daemon in Vixie Cron.   Each  user  can  have
14    their    own    crontab,    and    though    these    are    files    in
15    /var/spool/cron/crontabs, they are not intended to be edited directly.
16 ...
```

Esempio 3.5: *Comando* `man crontab`

figuration files and file formats. Per consultarlo, si può digitare direttamente `man 8 crontab` come nell'***Esempio 3.7***.

```
1 $ man 5 crontab
2 CRONTAB(5) File Formats Manual CRONTAB(5))
3
4 NAME
5     crontab - tables for driving cron
6
7 DESCRIPTION
8     A  crontab file contains instructions to the cron(8) daemon of the gen
      -
9     eral form: ``run this command at this time on this date''.   Each
      user
10 ...
```

Esempio 3.7: *Comando* `man 5 crontab`

Ovviamente per aggiungere l'argomento numerico sarebbe opportuno sapere se il manuale del comando è sviluppato anche per quel topic e solitamente a priori è difficile saperlo o

```
1 $ man crontab
2 ...
3 SEE ALSO
4        crontab(5), cron(8)
5 ...
```

Esempio 3.6: *Comando* `man crontab`*, sezione* `SEE ALSO`

ricordarlo. È pertanto utile aggiungere l'opzione `-a` o nel formato esteso `--all` come nell'***Esempio 3.8***.

```
1 $ man -a crontab
2 ...
```

Esempio 3.8: *Comando* `man -all crontab`

in cui si consulterà prima il manuale con indice minore, poi premendo il tasto `q` per uscire dalla pagina, si otterrà

```
1 --Man-- successivo: crontab(5) [ mostra (return) | salta (Ctrl-D) | esci (
      Ctrl-C) ]
```

Esempio 3.9: *Comando* `man -a crontab`

in cui sono indicate le alternative:

Enter
: per accedere al manuale successivo (in questo caso `crontab(5)`);

Ctrl + D
: per saltare il manuale proposto per accedere al successivo (se esiste);

Ctrl + C
: per uscire dalla manualistica.

3.2.3 whatis e apropos

Due comandi utili per la ricerca di manuali sono:

`whatis`[5]
: ricerca le pagine di manuale il cui nome è riferito all'argomento utilizzato nel digitare il comando (v. ***Esempio 3.10***);

`apropos`[6]
: ricerca le pagine di manuale le cui *keywords* sono equivalenti all'argomento utilizzato nel digitare il comando (v. ***Esempio 3.11***).

```
1 $ whatis man
2 man (1)               - un'interfaccia ai manuali di riferimento in linea
3 man (7)               - macros to format man pages
4 $ whatis echo
5 echo (1)              - display a line of text
```

Esempio 3.10: *Comando* `whatis`

[5] equivalente a `man -f`
[6] equivalente a `man -k`

```
1 $ apropos echo
2 echo (1)              - display a line of text
3 l2ping (1)            - Send L2CAP echo request and receive answer
4 lessecho (1)          - expand metacharacters
5 pam_echo (8)          - PAM module for printing text messages
6 ping (8)              - send ICMP ECHO_REQUEST to network hosts
7 ping4 (8)             - send ICMP ECHO_REQUEST to network hosts
8 ping6 (8)             - send ICMP ECHO_REQUEST to network hosts
9 xmessage (1)          - display a message or query in a window (X-based /
      bin/echo)
```

Esempio 3.11: *Comando* `apropos`

3.3 info

TODO

3.4 HowTo

TODO

3.5 Concetti chiave

documentazione: è accessibile attraverso diverse modalità, sia offline sia online. La principale fonte è la documentazione interna accessibile via `man`

3.6 Comandi introdotti nel capitolo

Ctrl + C ⇒ la sequenza trasmette un segnale *SIGINT*[7] al processo che può intercettare lo stesso. Solitamente però non è gestita da quest'ultimo, per cui la sequenza è spesso utilizzata per uccidere i processi

`apropos` ⇒ ricerca nei nomi e nelle descrizioni delle pagine di manuale

`groff` ⇒ è il frontend per l'omonimo sistema di formattazione dei documenti

`man` ⇒ un'interfaccia ai manuali di riferimento in linea

`whatis` ⇒ mostra le descrizioni delle pagine di manuale

Capitolo 4

Gestione di file e directory

QUESTO capitolo è dedicato alla gestione dei *file* e delle *directory*. Senza entrare nei dettagli[1], con file si intende un contenitore di dati, con directory una collezione di file.

4.1 File e percorsi

4.1.1 File

La memorizzazione di dati persistenti in un sistema avviene su sistemi di storage in entità denominate *file*.
In un sistema GNU/Linux sostanzialmente non ci non limitazioni nei caratteri utilizzabili, ad eccezione del carattere `/`[2] e del carattere [3]. In ogni caso, tra breve sarà consigliato un utilizzo più ragionato.

Creazione

Nel seguito del capitolo e nei successivi si vedrà la creazione di un file in una moltitudine di modi, nel seguente esempio si utilizzerà il comando `touch`:

```
1 $ touch lettera.txt
```

Esempio 4.1: *Creazione file tramite* `touch`

Caratteri consigliati Una buona regola è utilizzare esclusivamente quest'elenco di simboli nei nomi dei file per non incorrere in problemi:

```
1 ABCDEFGHIJKLMNOPQRSTUVWXYZ
2 abcdefghijklmnopqrstuvwxyz
3 0123456789+-._
```

Esempio 4.2: *Caratteri consigliati*

[1] l'argomento sarà ripreso nel capitolo 12 a pag. 131
[2] *slash*, separatore delle directory come vedremo nella sezione 4.2 a pag. 33
[3] codice ASCII 0, vedi ***Tabella 24.6***

Caratteri speciali Le limitazioni imposte dal sistema sono trascurabili, ma tendenzialmente sarebbe da evitare l'uso dei caratteri speciali elencato nella sezione 2.3.3 a pag. 20, inoltre taluni caratteri possono essere visualizzati in maniera differente se lo stesso nome di file è letto da un sistema con codifica differente[4], per cui è conveniente limitarsi ai caratteri stampabili dell'*ASCII base*[5][6].

In ogni caso, come mostrato nel seguente esempio, per la creazione degli stessi si può ricorrere al già citato *escaping*[7]:

```
1 $ touch new\ lettera.txt
2 $ touch 'prima o seconda lettera.txt'
```

Esempio 4.3: *Creazione file ed escaping*

Estensioni

Nei sistemi GNU/Linux l'estensione non caratterizza il tipo di file, quindi si utilizza esclusivamente per facilitare l'identificazione del contenuto di un file. In altri termini, se un file testuale si chiamerà `testo.txt` oppure solo `testo` non ci sarà nessuna differenza. E non ci sarà nessuna differenza nemmeno tra i file `eseguimi` ed `eseguimi.bash`. Quanto scritto sarà approfondito alla sezione **??** a pag. **??**.

File nascosti

Semplicemente i file che iniziano con il carattere `.` sono considerati file nascosti. Non sono realmente nascosti[8], ma tendenzialmente sono utilizzati per file contenenti configurazioni, impostazioni di software o credenziali.

4.1.2 Esempi di filename

Di seguito un esempio riepilogativo:

```
1 files_UP          # ok
2 foo.txt.bak       # ok
3 50.something      # estensione poco significativa, ma ok
4 .local_config     # ok, da considerare file nascosto/slash
5
6 7_of_9            # ok
7 -10°C             # inizia con - e contiene il carattere °
8 1/2-h             # contiene il carattere /, non può essere un filename
```

Esempio 4.4: *Esempi di nomi di file con commento*

[4] più precisamente, la rappresentazione dei caratteri dipende dalle impostazioni locali, ad esempio un sistema con impostazione locale *ISO 8859-1* e un altro con *ISO 8859-15* (conosciuti rispettivamente con *ISO-Latin-1* e *ISO-Latin-9*) hanno rappresentazioni grafiche differenti per l'estensione ASCII (cioè da valore ASCII 128_{10})

[5] vedi ***Tabella 24.6***

[6] per affrontare questi problemi, si può ricorrere al software `convmv` affrontato nella sezione 14.8 a pag. 177

[7] sempre della sezione 2.3.3

[8] il concetto sarà più chiaro alla sezione 4.4 a pag. 36

4.2 Directory

Le directory sono essenzialmente dei contenitori e in GNU/Linux le directory sono semplicemente file, anche se per l'accesso si utilizzano metodi differenti dai file standard. Nella precedente sezione si è evidenziata la regola per l'utilizzo dei caratteri nei nome dei file, in particolare sono stati elencati nell'***Esempio 4.2*** i caratteri consigliati, tale consiglio va ovviamente esteso per i nomi delle directory.
Il carattere `/` ha lo scopo di separare i nomi dei file dai nomi delle directory e i nomi delle directory tra loro.

4.2.1 Directory tree

Le directory possono contenere altre directory, da cui ne risulta una struttura ad albero, quindi generalmente si utilizza il termine *directory tree*. La radice di tutte le directory è `/` e si chiama *root directory*.

Struttura gerarchica preinstallata Molte distribuzioni comuni contengono decine di migliaia di file raccolti in una gerarchia di directory principalmente strutturata secondo convenzioni che saranno analizzate nella sezione 12.3 a pag. 133.

Esempi Per ora alcuni esempi sulla organizzazione locale che può essere analizzata attraverso il comando `tree`. Un primo esempio di una directory vuota:

```
1 $ tree
2 .
3
4 0 directories, 0 files
```

Esempio 4.5: *Comando* `tree`

e un esempio più strutturato nella ***Figura 4.1***.
Dai piccoli esempi mostrati è possibile già visualizzare alcune interessanti peculiarità come i caratteri `.` e ∼ che saranno affrontate nelle prossime sezioni.

4.2.2 Home directory

Ogni utente ha la propria home, solitamente corrisponde alla directory `/home` seguita dalla propria username[9].
Quando l'utente effettua il *login* con successo, sarà proiettato direttamente in questa posizione.

4.2.3 Pathname assoluti e relativi

In un sistema GNU/Linux ogni file è descritto da un *path* che inizia dalla *root directory* e menziona ogni directory del percorso fino ad arrivare al file in oggetto. Il path completo così definito è denominato **path assoluto**.
Esempi di path assoluti:

[9] l'assegnazione del path agli utenti sarà esaminato nella sezione 17.4.1 a pag. 218

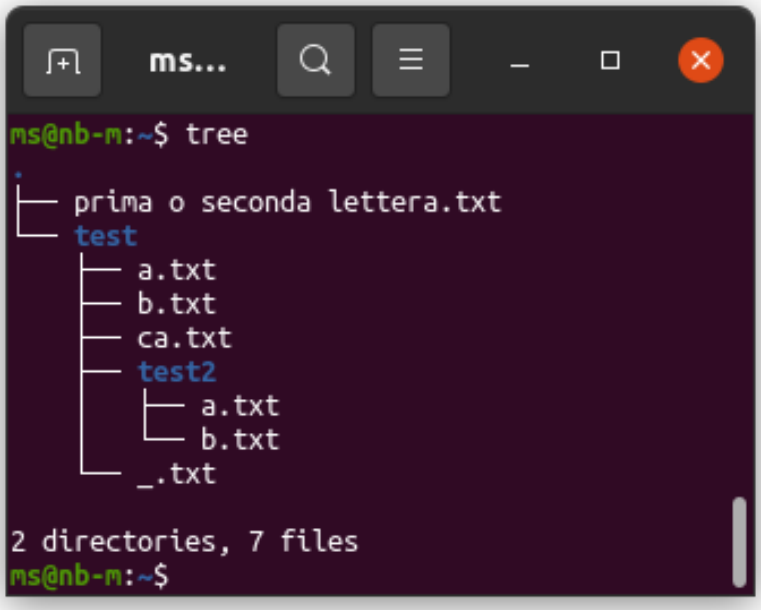


Figura 4.1: *Esempio di output del comando* `tree`

```
1 /home/ms
2 /home/ms/lettera.txt
```

Esempio 4.6: *Alcuni path name assoluti*

in cui il primo identifica il path assoluto della directory `ms`, il secondo path assoluto individua il file *lettera.txt* all'interno del path precedente.
Quando l'utente *ms* effettuerà il login e sarà proiettato nella propria **home directory**, la sua **current directory**[10] sarà `/home/ms`, per cui lui potrà accedere al file direttamente tramite il path relativo `lettera.txt`. Più in generale, tutti i path che non inizieranno con la `/` sono definiti **path relativi**. Ad esempio:

```
1 lettera.txt
2 ms/lettera.txt
```

Esempio 4.7: *Alcuni path name relativi*

il primo è il path relativo già esaminato, il secondo è il path relativo prendendo come riferimento base la directory `/home`.

4.2.4 Shortcuts

Carattere ~ Identifica la *home directory* dell'utente vista in sezione 4.2.2 a pag. 33. Nel caso dell'utente *ms* in cui la sua home è `/home/ms`, il carattere ~ è il riferimento alla stessa directory. Si veda l'***Esempio 4.8***.

[10] denominata in maniera equivalente **working directory**

```
1 $ echo ~
2 /home/ms
```

Esempio 4.8: *Shortcut ~*

Carattere . Il carattere . in una directory identifica la directory stessa, si veda l'***Esempio 4.5***. Non si utilizza molte volte questo shortcut poiché, se già raggiunta la directory di interesse, ha poco senso citare la directory stessa se non per eventuale chiarezza.
In ogni caso ci sono situazioni in cui è utile, ad esempio:

- ⇒ lo si può utilizzare in una liste di *path di ricerca* per indicare che la directory corrente è da includere[11];
- ⇒ uno script potrebbe invocare un programma (ad esempio `mytest`) che si trova nella stessa directory di lavoro attraverso il comando mostrato nell'***Esempio 4.9***, quindi senza dover immettere il pathname assoluto.

```
1 $ ./mytest
```

Esempio 4.9: *Esempio dello shortcut .*

Carattere .. Indica la **parent directory**, cioè una directory di un livello più alto, quindi di un livello più vicino a /. Ogni directory ha un livello più alto (quindi meno profondo nell'albero) ad esclusione della directory radice. Per essere più precisi, al livello radice, gli shortcut . e .. coincidono, in altri termini la directory / ha come genitrice se stessa.

4.3 Directory commands

4.3.1 Working directory

Eseguendo il comando `pwd` si chiederà al sistema ***p**rint **w**orking **d**irectory*:

```
1 $ pwd
2 /home/ms
```

Esempio 4.10: *Esempio del comando `pwd`*

In maniera indistinta si parla di **current directory**.

4.3.2 Change directory

Per il cambio di directory di lavoro si utilizza il comando `cd` ***c**hange **d**irectory*:

```
1 $ cd
2 $ pwd
3 /home/ms
4 $ cd dev
5 $ cd ..
```

Esempio 4.11: *Esempio del comando `cd`*

[11] si affronterà questo argomento alla sezione 11.1 pagina 123 alla variabile d'ambiente `PATH`

e con prompt più comune:

```
1 ms@nb:~$ cd dev
2 ms@nb:~/dev$ pwd
3 /home/ms/dev
4 ms@nb:~/dev$ cd courses
5 ms@nb:~/dev/courses$ cd ..
6 ms@nb:~/dev$ cd
7 ms@nb:~$
```

Esempio 4.12: *Esempi dei comandi* `cd` *e* `pwd` *con prompt più comune*

Dagli esempi si può notare che il comando `cd DIR` cambia la directory in `DIR`, con il valore `DIR` predefinito della *directory home* dell'utente[12]. In altri termini, se `DIR` non sarà passato, allora il comando utilizzerà il valore predefinito.

Tornare indietro Per tornare nella directory da cui si proveniva è possibile utilizzare l'argomento `-`, quindi `cd -` riporta alla directory di provenienza. Ovviamente se reiterato ci si alternerà tra due directory.
Appena si è effettuato il login, non c'è nessuna directory precedente, si veda l'esempio seguente:

```
1 $ cd -
2 -bash: cd: OLDPWD non impostata
```

Esempio 4.13: *Esempio del comando* `cd -`

quindi si nota che la variabile `OLDPWD` non è stata ancora settata, per cui non è possibile tornare al percorso indicato, percorso che sarà aggiornato ogni volta con la *current directory* appena prima di spostarsi sul nuovo path indicato dall'argomento del comando `cd`[13].

4.4 List dei file e directory

Il comando `ls` elenca i file e le directory presenti nella directory indicata come argomento, con argomento predefinito nella *directory corrente*. Senza opzioni, l'elenco è mostrato ordinato per filename in forma multicolonna.
Spesso il comportamento default è mostrare l'elenco di file in colori differenti secondo quanto mostrato nella ***Tabella 4.1***[14]. Ma come è stato possibile?[15]
Per il momento è sufficiente sapere che nel proprio environment sono solitamente impostati diversi *alias*[16] di default, ad esempio, per quando riguarda il comando `ls`, sul sistema che sto utilizzando ha un environment con questi alias a disposizione:

`ls` → `ls --color=auto`

`ll` → `ls -alF`

[12] più propriamente il valore della variabile è settato al valore della variabile di shell HOME. Per approfondire si veda la sezione 11.1 pagina 123

[13] per approfondire si veda la sezione 11.1 pagina 123

[14] la tipologia dei file sarà affrontato alla sezione 12.2 pagina 131

[15] se quanto appena scritto al lettore non accade, la domanda va traformata in: *Come è possibile che agli altri accada e non a me?*

[16] affronteremo l'argomento alla sezione 13.3 pagina 157

`la` → `ls -A`

`l` → `ls -CF`

Il primo spiega perché digitando semplicemente `ls` derivi l'output colorato. Nonostante quest'impostazione, qualora si desiderasse il comportamento del comando `ls` di default del software (cioè ignorando eventuali *alias*), è sufficiente far precedere il comando dal carattere `\`.

TIPO DI FILE	COLORE	SUFFISSO → `ls -F`	LETTERA ASSOCIATA → `ls -l`
regular file	nero	nessuno	-
eseguibile	verde	*	-
directory	blu	/	d
link	ciano	@	l

Tabella 4.1: *Comando `ls`*

OPZIONE	EFFETTO
`-a` o `--all`	mostra anche i file nascosti
`-C`	mostra la lista in colonne
`-d` o `--directory`	mostra le directory, non il contenuto
`-i` o `--inode`	mostra l'*inode number*[17]
`-l` o `--format=long`	mostra informazioni extra
`-o` o `--no-color`	evita che l'output sia colorato
`-p` o `-F`	aggiunge un'indicatore per tipologia di file
`-r` o `--reverse`	mostra l'output invertendo l'ordine
`-R` o `--recursive`	esegue ricorsivamente il comando nelle sottodirectory
`-S` o `--sort=size`	ordina secondo la *size* (prima i maggiori)
`-t` o `--sort=time`	ordina secondo la data di modifica (prima i più recenti)
`-X` o `--sort=extension`	ordina secondo l'estensione

Tabella 4.2: *Opzioni del comando `ls`*

Di seguito un esempio riepilogativo.

Fretta

Erroneamente capita spesso di digitare il comando `sl` al posto di `ls`, e solitamente per un comando errato si ha un comportamento come nell'***Esempio 4.15***.

```
1 $ sl
2 sl: comando non trovato
```

Esempio 4.15: *Esempio di digitazione errata del comando `ls`*

Ma digitandolo su questo sistema `sl` appare quanto mostrato nella ***Figura 4.2***.
In questa distribuzione[18] ho trovato preinstallato[19] il pacchetto `sl` che installa il bina-

[17] l'inode sarà affrontato nella sezione **??** a pag. **??**
[17] l'argomento sarà affrontato in 12.2 a pag. 131
[18] anche in altre versioni distribuzioni, quindi immagino discretamente comune
[19] l'argomento della pacchettizzazione sarà affrontato nella sezione 19.2.3 a pag. 265

```
1 $ ls
2 test
3 $ ls -l
4 drwxrwxr-x 3 ms ms 4096 ott 11 18:28 test
5 $ ls -al test
6 totale 12
7 drwxrwxr-x 3 ms ms 4096 ott 11 18:28 .
8 drwxr-xr-x 3 ms ms 4096 ott 14 19:21 ..
9 -rw-rw-r-- 1 ms ms    0 ott 11 18:27 a.txt
10 drwxrwxr-x 2 ms ms 4096 ott 11 18:28 test2
11 $ ls -l test/test2
12 totale 0
13 -rw-rw-r-- 1 ms ms 0 ott 11 18:28 a.txt
14 -rw-rw-r-- 1 ms ms 0 ott 11 18:28 b.txt
15 $ ls -ld test/test2
16 drwxrwxr-x 2 ms ms 4096 ott 11 18:28 test/test2/
```

Esempio 4.14: *Esempi con il comando* `ls`

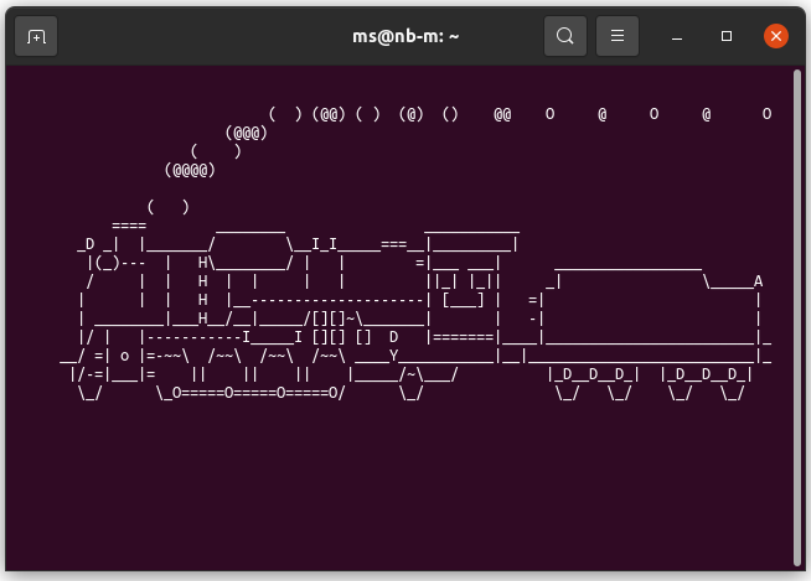


Figura 4.2: *Output del software* **S**team **L**ocomotive *richiamato attraverso il comando* `sl`

rio `/usr/games/sl`[20]. Consultanto il `man` relativo, si scopre che il software ***Steam Locomotive*** ha anche ulteriori opzioni di animazione.

4.5 Creazione e cancellazione di directory

Per creare una directory il comando da digitare è `mkdir`:

[20] come ricercare il binario si è visto nell'***Esempio 2.9*** e nell'***Esempio 2.10*** a pag. 20

```
1 $ mkdir test1
2 $ mkdir test1
3 mkdir: impossibile creare la directory "test1": File già esistente
4 $ mkdir -p test1/test1a
5 $ mkdir -p test1/test1b
6 $ mkdir -p test2/test2a
7 $ mkdir -p test2/test2b
8 $ ls test1 test2
9 test1:
10 test1a  test1b
11
12 test2:
13 test2a  test2b
```

Esempio 4.18: *Esempio con il comando* `mkdir -p`

```
1 RMDIR(1)                          User Commands
2
3 NAME
4     rmdir - remove empty directories
5 ...
6     -p, --parents
7           remove  DIRECTORY  and  its ancestors; e.g., 'rmdir -p a/b/c' is
8           similar to 'rmdir a/b/c a/b a'
9 ...
```

Esempio 4.19: *Segmento del manuale di* `rmdir`

```
1 $ mkdir test
```

Esempio 4.16: *Esempio con il comando* `mkdir`

in cui si è chiesto di creare la directory `test` che non dovrà già esistere pena errore.
Tra le opzioni disponibile, sicuramente la più utile è `-p`:

```
1 MKDIR(1)                          User Commands
2
3 NAME
4     mkdir - make directories
5 ...
6     -p, --parents
7           no error if existing, make parent directories as needed
8 ...
```

Esempio 4.17: *Segmento del manuale di* `mkdir`

utilizzabile come nell'***Esempio 4.18***.
Simmetricamente c'è il comando `rmdir` per la rimozione della directory, e consultando il `man` anch'esso con una opzione `-p` come nell'***Esempio 4.19*** e utilizzabile come nell'***Esempio 4.20***.

```
$ ls test1 test2
test1:
test1a  test1b

test2:
test2a  test2b
$ rmdir test1
rmdir: rimozione di 'test1' non riuscita: Directory non vuota
$ rmdir -p test1/test1a
rmdir: rimozione della directory 'test1' non riuscita: Directory non vuota
$ rmdir -p test1/
rmdir: rimozione di 'test1/' non riuscita: Directory non vuota
$ rmdir -p test1/test1b
$ rmdir test2/test2a
$ rmdir -p test2/test2b
```

Esempio 4.20: *Esempio con il comando* `rmdir -p`

```
CP(1)                          User Commands

NAME
    cp - copy files and directories

SYNOPSIS

    cp [OPTION]...  [-T] SOURCE DEST
    cp [OPTION]...  SOURCE...  DIRECTORY
    cp [OPTION]...  -t DIRECTORY SOURCE...

...
```

Esempio 4.21: *Segmento del manuale di* `cp`

4.6 Gestione dei file

4.6.1 Copiare, spostare e cancellare

Copiare È possibile copiare file utilizzando il comando `cp`. Consultando il `man` 4.21 e tralasciando la terza modalità indicata[21], le prime si possono rispettivamente sintetizzare nel seguente modo:

`cp [OPTION]... SOURCE DEST]`
: copia il file sorgente nella destinazione, quindi è una copia *1 a 1* con file di destinazione indicato;

`cp [OPTION]... SOURCE... DIRECTORY`
: copia il (o i file) sorgenti nella directory di destinazione indicata, e in quest'ultimo caso il nome del o dei file indicati come sorgente rimarranno gli stessi.

Il comando prevede diverse opzioni interessanti, nella ***Tabella 4.3*** sono illustrate le principali.

[21] è mostrato che attraverso l'opzione `-t` (o per esteso `--target-directory=DIRECTORY`) si possono invertire gli argomenti, quindi in questo specifico caso si farà precedere la directory di destinazione e poi il o i file sorgenti

OPZIONE	SIGNIFICATO	DESCRIZIONE
`-f`	*force*	procede alla sovrascrittura senza chiedere conferma
`-i`	*interactive*	chiede per ciascun file di origine se la destinazione (qualora già esista) deve essere sovrascritta
`-p`	*preserve*	prova a preservare tutti gli attributi[22]
`-R`	*recursive*	copia le directory con tutto il loro contenuto
`-u`	*update*	copia il file sorgente solo se è più recente di quello di destinazione (o se quest'ultimo non esiste)
`-v`	*verbose*	mostra in output tutta l'attività svolta

Tabella 4.3: *Principali opzioni del comando* `cp`

```
1  $ mkdir -p test/test1
2  $ touch test_a.txt test_b.txt
3  $ cp test_a.txt test_b.txt test
4  $ cp test_a.txt test
5  $ cp test_a.txt test/test1/test.txt
6  $ cp -i test_b.txt test/test1/test.txt
7  cp: sovrascrivere 'test/test1/test.txt'? n
8  $ ls -R
9  .:
10 test  test_a.txt  test_b.txt
11
12 ./test:
13 test1  test_a.txt  test_b.txt
14
15 ./test/test1:
16 test.txt
```

Esempio 4.22: *Esempio con il comando* `cp`

È utile sapere che di default il comando `cp` non chiede conferma di *overwrite*, per cui se la destinazione contiene già il file con lo stesso nome, esso sarà sovrascritto. Per evitare questo comportamento, esiste l'opzione `-i`[23].
L'***Esempio 4.22*** per illustrare quanto sopra.

Spostare A differenza del comando `cp` che esegue una copia del file tramite duplicazione, il comando `mv` sposta il file su una differente posizione eventualmente rinominandolo[24].
Consultando il `man` 4.23 e confrontandolo con il `man` 4.21 del comando `cp`, si nota che la sintassi è equivalente, e anche in questo caso se si fornirà una lista di più file sorgenti, la destinazione dovrà essere una directory (e ovviamente non si potrà procedere con il *rename*). La differenza è nel caso in cui il file sorgente è una directory, poiché oltre a spostarla si potrà rinominarla.
Continuando dal contenuto prodotto dall'***Esempio 4.22***, si procede con l'***Esempio 4.24***.

[23] `--interactive`, quindi chiederà conferma prima di procedere all'eventuale sovrascrittura
[24] l'operazione di spostamento sarà approfondita nella sezione **??** a pag. **??**

```
MV(1)                          User Commands

NAME
    mv - move (rename) files

SYNOPSIS

    mv [OPTION]...  [-T] SOURCE DEST
    mv [OPTION]...  SOURCE...  DIRECTORY
    mv [OPTION]...  -t DIRECTORY SOURCE...

...
```

Esempio 4.23: *Segmento del manuale di* `mv`

```
$ mv test_a.txt test_c.txt
$ mv -i test/test_a.txt test/test_b.txt .
mv: sovrascrivere './test_b.txt'? y
$ mv test_c.txt test_a.txt
$ mv test test_bak
$ ls -R
.:
test_a.txt  test_bak  test_b.txt

./test_bak:
test1

./test_bak/test1:
test.txt
```

Esempio 4.24: *Esempio con il comando* `mv`

```
1 $ touch a.txt
2 $ ln a.txt b.txt
3 $ ls -l
4 -rw-rw-r-- 2 ms ms    0 ott 29 16:54 a.txt
5 -rw-rw-r-- 2 ms ms    0 ott 29 16:54 b.txt
```

Esempio 4.26: *Esempio con il comando* `ln` *(*hard link*)*

Cancellare Il comando per rimuovere i files è `rm`. Senza entrare nei dettagli sui permessi[25], nella propria *home directory* si potrà cancellare tutto, anche i file precedenti il primo *login*.
Le opzioni più interessanti sono riportate nella ***Tabella 4.4***.

OPZIONE	SIGNIFICATO	DESCRIZIONE
`-f`	*force*	procede senza chiedere conferma
`-i`	*interactive*	chiede per ciascun file prima di procedere
`-p`	*preserve*	prova a preservare tutti gli attributi[26]di ciascun file sorgente
`-r` o `-R`	*recursive*	cancella le directory e i loro contenuti ricorsivamente
`-v`	*verbose*	mostra in output tutta l'attività svolta

Tabella 4.4: *Principali opzioni del comando* `rm`

Come i precedenti comandi `cp` e `mv`, di default la rimozione avviene senza conferma, quindi anche in questo caso è utile l'opzione `-i` che chiederà conferma esplicita prima di eventuali danni.
Continuando dal contenuto prodotto dall'***Esempio 4.24***, si procede con l'***Esempio 4.25***.

```
1 $ rm -i test_a.txt
2 rm: rimuovere file regolare vuoto 'test_a.txt'? y
3 $ rm test_bak
4 rm: impossibile rimuovere 'test\_bak/': \'E una directory
5 $ rm -r test_bak
6 $ ls -R
7 .:
8 test_b.txt
```

Esempio 4.25: *Esempio con il comando* `rm`

4.6.2 Link

I *link* sono dei riferimenti ai file. L'obiettivo della creazione è poter accedere ad uno stesso file da più punti di accesso.

Hard link Come nell'esempio seguente, il comando per la creazione di un *link* è `ln` in cui il primo argomento è il file esistente, il secondo sarà il nome del link.
Esaminando l'output del comando `ls -l`, ignorando al momento la prima colonna[27], si nota per entrambi i filename il numero `2` che identifica il numero di riferimenti allo stesso

[25] affrontati nella sezione 18 a pag, 227
[27] sarà analizzata nella sezione 18 a pag. 227

file.
Questo è il motivo per il quale il comando così utilizzato crea un link definito *hard link*, in cui dal momento della creazione è indistinguibile dal file originale. In altri termini si potrà cancellare il filename originale, ma il contenuto sarà accessibile dall'*hard link* creato. Si potranno creare ulteriori *hard link* allo stesso file, e il contenuto del file sarà accessibile finché rimarrà almeno un riferimento.

Soft link I *link simbolici* sono anch'essi dei link, ma sono definiti *soft link*. Continuando l'***Esempio 4.26***, al comando visto per la creazione andrà aggiunta l'opzione `-s` come nel seguente esempio:

```
1 $ ln -s a.txt c.txt
2 $ ls -F
3 a.txt  b.txt  c.txt@
4 $ ls -l
5 -rw-rw-r-- 2 ms ms    0 ott 29 16:54 a.txt
6 -rw-rw-r-- 2 ms ms    0 ott 29 16:54 b.txt
7 lrwxrwxrwx 1 ms ms    5 ott 29 16:50 c.txt -> a.txt
```

Esempio 4.27: *Esempio con il comando* `ln -s` *(*soft link*)*

in cui

`ls -F`
: elenca i file indicando la tipologia, e si nota che:

  - non c'è segnata nessuna indicazione relativamente al file `b.txt` perché è indistinguibile dall'originale;
  - c'è il simbolo `@` relativamente al link simbolico proprio perché è semplicemente un riferimento a filename;

`ls -l`
: indica esplicitamente che il file `c.txt` è un riferimento al filename `a.txt`. Si può notare dalla seconda colonna numerica l'indicazione dello spazio occupato dai file, sia `a.txt` e `b.txt` mostrano l'occupazione di *0 byte* perché il contenuto è nullo[28], invece `c.txt` occupa *5 byte* poiché l'occupazione dei caratteri del nome usato come *filename target* occupano esattamente quel numero[29].

4.7 Espansione

L'espansione è effettuata direttamente dalla shell e non credo sia utile in contesti di ricerca, ma potrebbe esserlo per la creazione di file o directory.

4.7.1 Parentesi graffe

L'espansione tramite parentesi graffe è indicato da una lista in cui ogni elemento sarà separato dal carattere `,`. Ad esempio:

[28] quindi il numero è esattamente l'indicazione del contenuto in byte del medesimo file

[29] quindi questo numero indica l'occupazione del nome del link e non del file, per cui non crescerebbe al crescere del contenuto del file a cui si riferisce

```
1 $ echo {rosso,verde,blu}.txt
2 rosso.txt verde.txt blu.txt
```

Esempio 4.28: *Esempio di lista/1*

Ovviamente è possibile una successione di liste tra parentesi come si seguito:

```
1 $ echo {rosso,verde,blu}{1,2,3}.txt
2 rosso1.txt rosso2.txt rosso3.txt verde1.txt verde2.txt verde3.txt blu1.txt
3 blu2.txt blu3.txt
```

Esempio 4.29: *Esempio di lista (2)*

Di seguito un esempio di creazione di 1 directory e 12 subdirectory:

```
1 $ mkdir -p classe/{1,2,3}{A,B,C,D}
```

Esempio 4.30: *Esempio di creazione di directory tramite*

4.8 Search pattern

Spesso si ha la necessità di eseguire comandi diretti contemporaneamente a più file, per tale attività si utilizza il *globbing*[30][31].
In particolare la shell permette di esprimersi tramite *wildcard* (detti anche *metacaratteri*) i quali saranno espansi secondo il contesto in cui si trovano.
Per evitare problemi, l'espansione sarà effettuata solo nella directory corrente (o comunque quella indicata) e escluderà i file nascosti (cioè che iniziano con il carattere `.`), a meno che non siano esplicitamente inclusi nel match[32].

4.8.1 Simple pattern

I più semplici wildcard sono:

`*`
: matcherà[33] qualsiasi stringa, anche vuota;

`?`
: matcherà qualsiasi carattere singolo.

Ovviamente entrambi i metacaratteri non potranno matchare con il carattere `/`.
Nell'esempio:

```
1 *
2 t*.txt
3 test1/t*.bash
4 ?a
5 ??
6 *a1?b*
7 .*
```

[30] si veda `man 7 glob`
[31] in alternativa si utilizzano le *espressioni regolari*, capitolo analizzato in 7 a pag. 79
[32] si veda l'***Esempio 4.31***
[33] l'inglese *to match* è difficile tradurlo, qualcuno preferisce *fare il match*, io preferisco la forma utilizzata

Esempio 4.31: *Simple pattern*

rispettivamente la linea indicherà che il nome del file:

1. potrà essere qualunque;
2. dovrà iniziare con `t` e terminare con `.txt`;
3. dovrà essere contenuto della directory `test1`, iniziare con `t` e terminare con `.bash`;
4. dovrà iniziare con un qualsiasi carattere, il secondo dovrà essere il carattere `a`;
5. dovrà essere composto da due caratteri;
6. conterrà una sottostringa `a1`, poi un carattere qualunque, poi il carattere `b`;
7. sarà nascosto.

Se ci saranno match, la lista restituita sarà ordinata per nome.
Il metodo probabilmente più conveniente per verificare l'espansione dei metacaratteri è utilizzare il comando `echo` come nell'esempio:

```
1 $ ls -a
2 .  ..  a.txt  A.txt b.txt  ca.txt  test1  test2  .test.conf  _.txt
3 $ echo *
4 a.txt A.txt b.txt ca.txt test1 test2 _.txt
5 $ echo .*
6 .  ..  .test.conf
7 $ echo ??.txt
8 ca.txt
```

Esempio 4.32: *Simple pattern con* `echo`

Il pattern non matcha Cosa succede se non ci sono file che possano matchare? Semplicemente il globbing non avrà successo, per cui non sarà effettuata nessun espansione, quindi il pattern impostato sarà riportato nella console letteralmente. Ripartendo dall'***Esempio 4.32***, si mostra un pattern non espanso nell'***Esempio 4.33***.

```
1 $ echo bac?.txt
2 bac?.txt
3 $ ls prova*
4 ls: impossibile accedere a 'prova*': File o directory non esistente
```

Esempio 4.33: *Esempio di pattern che non matcha*

4.8.2 Classi di carattere

Un matching maggiormente specifico si ha utilizzando le *classi di carattere* nell'espressione `[...]` in cui tra le parentesi andrà inserita la classe di caratteri tra i quali dovrà essere effettuato il match. Sempre dall'***Esempio 4.32***, si illustra l'utilizzo delle classi nell'***Esempio 4.34***.

```
1 echo [abcd].txt
2 a.txt b.txt
```

Esempio 4.34: *Esempio di classe di carattere*

Le classi di carattere utilizzabili possono essere espresse anche tramite intervalli come di seguito:

```
1 echo[2-9].bak
2 [A-Z]bracadabra.txt
```

Esempio 4.35: *Esempio di classi di carattere con intervalli*

in cui la classe espressa tra parentesi quadre nella riga:

1. eseguirebbe un match di cifra decimale da 2 a 9;
2. nella seconda di una lettera maiuscola qualunque.

Nell'***Esempio 4.34*** la classe utilizzata può quindi essere riscritta nell'***Esempio 4.36*** che avrà analogo output.

```
1 echo [a-d].txt
2 a.txt b.txt
```

Esempio 4.36: *Esempio di classe di carattere con intervallo*

Un esempio interessante è il seguente:

```
1 $ ls [A-z].txt
2 a.txt
3 A.txt
4 b.txt
5 _.txt
```

Esempio 4.37: *Esempio di classi di carattere con intervallo ampio*

in cui l'ultimo match potrebbe essere inatteso.
Nella ***Tabella 24.6*** mostrata nella parte V a pag. 399[34] si nota che tra i caratteri `A` e `z` non ci sono solo le lettere maiuscole e minuscole, ma anche altri caratteri tra cui _[35].
Affinché non matchino i caratteri non alfabetici del precedente intervallo, l'***Esempio 4.37*** può essere scritto come di seguito:

```
1 $ ls [A-Za-z].txt
2 a.txt
3 A.txt
4 b.txt
```

Esempio 4.38: *Esempio di classi di carattere con intervallo ampio (2)*

Ovviamente è possibile utilizzare concatenazioni di pattern anche eterogenei come riportato di seguito:

```
1 $ ls [a-z][a-z].txt
2 ca.txt
3 $ ls [A-z]?.txt
4 ca.txt
```

Esempio 4.39: *Esempio di concatenazione di classi di concatenazione di pattern*

[34] oppure digitando `man ascii`
[35] *underscore*

Negazione di una classe È anche possibile negare una classe di caratteri, interpretato come *qualsiasi carattere ad esclusione di quelli indicati*, utilizzando l'espressione `[!...]` in cui al posto dei puntini andrà inserita la classe di caratteri da negare. Sempre dall'***Esempio 4.32***, si illustra l'utilizzo delle negazione nel seguente esempio:

```
1 $ echo [!a-z].txt
2 A.txt _.txt
```

Esempio 4.40: *Esempio di negazione di classi di carattere*

in cui semplicemente si è affermato che il primo carattere non deve essere una lettera.

4.8.3 Classi di carattere e internazionalizzazione

Le classi di carattere esaminate precedentemente sono in effetti intervalli ASCII, quindi il range `[ -%]` corrisponde all'insieme `[ !"#$%]`, `[a-z]` corrisponde a *qualsiasi lettera minuscola*. Questo comporta che è necessario conoscere il range corretto, ma in ogni caso si ha un problema con i caratteri di alfabeti che non abbiano i caratteri latini.
Lo standard POSIX[36] ha pertanto creato delle classi di carattere nominali, quali ad esempio `[[:lower:]]` da utilizzare al posto di `[a-z]`. Inoltre ha creato delle classi di equivalenza quali `[=a=]` equivalente all'insieme `[aáàäâ]` a sua volta equivalente a `[a[.a-acute.][.a-grave.][.a-umlaut.][.a-circumflex.]]`, con la lettera `a` da sostituire a proprio piacimento. Di seguito le *classi POSIX* che definiscono le categorie di caratteri descritti.

POSIX CLASS	SIMILE A	DESCRIZIONE
`[:upper:]`	`[A-Z]`	carattere maiuscolo
`[:lower:]`	`[a-z]`	carattere minuscolo
`[:alpha:]`	`[A-Za-z]`	carattere maiuscolo o minuscolo
`[:digit:]`	`[0-9]`	cifra
`[:xdigit:]`	`[0-9A-Fa-f]`	cifra esadecimale
`[:alnum:]`	`[A-Za-z0-9]`	lettera maiuscolo, minuscola e cifra
`[:punct:]`	`[^A-Za-z0-9]`	segno di interpunzione (tutti i caratteri grafici eccetto lettere o cifre)
`[:blank:]`	`[ ]`	spazio o carattere *TAB*
`[:space:]`	`[ \t\n\r\f\v]`	caratteri *TAB*, *newline*, *return*, *line feed* e *vertical space*
`[:cntrl:]`		caratteri di controllo
`[:graph:]`	`[^[:cntrl:]]`	caratteri grafici (tutti i caratteri che hanno una rappresentazione grafica)
`[:print:]`	`[[:graph:] ]`	caratteri grafici e lo spazio
`[:word:]`	`[[:alnum:]_]`	caratteri alfanumerici e l'underscore (è una classe specifica del Perl)

Tabella 4.5: *Classi di carattere nominali* POSIX

[36] acronimo di ***P****ortable* ***O****perating* ***S****ystem****I****nterface for Uni****x***, indica la famiglia degli standard definiti dall'*IEEE* per definire concetti base da seguire durante la realizzazione del SO

4.9 Ricerca di file

Un sistema GNU/Linux è costituito da molte directory e files e il sistema fornisce diversi tool per la ricerca di risorse.

4.9.1 Ricerca rapida di file

Di seguito alcuni tool per la ricerca rapida.

`locate`†, `slocate`† e `mlocate`†

Tool storici in cui nel secondo la `s` suggerisce una più sicura alternativa al primo, ma in ogni caso sono in disuso e le distribuzioni hanno adottato il package `mlocate` che a sua volta è per molte distribuzioni un *transitional package*[37] a favore di `plocate`, anch'esso retrocompatibile con lo storico `locate`.

`plocate`

Le funzionalità di ricerca sono preservate rispetto le alternative più datate[38], ma gli algoritmi utilizzati sono oggettivamente migliori poiché il database di indicizzazione è minimizzato nelle dimensioni[39], ma soprattutto si ottengono più rapide procedure di ricerca[40].
Il tool si invoca tramite il comando `plocate` e, per retrocompatibilità, anche attraverso il comando `locate` come ad esempio:

```
1 $ locate utf8
2 /usr/lib/locale/C.utf8
3 /usr/lib/locale/C.utf8/LC_ADDRESS
4 /usr/lib/locale/C.utf8/LC_COLLATE
5 ...
```

Esempio 4.41: *Esempio di utilizzo di* `locate`

in cui è stata cercato il termine `utf8` tra i file indicizzati.
Di default la ricerca si comporta come se l'argomento fosse un pattern con il wildcard `*` sia all'inizio sia alla fine[41], cosicché l'output mostrerà tutte le ricorrenze in cui sarà presente la sottostringa di interesse. Ma tale comportamente può essere modificato utilizzando le opzioni[42] per sfruttando la potenza delle espressioni regolari[43], oppure richiedere che la ricerca non sia anche nel path ma isolata al nome, oppure che per la ricerca sia eventualmente attivato il *case insensitive*, ...
È importante tener presente che, poiché il tool utilizza un DB, è possibile non trovare un file molto recente in quanto, per tenerne traccia, lo stesso DB dev'essere aggiornato.
Per tale procedura si utilizza il software `updatedb.locate` che dovrà essere eseguito periodicamente, di default ciò avviene giornalmente[44].

[37] è un pacchetto mantenuto per retrocompatibilità a favore di uno nuovo. Il concetto di *pacchetto* sarà argomentato in 19.2.1 a pag. 257
[38] v. sezione precedente
[39] per `plocate` è localizzato di default in `/var/lib/plocate/plocate.db`
[40] per approfondire v. [Ste]
[41] quindi al posto di `WORD` sarà cercato `*WORD*`
[42] ovviamente si invita a consultare il manuale tramite `locate --help`
[43] argomento trattato nel capitolo 7 a pag. 79
[44] la schedulazione periodica di task sarà argomentata in **??** a pag. **??**

4.9.2 Ricerca completa dei file[45]

Per la ricerca di file, lo strumento più completo da utilizzare è `find`.
Questo comando ricerca i file nell'albero del percorso indicato attraversando le varie directory *ricorsivamente*[46], mostrando in output i file che matcheranno secondo il criterio indicato. Se il comando è lanciato senza parametri eseguirà una ricerca ricorsiva dalla directory corrente e matcheranno tutti i file, per cui si avrà una lista di tutti i file a partire dalla directory corrente:

```
1 $ find
2 .
3 ./b.txt
4 ./test
5 ./a
6 ...
```

Esempio 4.42: *Esempio di* `find` *senza parametri (*path relativo *di default* `.`*)*

in cui per brevità si è sintetizzato l'output. Un output simile l'avremmo anche dal comando `ls -RfA -w 1`[47], ma a breve si apprezzerà la flessibilità di questo potente strumento.

Consultando il manuale `man find`:

```
1 FIND(1)                    General Commands Manual
2
3 NAME
4     find - search for files in a directory hierarchy
5
6 SYNOPSIS
7     find  [-H]  [-L] [-P] [-D debugopts] [-Olevel] [starting-point...]
8     [expression]
9 ...
```

Esempio 4.43: *Segmento del manuale di* `find`

non trattando le opzioni precedenti lo `starting-point`, in particolare:

`-H`, `-L` e `-P`
: riguardano la gestione dei link simbolici;

`-D`
: per aggiungere nell'output del debug;

`-O`
: abilita opzioni di ottimizzazione;

la linea di comando si semplifica in:

[45] per completare questa sezione saranno necessarie competenze apprese solo successivamente (in particolare alcuni argomenti saranno presentati al capitolo 12 a pag. 131 e al capitolo **??** a pag. **??**), ma si è preferito non segmentare questa sezione e presentarla in maniera più organica. Pertanto sarà utile tornare successivamente

[46] per *ricorsivamente* si intende che attraverserà anche le subdirectory, a loro volta le relative subdirectory contenute, e così di seguito ...

[47] in realtà l'output si avvicinerebbe ancora di più tramite la sequenza `ls -RfA -w 1 | grep -v '^$'` che corrisponde allo stesso comando con un *postprocessamento*, argomento esaminato nei prossimi capitoli che termineranno in 8.2 a pag. 92

```
find [starting-point...]  [expression]
```

che sarà di seguito trattata.

Starting point

È una lista di path[48] dai quali `find` inizierà la ricerca[49], se non è fornito assumerà `.`.
Sicuramente la directory passata come argomento andrà scelta con accortezza, ad esempio:

- più vicino alla *root directory*, maggiore sarà il tempo di attraversamento;
- porre attenzione ai permessi[50].

Per evitare entrambe le questioni appena poste, nei prossimi esempi mi limiterò alla *home directory*. Un ultimo aspetto è se fornire la directory tramite *path assoluto* o *relativo*. Semplicemente dipende dall'output che si desidera, si confronti il precedente ***Esempio 4.42*** con ***Esempio 4.44***, entrambi con path l'*home* dell'utente `ms`, rispettivamente il primo con path relativo, il secondo con path assoluto.

```
1 $ find ~
2 /home/ms
3 /home/ms/b.txt
4 /home/ms/test
5 /home/ms/a
6 ...
```

Esempio 4.44: *Esempio di `find` dalla* home *(*path assoluto *`/home/ms`)*

Espressione

La potenza deriva proprio dalla flessibilità dell'espressione che può essere immessa. L'espressione può essere composta da diversi elementi, i principali si possono suddividere in:

test: ritornano *true* o *false* solitamente in base alle proprietà del file considerato. Ad esempio, un test potrebbe essere `-empty` che è *true* solo quando il file esaminato è vuoto.
In ***Tabella 4.6*** sono elencati i principali test sui file.

operatori: uniscono assieme gli elementi nel formare l'espressione.
In ***Tabella 4.7*** sono elencati i principali operatori. Se tra i test non si pone nessun operatore, è implicito che siano in *and*.

azioni: sono le azioni da effettuare sul file. Se non specificata l'azione di default sarà `-print` che scriverà il nome del file corrente sullo *standard output*[51].
In ***Tabella 4.8*** sono elencate le principali azioni disponibili.

opzioni globali: influiscono sul funzionamento di test e azioni specificate in qualsiasi parte della riga di comando. Ad esempio, l'opzione `-depth` rende l'attraversamento del filesystem in ordine *depth-first*.
In ***Tabella 4.9*** sono elencate le principali opzioni globali disponibili.

[48] solitamente è composta solo da una directory
[49] con maggior precisione è la lista delle radici di ricerca dell'albero delle directory
[50] l'argomento sarà affrontato al capitolo 18 a pag. 227
[51] l'argomento sarà affrontato in 8.1 a pag. 91

opzioni posizionali: influiscono sul funzionamento di test o azioni che siano nell'espressione in posizione successiva. Una delle poche utilizzate è `-daystart` la quale misura il tempo (per `-amin`, `-atime`, `-cmin`, `-ctime`, `-mmin` e `-mtime`) dall'inizio della giornata odierna piuttosto che da 24 ore prima.

TEST	DESCRIZIONE
`-name`	specifica un pattern per il filename. Nella variante `-iname` è *case insensitive*
`-type t`	tramite *t* si specifica il tipo di file[52]: `b` per *block device* `c` per *character device* `d` per *directory* `f` per *regular file* `l` per *symbolic link* `p` per *FIFO (named pipe)* `s` per *Unix domain socket*
`-user`	specifica l'utente al quale in file dovrà appartenere (può essere utilizzato lo UID[53]al posto del nome utente)
`-group`	specifica il gruppo al quale il file dovrà appartenere (può essere utilizzato lo GID[54]al posto del nome gruppo)
`-size n[U]`	`n` specifica la grandezza del file. Se `U` non è espresso, si intende `b`. I suffissi[55]sono: `b` per *blocchi* da 512 bytes `c` per bytes `w` per *word* da 2 bytes `k` per *kibibytes* (KiB, unità di 1024 bytes) `M` per *mebibytes* (MiB, unità di $1024^2 = 1.048.576$ bytes) `G` per *gibibytes*, (GiB, unità di $1024^3 = 1.024$ MiB) I prefissi + o − se utilizzati implicano un *limite inferiore* o *superiore* (non compresi).Ad esempio, con `-size +20M` il file matcherà se più grande di 20 MiB. Importante che il valore calcolato è arrotondato superiormente all'unità espressa, quindi `-size -1M` non è equivalente a `-size -1048576c`, il primo matcherà solo file vuoti, il secondo da 0 fino a 1MiB - 1 byte, quindi 1.048.575 bytes
`-atime n`	*access time* specificherà l'ultima volta in cui il file è stato acceduto (lettura o scrittura) espresso in $n*24h$ fa. Poiché la parte frazionaria è ignorata, `-atime +1` matcherà se il file sarà acceduto almeno 2 giorni fa
`-amin n`	vedi `-atime` con `n` indicherà minuti al posto di gg
`-ctime n`	*changed time* specificherà l'ultima volta in cui lo status[56] del file è stato modificato espresso in $n*24h$ fa. Per la parte frazionaria, riferirsi a quanto scritto per `-atime`
`-cmin n`	vedi `-ctime` con `n` indicherà minuti al posto di gg
`-mtime n`	`n` specificherà l'ultima volta in cui il file è stato modificato espresso in $n*24h$ fa. Per la parte frazionaria, riferirsi a quanto scritto per `-atime`
`-mmin n`	vedi `-mtime` con `n` indicherà minuti al posto di gg
`-path pattern`	il pattern è matchato con l'intero pathname, non trattando

	i metacaratteri come tali, quindi `/` e `..` Ad esempio `find . -path "./sr*sc"` avrà match con un path che potrebbe essere `./src/misc`'
`-perm mode`	i permessi[57] del file sono esattamente quelli indicati da *mode* (permessa espressione in ottale o simbolico). Per esempio, `-perm g=w` matcherà solo i file che avranno `mode 0020` (quindi solo permesso di scrittura del gruppo). È molto probabile che si voglia utilizzare la forma `-`, quindi ad esempio `-perm -g=w`, la quale matcherà qualsiasi file con i permessi di scrittura sul gruppo. Vedi l'***Esempio 4.54***
`-perm -mode`	tutti i bit di *mode* sono impostati per il file. Vedi l'***Esempio 4.55***
`-readable`	matcha file che sono leggibili dall'utente corrente
`-writable`	matcha file che sono scrivibili dall'utente corrente

Tabella 4.6: *Alcuni test del comando* `find`

OPERATORE	DESCRIZIONE
`( expr )`	forza la precedenza. Definiti nella shell, sono da quotare, quindi al posto di `(...)` si dovrà usare `\(...\)`
`!expr`	*True* se `expr` è *False*. Definito nella shell, il carattere `!` necessiterà protezione
`-not expr`	Equivalente a `!`, ma non *POSIX compliant*
`expr1 -a expr2`	*and* tra le due espressioni (operatore di default tra `expr1 expr2`, quindi se omesso)
`expr1 -and expr2`	equivalente a `expr1 expr2` ma non *POSIX compliant*
`expr1 -o expr2`	*or* tra le due espressioni
`expr1 -or expr2`	equivalente a `expr1 -and expr2` ma non *POSIX compliant*

Tabella 4.7: *I principali operatori del comando* `find`

AZIONE	DESCRIZIONE
`-delete`	rimuove il file
`-exec command ;`	esegue il comando
`-ls`	mostra l'elenco in maniera equivalente al formato

[52] v. 12.2 a pag. 131
[53] lo *UID* sarà argomentato in 17.2 a pag. 214
[54] lo *GID* sarà argomentato in 17.2 a pag. 214
[55] si può confrontare questa scala con le più moderne presenti in ***Tabella 24.8*** a pag. 405
[56] si intende la modificato del contenuto oppure dei metadati del file (quindi autorizzazione, proprietario, ...), aspetto argomentato in 17.2 a pag. 214
[57] i permessi saranno argomentati in 18 a pag. 227

	`ls -dils`
`-ok command ;`	come `-exec` ma chiede la conferma all'utente
`-print`	scrive il filename nello standard output, seguito da un *newline*
`-print0`	scrive i filename nello standard output, seguito dal *null character*. Ciò permette a filename contenenti caratteri quali *newline* o spazi di essere poi correttamente processati
`-printf format`	scrive secondo il formato specificato nello standard output. Alcuni[59] degli *escapes* utilizzabili sono: `\n` *newline* `\\` letteralmente il carattere \ `%%` letteralmente il carattere % `%a` l'ultimo accesso temporale al file `%b` la quantità di disco utilizzato dal file `%c` l'ultimo cambiamento temporale del *file status* `%d` la profondità nell'albero; 0 indica lo *starting point* `%f` il nome del file, con le directory precedente rimosse dall'output (quindi rimane solo il nome) `%g` nome del gruppo del file `%G` *GID* `%h` la directory che contiene il file medesimo; se non ci sono si userà il carattere `.` `%H` *starting point* sotto il quale è stato trovato `%i` l'*inode* del file (espresso in decimale) `%k` la quantità di spazio disco utilizzato espressa in blocchi di $1KB$ `%m` i bit che rappresentano i permessi associati (in ottale) `%M` i permessi associati in forma simbolica (espressa come nel comando `ls`) `%n` numero degli *hard link* al file `%p` nome del file `%P` filename a partire dallo *starting point* escluso `%s` grandezza del file espressa in byte `%t` ultima modifica temporale del file `%u` username del file `%U` *UID* `%y` tipo di file (come nel comando `ls -l`)
`-quit`	`exit` immediato, quindi alla prima occorrenza

Tabella 4.8: *Le principali azioni disponibili per il comando* `find`

Esempio 1

```
1 $ find .  -size +1G
```

Esempio 4.45: *Esempio di* `find`

[58] per un elenco esaustivo si veda `man find`

OPZIONE	DESCRIZIONE
`-d`	sinonimo di `-d` per compatibilità con i sistemi della famiglia *BSD*
`-depth`	processa il contenuto di ciascuna subdirectory ricorsivamente prima di procedere al contenuto della directory stessa. L'azione `-delete` implica quest'opzione.
`-maxdepth levels`	discende al livello `levels` a partire dagli *starting points*. Impostando il `levels` a 0 saranno esaminati solo i contenuti degli *starting points*
`-mindepth levels`	salterà il contenuto delle directory il cui livello di profondità (dallo *starting point* relativo) sia minore di quello impostato. Essendo lo starting point sempre 0, l'opzione `-mindepth 1` implicherà processare tutti i file ad esclusione di quelli degli *starting points*.
`-mount`	equivalente a `-xdev`, lasciato per compatibilità con altre versioni di `find`
`-xdev`	non discende directory di altri filesystem[60]

Tabella 4.9: *Le principali opzioni globali disponibili per il comando `find`*

Il comando cercherà nell'albero della directory corrente file più grandi di 1GiB e li invierà in output.

Esempio 2

```
1 $ find /tmp -type f,d,l
```

Esempio 4.46: *Esempio di `find` (2a)*

Ricerca in `/tmp` i file che siano di tipologia file, directory o symbolic link.
Sostanzialmente equivalente, ma linea un po' più lunga e maggiormente portabile:

```
1 $ find /tmp \( -type f -o -type d -o -type l \)
```

Esempio 4.47: *Esempio di `find` (2b)*

Esempio 3

```
1 $ find $HOME -mtime 0
```

Esempio 4.48: *Esempio di `find` (3)*

Ricerca nella propria *home directory* dei file che sono stati modificati nelle ultime 24 ore. Il valore 0 coprirà le ultime 24 ore poiché con `-mtime` la modifica sarà espressa in unità di 24 ore e la parte frazionaria sarà scartata.

Esempio 4

```
1 $ find /sbin /usr/sbin -executable \!  -readable -print
```

Esempio 4.49: *Esempio di `find` (4)*

Degli *starting points* indicati riporterà in output i file che sono eseguibili ma non leggibili.

Esempio 5

```
1 $ find .  -type f -exec file '{}'\;
```

Esempio 4.50: *Esempio di `find` (5)*

Esegue il comando `file`[61] per ogni file incontrato nella directory corrente o subdirectory. Da notare le protezioni utilizzare per evitare l'interpretazioni da parte della shell:

1. gli apici che racchiudono le parentesi graffe;
2. il carattere *backslash* antecedente il carattere `;`, ma anche in questo caso potevano essere utilizzati gli apici.

Esempio 6

```
1 $ find /tmp -name core -type f -print | xargs /bin/rm -f
```

Esempio 4.51: *Esempio di `find` (6)*

Il comando cercherà nell'albero della directory `/tmp` file il cui nome sia `core` e li cancellerà[62]. Da notare che lavorerà non correttamente se i file conterranno *newline*, spazi, i caratteri `'` o `"`.

Esempio 7

```
1 $ find /tmp -name core -type f -print0 | xargs -0 /bin/rm -f
```

Esempio 4.52: *Esempio di `find` (7)*

Il comando cercherà nell'albero della directory `/tmp` file il cui nome sia `core` e li cancella, processerà i filename contenente `'` o `!`, spazi o *newline* correttamente.

[61] v. **??** a pag. **??**

[62] il comando `xargs` sarà argomentato in 14.2 a pag. 170

Esempio 8

```
1 $ find /tmp -name core -type f -delete
```

Esempio 4.53: *Esempio di* `find` *(8)*

Equivalente all'esempio precedente, ma non ha ci saranno malfunzionamenti con i caratteri segnalati.

Esempio 9

```
1 $ find .  -perm 664
```

Esempio 4.54: *Esempio di* `find` *(9)*

Ricerca dalla *current directory* i file che hanno permesso di lettura e scrittura dal proprietario e gruppo, ma hanno solo il permesso di lettura degli altri utenti.
I file che matcheranno questo criterio dovranno avere esattamente i bit dei permessi come indicato (ad esempio il proprietario non può avere permesso di esecuzione), altrimenti non matcheranno. È utile confrontare l'esempio successivo.

Esempio 10

```
1 $ find .  -perm -664
```

Esempio 4.55: *Esempio di* `find` *(10)*

Equivalente all'esempio precedente, ma i permessi dovranno almeno essere quelli indicati, quindi è indifferente se il proprietario o il gruppo abbiano permesso di esecuzione oppure altri utenti abbiano permesso di scrittura o esecuzione.

4.10 Approfondimenti

4.10.1 Rimozione di file

In 4.6.1 a pag. 43 si è argomentato il comando `rm`.
Può accadere di incontrare nomi di file cui dovrà essere effettuato l'*escaping*[63], ma talvolta possono capitare nomi di file particolarmente *antipatici* da cancellare. Ad esempio per il seguente elenco:

```
1 $ ls -c1
2 'questo&file'
3 'questo file'
```

Esempio 4.56: *Elenco di file con sequenze antipatiche*

in cui entrambi i file sono marcati dalla shell con il carattere di *single quote* proprio per evidenziarne la presenza di caratteri per cui dovrà essere effettuato l'*escaping*. Per procedere all'eliminazione si potrà cancellarli uno alla volta, come di seguito:

[63] v. 2.3.3 a pag. 20

```
1 $ rm questo\&file.txt
2 $ rm questo\ file.txt
```

Esempio 4.57: *Cancellazione file precedenti tramite* escaping

o alternativamente:

```
1 $ rm 'questo&file.txt'
2 $ rm 'questo file.txt'
```

Esempio 4.58: *Cancellazione file precedenti tramite* quoting

in cui si ricorda che il *quoting* è realmente necessario solo per una sequenza di metacaratteri, quindi nel caso in oggetto solo per il carattere per cui si sarebbe effettuato l'*escaping*[64].

Utilizzando un pattern semplice[65] e possibile cancellarli anche con un solo comando:

```
1 $ rm questo*
```

Esempio 4.59: *Cancellazione file precedenti/2*

Talvolta i nomi di file possono però essere più complessi da trattare, ad esempio nell'elenco seguente:

```
1 $ ls -c1
2 --questo
3 -questo
4 'questo&file'
5 'questo file'
```

Esempio 4.60: *Elenco di file con sequenze molto antipatiche*

il primo nomefile inizia con due caratteri −, il secondo uno. Se provo a cancellare i file normalmente avrò qualche difficoltà:

```
1 $ rm -questo
2 rm: opzione non valida -- "q"
3 $ rm --questo
4 rm: unrecognized option '--questo'
```

Esempio 4.61: *Tentativi singoli di cancellazione dei file precedenti*

e nemmeno utilizzando un pattern alternativo:

```
1 $ rm *questo*
2 rm: unrecognized option '--questo'
```

Esempio 4.62: *Tentativo di cancellazione dei file precedenti tramite pattern*

poiché il pattern si è comunque espanso con i caratteri − come mostrato ricorrendo al comando `echo`:

```
1 $ echo *
2 --questo -questo questo file.txt questo&file.txt
```

Esempio 4.63: *Comprensione del pattern tramite* `echo`

[64] quindi avremmo ad esempio potuto cancellare il primo file con `rm 'questo&'file` oppure con `rm questo'&'file` o alternative valide

[65] v. 4.8.1 a pag. 45

In realtà il suggerimento è arrivato dai feedback ottenuti `unrecognized option` e `opzione non valida` poiché i file con iniziale - o -- sono in realtà considerati opzioni del comando `rm`. Quindi si potrà rimuoverli comunicando nella linea di comando che non sono opzioni.
Il metodo più corretto (e probabilmente il meno utilizzato) è *esplicitare* nella linea di comando il termine delle opzioni:

```
1 $ rm -- -questo
2 $ rm -- --questo
```

Esempio 4.64: *Utilizzare la sequenza -- per esplicitare la fine delle opzioni*

utilizzando la sequenza minimale --[66].
Un altro metodo che però può essere considerato un *workaround* è di seguito:

```
1 $ rm ./-questo
2 $ rm ./--questo
```

Esempio 4.65: *Workaround per la cancellazione dei file*

quindi far precedere al nome del file il path relativo, in altri termini è stato aggirato il problema non facendo più iniziare il nome con i trattini ma con la sequenza `./`.

Ovviamente si può utilizzare ingegno per la cancellazione, ad esempio ricorrendo al comando `find`:

```
1 $ find .  -name '*-questo' -exec rm
```

Esempio 4.66: *Workaround per la cancellazione dei file tramite* `find`

oppure rimuoverlo non tramite nome ma tramite *inode*[67], ad esempio:

```
1 $ ls -ic1
2 13375025 --questo
3 13372170 -questo
4 13375019 'questo file.txt'
5 13370253 'questo&file.txt'
6 $ find .  -inum 13375025 -exec rm
7 $ find .  -inum 13372170 -exec rm
```

Esempio 4.67: *Cancellazione tramite* inode

[66] poco utilizzata, è una sequenza utile per molti comandi

[67] l'argomento sarà affrontato nella sezione **??** a pag. **??**, al momento è sufficiente sapere che è un indice numerico per individuare il file univocamente all'interno del filesystem

4.11 Concetti chiave

alias: creazione di un nome di comando che la shell sostituirà con un comando (anche omonimo) con eventuali parametri aggiuntivi

case sensitive: è tale ogni operazione di analisi del testo *sensibile alle maiuscole*, cioè se si comporta differentemente se i caratteri sono maiuscoli o minuscoli (o una combinazione delle precedenti)

classi di carattere: genericamente è definita tale un insieme di caratteri

metacarattere: è un carattere jolly, cioè un carattere che, all'interno di una stringa, non rappresenta se stesso bensì un insieme di altri caratteri o sequenze di caratteri

directory: è un file contenitore di indici di file e della *parent directory*. Le più importanti sono la *root d.* (genitrice assoluta), la *home d.* (relativa all'utente, suo punto di approdo iniziale, definita da $\sim$), la *working d.* (anche *current d.*) coincide con la posizione corrente (è definita dal carattere `.`), la *parent d.* definita da `..`

escaping: è un metodo per effettuare il quoting di un singolo carattere, quindi non interpretarlo ma considerarlo letteralmente. Nella shell è il carattere \

file: è una sequenza di dati

globbing: metodo di espressione arricchita da metacaratteri affinché il pattern possa matchare con i filename desiderati

link: è un collegamento tra file, in particolare se *hard link* esso diviene indistinguibile dal file target, se *soft link* (*simbolico*) diviene solo un puntatore al file

path: anche *pathname*, indica il percorso si un elemento, in particolare se il path inizia dalla *root* si dice *assoluto*, negli altri casi è definito *relativo*

POSIX: acronimo di ***P**ortable **O**perating **S**ystem**I**nterface for Uni**x***, indica la famiglia degli standard definiti dall'*IEEE* (denominati formalmente *IEEE 1003*) per definire concetti base da seguire durante la realizzazione del SO. Il nome standard internazionale è *ISO/IEC 9945*

IEEE: acronimo di ***I**nstitute of **E**lectrical and **E**lectronics **E**ngineers*, è un'associazione internazionale di scienziati con l'obiettivo della promozione delle scienze tecnologiche

/: v. *directory root*

wildcard: v. *metacarattere*

4.12 Comandi introdotti nel capitolo

`convmv` ⇒ converte i filename da un encoding ad un altro

`find` ⇒ ricerca di file all'interno dell'albero indicato

`ln` ⇒ crea link tra file

`locate` ⇒ localizza file e directory

`ls` ⇒ mostra il contenuto di directory

`sl` ⇒ mostra una animazione con l'obiettivo di avvertire l'utente che accidentalmente ha digitato tale comando al posto di `ls`

`mkdir` ⇒ crea directory

`plocate` ⇒ v. `locate`

`pwd` ⇒ ***p**rint **w**orking **d**irectory*, mostra l nome dalla directory di lavoro (corrente)

`rmdir` ⇒ rimuove directory vuote

`touch` ⇒ modifica il timestamp di un file. Utile per crearne un file vuoto

`tree` mostra il contenuto delle directory in un formato ad albero

Capitolo 5

Contenuto di file

QUESTO capitolo avrà per oggetto la riproduzione del contenuto dei file sullo schermo.

5.1 `cat`

Il comando `cat` concatena i file in argomento e invia il risultato allo *standard output*. Utilizzato senza argomenti come nell'esempio seguente, lo *standard input* sarà copiato nello *standard output*.

```
1 $ cat
2 Hello world!!
3 Hello world!!
```

Esempio 5.1: *Comando `cat` senza argomenti e input come nell'esempio*

Se al comando `cat` non si fornirà nessun argomento, attenderà lo standard input e, al termine[1], replicherà quanto immesso. Solitamente l'utilizzo genericamente è del tipo:

```
1 $ cat file1 file2 ...
2 ...
```

Esempio 5.2: *Comando `cat` con argomenti*

in cui, al posto dei `...` della seconda linea, semplicemente sarà inviato in output la concatenazione dei contenuti degli stessi file nell'ordine immesso.
Anche se poco frequente, è possibile utilizzare il comando con le opzioni in ***Tabella 5.1***, come nell'***Esempio 5.3***.

5.1.1 `tac`

Similmente al comando `cat`, il comando `tac` invia il contenuto dei file in argomento allo standard output, ma inverte l'ordine delle linee, come indicato dal manuale mostrato in nell'***Esempio 5.4*** e poi mostrato nell'***Esempio 5.5***.

Ovviamente è possibile inserire nello standard input il contenuto da processare, come nell'***Esempio 5.9***. in cui si è digitato un Ctrl + D dopo aver inserito la prima linea

[1] tipicamente con le pressione del tasto Enter

```
$ cat cat.txt
Prima riga


Quarta riga
  Quinta riga

$ cat -nEsTv cat.txt
     1  Prima riga$
     2  $
     3  Quarta riga$
     4  ^IQuinta riga$
     5  ^Z$
```

Esempio 5.3: *Comandi* `cat` *senza e con opzioni*

```
TAC(1)                          User Commands

NAME
    tac - concatenate and print files in reverse

SYNOPSIS
    tac [OPTION]... [FILE]...

DESCRIPTION
    Write each FILE to standard output, last line first

    With no FILE, or when FILE is -, read standard input.
...
```

Esempio 5.4: *Segmento del manuale di* `tac`

```
$ cat testo_semplice.txt
Prima riga
Seconda riga
Terza riga
$ tac testo_semplice.txt
Terza riga
Seconda riga
Prima riga
```

Esempio 5.5: *Comandi* `cat` *e* `tac` *a confronto*

```
$ tac
Uno
 Due
  Tre
  Tre
 Due
Uno
```

Esempio 5.6: *Processamento dello standard input da* `tac`

OPZIONI	DESCRIZIONE
-A	equivalente a -vET
-E	aggiunge il carattere $ al termine di ogni linea
-n	aggiunge il numero di riga all'output
-s	sopprime la ripetizione di linee vuote
-T	mostra il carattere ⟺ sostituendolo con ^I
-v	mostra i metacaratteri[2] eccetto ↩ e ⟺ [3]

Tabella 5.1: *Opzioni principali del comando* `cat`

```
1  REV(1)                          User Commands
2
3  NAME
4      rev - reverse lines characterwise
5
6  SYNOPSIS
7      rev [option] [file...]
8
9  DESCRIPTION
10     The  rev utility copies the specified files to standard output,
11     reversing the order of characters in every line. If no files are
12     specified, standard input is read.
13 ...
```

Esempio 5.7: *Segmento del manuale di* `rev`

con scritto `Tre` e premuto Enter [4].

5.1.2 rev

Similmente al comando `tac`, il comando `rev` invia il contenuto dei file in argomento allo standard output, ma inverte l'ordine dei caratteri nelle linee, come indicato dal manuale mostrato in nell'***Esempio 5.7*** e poi mostrato nell'***Esempio 5.8***.

Anche con questo comando è possibile inserire nello standard input il contenuto da processare, come nell'***Esempio 5.9***. in cui si è digitato un Ctrl + D [5] solo al termine di tutti gli

[4] per eventuali dubbi sulla sequenza, si veda 15.1 a pag. 187

[5] si poteva optare per la sequenza Ctrl + C per terminare forzosamente il processo

```
1 $ rev testo_semplice.txt
2 agir amirP
3 agir adnoceS
4 agir azreT
```

Esempio 5.8: *Comando* `rev` *a confronto con quanto mostrato nell'****Esempio 5.5***

```
1 $ tac
2 Uno
3 onU
4  Due
5 euD
6 Tre
7  erT
```

Esempio 5.9: *Processamento dello standard input da* `tac`

inserimenti e i processamenti che avvengono immediatamente dopo aver premuto il tasto Enter [6].

5.2 `more`

Per visualizzare il contenuto di file lunghi, è comodo utilizzare un paginatore. Il comando `more` ha proprio tale scopo, mostrare il contenuto del file e fermarsi al termine dello screen, riportando sul fondo il messaggio:

```
--Ancora--
```

[7] e in alcune versioni di seguito anche la percentuale mostrata rispetto al totale come nella ***Figura 5.1***.

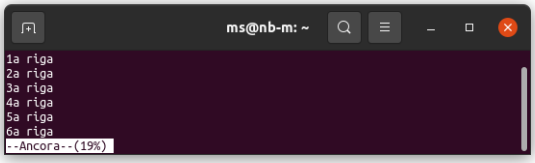


Figura 5.1: *Esempio del comando* `more`

La paginazione può andare avanti essenzialmente in diversi modi, i principali premendo:

Enter
: avanza di una linea;

[8]
: avanza di una pagina;

D
: avanza di mezza pagina;

Q o Ctrl + C
: esce.

[6] nel caso del comando `tac` è necessario comunicare con il Ctrl + D il termine dell'immissione poiché il comando dovrà processare *tutte le linee*, in questo caso è sufficiente il *newline* al termine della linea per iniziare il processamento in quanto l'oggetto è la singola linea immessa

[7] più genericamente riportano un `--More--`

[8] barra spaziatrice

Per esempio:

```
1 $ more testo.txt
2 1a riga
3 2a riga
4 3a riga
5 4a riga
6 5a riga
7 6a riga
8 --Ancora--(19%)
```

Esempio 5.10: *Paginazione tramite* `more/1`

premendo Enter si avanza di una linea:

```
1 2a riga
2 3a riga
3 4a riga
4 5a riga
5 6a riga
6 7a riga
7 --Ancora--(22%)
```

Esempio 5.11: *Paginazione tramite* `more` *(2)*

e premendo [ ] si avanza di una pagina:

```
1 8a riga
2 9a riga
3 10a riga
4 11a riga
5 12a riga
6 13a riga
7 --Ancora--(42%)
```

Esempio 5.12: *Paginazione tramite* `more` *(3)*

premendo Ctrl + C si termina tornando al prompt.

Il comando è utile nella sua semplicità, ma alcune limitazioni come ad esempio *tornare indietro/all'inizio* del sorgente potrebbero essere eccessive. Nella sezione 5.3 si porrà rimedio.

5.3 `less`

Un piccolo gioco di parole è *less is more*, per il comando `less` è sicuramente indovinato poiché, ad esempio, non c'è la limitazione segnalata nella sezione precedente.
Consultando il manuale:

```
1 LESS(1)                       General Commands Manual
2
3 NAME
4     less - opposite of more
5 ...
6 DESCRIPTION
7     Less is a program similar to more (1), but it has many more features.
        Less
8     does  not  have to read the entire input file before starting, so with
       large
```

```
 9      input files it starts up faster than text editors like vi  (1).   Less
          uses
10      termcap  (or terminfo on some systems), so it can run on a variety of
        termi-
11      nals.  There is even limited support for hardcopy terminals.  (On a
        hardcopy
12      terminal,  lines  which  should be printed at the top of the screen
        are pre-
13      fixed with a caret.)
14  ...
```

Esempio 5.13: *Segmento del manuale di* `less`

Ad esempio, sullo stesso file dell'***Esempio 5.10***:

```
1 $ less testo.txt
2 1a riga
3 2a riga
4 3a riga
5 4a riga
6 5a riga
7 6a riga
8 :
```

Esempio 5.14: *Paginazione tramite* `less`

oltre ai comandi di avanzamento segnalati in 5.2 e a tante altre opzioni, è possibile utilizzare i tasti e rispettivamente per scorrere e tornare indietro nel testo.

Da segnalare che il comando `less` è attivato anche per accedere alla documentazione via `man`.

5.4 Inizio e fine

Talvolta si è interessati solamente alla porzione iniziale o finale di un file, la prima di utilità generica per avere un'idea dell'intestazione del file, la finale utile specialmente con i file di log[9]. La sezione 5.4.1 avrà ad oggetto la parte iniziale, la successiva sezione 5.4.2 la finale.

5.4.1 `head`

Il comando `head` mostra di default le prime 10 linee dei file usati come argomento[10], si veda l'***Esempio 5.15***.
In ***Tabella 5.2*** sono sintetizzate le opzioni principali.
Tra le opzioni riportate, la più frequente è `-n` con la quale si potrà specificare un numero differente di linee, come nell'***Esempio 5.16***.
Solitamente con `head` il parametro numerico si userà con valori maggiori di 10, proprio per capire se si sta esaminando il file corretto.
Inoltre, per tutte le opzioni, si potrà aggiungere al numero `N` un suffisso per indicare un moltiplicatore come sintetizzato in ***Tabella 5.3***[11].

[9] saranno mostrati in diverse sezioni, maggiormente presenti nel capitolo **??** a pag. **??**
[10] ovviamente se l'argomento non c'è, il comando esaminerà lo *standard input*
[11] si può confrontare questa scala con le più moderne presenti in ***Tabella 24.8*** a pag. 405

```
1 $ head testo.txt
2 1a riga
3 2a riga
4 3a riga
5 4a riga
6 5a riga
7 6a riga
8 7a riga
9 8a riga
10 9a riga
11 10a riga
12 $
```

Esempio 5.15: *Comando* `head` *su file di testo*

OPZIONI		DESCRIZIONE
BREVE	ESTESA	
`-c [-]N`	`--bytes [-]N`	indica il numero di byte `N` da stampare. Se sarà indicato `-N`, l'output sarà tutto il file ad esclusione degli ultimi byte indicati da `N`
`-n [-]N`	`--lines [-]N`	specifica un numero `N` di linee. Se sarà indicato `-N`, l'output sarà tutto il file ad esclusione delle ultime linee indicate da `N`

Tabella 5.2: *Opzioni principali del comando* `head`

5.4.2 `tail`

Il comando `tail` mostra di default le ultime 10 linee dei file usati come argomento[12].
Analogamente a quanto mostrato nella sezione precedente, si mostra l'***Esempio 5.17***.
In ***Tabella 5.4*** sono sintetizzate le opzioni principali del comando e, ad eccezione di `-f`, si nota che esse sono equivalenti ma opposte a quelle della ***Tabella 5.2***, proprio per la natura contrapposta dei due comandi.
Quindi, analogamente all'***Esempio 5.16***, con l'opzione `-n` si potrà specificare un numero differente di linee, come nell'***Esempio 5.18***.
Sia per l'opzione `-l` sia per `-c` si potrà aggiungere, al numero `N`, un suffisso per indicare un moltiplicatore come sintetizzato nella ***Tabella 5.3***.

[12] ovviamente se l'argomento non c'è, il comando esaminerà lo *standard input*

```
1 $ head -n 5 testo.txt
2 1a riga
3 2a riga
4 3a riga
5 4a riga
6 5a riga
7 $
```

Esempio 5.16: *Comando* `head -n` *su file di testo*

SUFFISSO	VALORE
b	512
kB	1000
K	1024
MB	$1.000^2 = 1.000kB = 1.000.000$
M	$1.024^2 = 1.024K = 1.048.576$
GB	$1.000^3 = 1.000MB = 1.000.000.000$
G	$1.024^3 = 1.024M = 1.073.741.824$
TB	$1.000^4 = 1.000GB = 1.000.000$
T	$1.024^4 = 1.024G = 1.099.511.627.776$
PB	$1.000^5 = 1.000TB$
P	$1.024^5 = 1.024T$
EB	$1.000^6 = 1.000PB$
E	$1.024^6 = 1.024P$

Tabella 5.3: *Suffissi e valore associato*

```
1 $ tail testo.txt
2 20 riga
3 21a riga
4 22a riga
5 23a riga
6 24a riga
7 25a riga
8 26a riga
9 27a riga
10 28a riga
11 29a riga
12 $
```

Esempio 5.17: *Comando* `tail` *su file di testo*

```
1 $ tail -n 4 testo.txt
2 26a riga
3 27a riga
4 28a riga
5 29a riga
6 $
```

Esempio 5.18: *Comando* `tail -n` *su file di testo*

OPZIONI		DESCRIZIONE
BREVE	ESTESA	
`-c [+]N`	`--bytes [+]N`	indica il numero di byte `N` da stampare. Se sarà indicato `+N`, l'output sarà tutto il file a partire dal byte indicato da `N`
`-f`	`--follow`	aggiunge output appena al file è aggiunto contenuto
`-n [+]N`	`--lines [+]N`	specifica un numero `N` di linee. Se sarà indicato `+N`, l'output sarà tutto il file a partire dalla linea indicata da `N`

Tabella 5.4: *Opzioni principali del comando* `tail`

Come già accennato, per questo comando è spuntata l'opzione `-f`, fondamentale per i file di log. Esso permette al comando di restare in attesa che si immetta contenuto nel file e lo invierà in output fino alla pressione di un Ctrl + C da parte dell'utente. Si mostra un esempio parzialmente oscurato:

```
1 $ tail -n4 -f /var/log/kern.log
2 mese  d hh:mm:ss nb-m kernel: [650430.679820] usb 1-2: Manufacturer:
      SIGMACHIP
3 mese  d hh:mm:ss nb-m kernel: [650430.683915] input: SIGMACHIP Usb Mouse
      as /devices/pci0000:00/0000:00:14.0/usb1/1-2/1-2:1.0/0003:1C4F
      :0051.0016/input/input172
4 mese  d hh:mm:ss nb-m kernel: [650430.747773] hid-generic 0003:1C4F
      :0051.0016: input,hidraw0: USB HID v1.10 Mouse [SIGMACHIP Usb Mouse]
      on usb-0000:00:14.0-2/input0
5 mese  d hh:mm:ss nb-m kernel: [651609.427554] usb 1-2: USB disconnect,
      device number 74
6 $
```

Esempio 5.19: *Comando* `tail -f` *su file di testo*

5.5 Concetti chiave

concatenazione: immediatamente dopo un contenuto (ad esempio di un file) ne si aggiunge un altro (ad esempio contenuto di un secondo file)

paginazione: mostrare solo una porzione del contenuto in base alle dimensioni della console

5.6 Comandi introdotti nel capitolo

`cat` ⇒ concatena file e li mostra allo standard output

`head` ⇒ mostra la prima parte del file (opposto a `tail`)

`less` ⇒ *less is more*, più moderno del paginatore `more`, pagina il contenuto consentendo lo scroll sia in forward sia in backward

`more` ⇒ paginatore di contenuto primitivo che consente solo l'avanzamento

`rev` ⇒ inverte l'ordine dei caratteri delle linee

`tac` ⇒ opposto del comando `cat`, concatena i file e li mostra invertendo l'ordine delle linee

`tail` ⇒ mostra la parte finale del file (opposto a `head`)

Capitolo 6

Editor di file

QUESTO capitolo sarà molto breve, elenca e descrive in maniera estremamente breve i più popolari editor testuali a disposizione.

6.1 Pico

Pico, acronimo di ***Pi**ne **co**mposer*, è un editor di testo scritto in *C* per sistemi operativi *Unix-like*. Il *progetto GNU*, a causa della licenza[1] di Pico, sviluppò da parte del di un editor clone chiamato *Nano*. Attualmente nella distribuzione dell'autore, il comando `pico` è un link al comando `nano`, prossimo editor esaminato.

6.2 Nano

nano è un semplicissimo editor che somiglia molto all'editor *Pico*, ma è *free software* ed implementa alcune features che mancano a quest'ultimo, tra i quali:

- apertura di file multipli;
- scrolling per linea;
- undo/redo[2];
- syntax highlighting;
- numerazione delle linee.

Nella ***Figura 6.1*** un file pronto per essere modificato con `nano`.

[1] consultando l'url `https://www.nano-editor.org/dist/v2.8/README` si legge *[...] Pine suite does not use the GPL or a GPL-friendly license, and has unclear restrictions on redistribution. Because of this, Pine and Pico are not included with many GNU/Linux distributions. [...]*

[2] sintetizzano le comuni azioni di annullare o ripetere l'ultima azione (e quindi memorizzare uno storico di operazioni e annullarle/ripeterle)

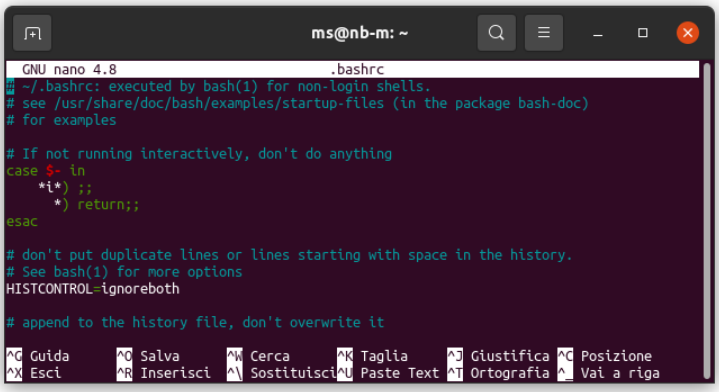


Figura 6.1: *Esempio di editing tramite* `nano`

6.3 Vim

Vim, o ***Vi IMproved***, è un editor di testo FOSS e multipiattaforma, nato per fornire una versione migliorata di *Vi*. Attualmente è molto diffuso tra programmatori ed utenti di sistemi operativi *Unix-like*, nella ***Figura 6.2*** un file pronto per essere modificato con `vim`. La

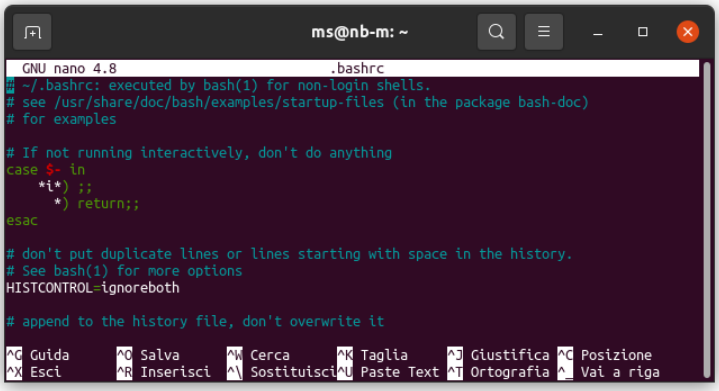


Figura 6.2: *Esempio di editing tramite* `vim`

caratteristica principale è l'essere *modale*, cioèin base alla modalità di utilizzo la digitazione avrà effetti differenti, in particolare sarà del testo inserito (come un qualunque editor) oppure sarà una sequenza di comandi. L'editor può essere in varie modalità, le principali sono sintetizzate nella ***Figura 6.3***, la quale riporta anche la sequenza di tasti per passare da una all'altra.
Di seguito un esempio di chiamata su un file da editare:

```
1 $ vi testo.txt
```

Esempio 6.1: *Edit di un file tramite* `vi`

in cui si è digitato semplicemente `vi` poiché si può notare che nel mio sistema:

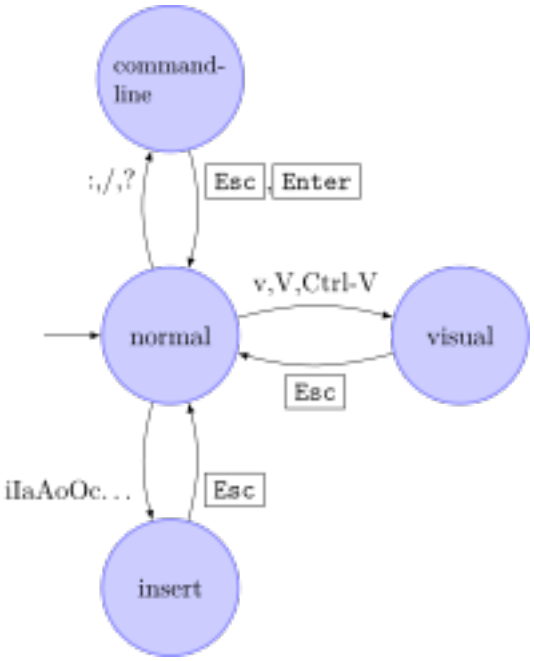


Figura 6.3: *Esempi di modalità dell'editor* Vim *e passaggi tra le varie modalità*

```
1 $ which vi
2 /usr/bin/vi
3 $ ls -l /usr/bin/vi
4 lrwxrwxrwx 1 root root 20 nov 28  2020 /usr/bin/vi -> /etc/alternatives/vi
5 $ ls -l /etc/alternatives/vi
6 lrwxrwxrwx 1 root root 18 nov 28  2020 /etc/alternatives/vi -> /usr/bin/
      vim.basic
7 $ which vim
8 /usr/bin/vim
9 $ ls -l /usr/bin/vim
10 lrwxrwxrwx 1 root root 21 nov 28  2020 /usr/bin/vim -> /etc/alternatives/
      vim
11 $ ls -l /etc/alternatives/vim
12 lrwxrwxrwx 1 root root 18 nov 28  2020 /etc/alternatives/vim -> /usr/bin/
      vim.basic
```

Esempio 6.2: `vi` *al posto di* `vim`

quindi digitare il comando `vi` o il comando `vim` è (in questo caso) equivalente.
In ultimo si segnala il comando `vimtutor`, utile per chi volesse imparare i primi comandi *Vim*.

6.4 Emacs

Emacs è un editor di testo libero multipiattaforma estremamente versatile, molto popolare fra i programmatori per la sua estensibilità e personalizzazione. È stato sviluppato origi-

nariamente come insieme di ***editor macros*** per l'editor *TECO*[3].
Nella ***Figura 6.4*** un file pronto per essere modificato con `emacs`, con console splittata in vari frame per altre utilità.

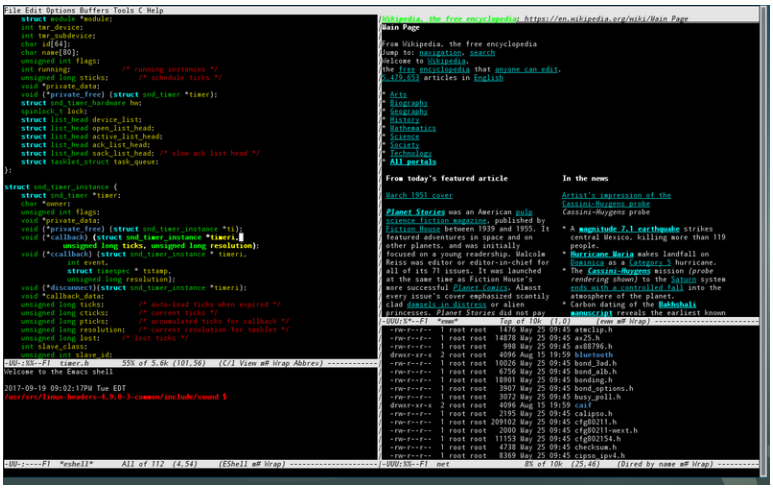


Figura 6.4: *Esempio di editing tramite* `Emacs`

Di seguito un esempio di chiamata su un file da editare:

```
1 $ emacs testo.txt
```

Esempio 6.3: *Edit di un file tramite* `emacs`

6.5 La guerra degli editor

Esiste una simpatica *guerra filosofica*[4] tra chi afferma che il miglior editor di testi sia *Emacs* e chi afferma sia *Vim*.
Tra i punti maggiori spicca quello dell'emulazione, in particolare *Emacs* è in grado di emulare autonomamente *Vim*, invece *Vim* deve utlizzare script di shell per riuscire ad emulare *Emacs*.

[3] anch'esso un acronimo: ***T**ext **e**ditor and **co**rrector*
[4] anni fa avrei scritto *guerra di religione*, oggi sarebbe considerata dai più un'espressione infelice

6.6 Concetti chiave

editor: software per la composizione di testi

syntax highlighting: con *colorazione della sintassi* si intende la caratteristica di un software di visualizzare un testo con differenti colori e/o font in base alle regole sintattiche (relativamente al linguaggio in cui è scritto il file)

6.7 Comandi introdotti nel capitolo

`emacs` ⇒ ***e**ditor **mac**ros*, editor FOSS multipiattaforma molto popolare

`nano` ⇒ *Nano* è un editor ispirato da *Pico*, ma è free software

`pico` ⇒ v. `nano`

`vi` ⇒ v. `vim`

`vim` ⇒ *Vim* (***Vi IM**proved*) è un editor di testo FOSS e multipiattaforma molto popolare

`vimtutor` ⇒ breve corso per chi volesse imparare *Vim*

Capitolo 7

Ricerca tramite espressioni regolari

QUESTO capitolo affronta l'argomento delle *espressioni regolari* solo superficialmente. Una migliore trattazione sempre introduttiva ma maggiormente argomentata la si può trovare in nota[1], ma per approfondire con maggior profondità si consigliano i manuali:

- `re_format`, `pcre` e `pcresyntax` accessibili tramite `man`;
- `perlrequick`, `perlre` e `perlretut` accessibili tramite `perldoc`[2].

Nel seguito sarà presentata la sintassi per la formazione delle espressioni regolari, successivamente sananno mostrati esempi tramite l'utilizzo del comando `grep`.

7.1 Terminologia

Una *espressione regolare* è un pattern definito da un linguaggio per descrive un insieme di stringhe. Tecnicamente è definita da una sequenza di espressioni minori e operatori che, combinati assieme, formano l'espressione regolare voluta.

7.2 Regole di base

L'elemento costruttivo di base di una *regexp*[3] sono le *regexp minimali* che corrispondono ad un singolo carattere. La quasi totalità dei caratteri, comprese le lettere alfabetiche e le cifre sono di questo tipo. I metacaratteri per essere trattati letteralmente devono essere quotati facendo precedere loro il carattere di escaping `\`.
Sono proprio i metacaratteri gli elementi che permettono espressività ad una *regexp*. Le varianti di seguito esaminate si distinguono proprio nella differente flessibilità espressiva, e con l'avanzare delle sezioni si potrà ricorrere ad un numero maggiore di metacaratteri.

[1] all'indirizzo web https://one2zero.it/repository/espressioni_regolari.pdf

[2] quindi tramite la sequenza `perldoc argomento` sostituendo ad `argomento` uno dei manuali proposti

[3] intuibile contrazione di ***r**egular **e**xpression*

Nell'utilizzo delle regexp nella shell, è utile inserirle tra apici o doppi apici per evitare di dimenticare l'escaping dei metacaratteri della shell stessa.

7.3 Varianti delle espressioni regolari

7.3.1 Basic regular expression

Le **BRE** sono le ***b**asic **r**egular **e**xpression*, permettono la minor espressività attivando solo i pochi operatori mostrati in ***Tabella 7.1***.

OPERATORE	DESCRIZIONE
`.`	il metacattere matcha con qualsiasi carattere
`[...]`	le parentesi attivano il match di un singolo carattere contro una lista di caratteri (che sarà al posto dei `...`)
`[.-.]`	le parentesi attivano il match di un singolo carattere contro una range di caratteri (il primo carattere dell'intervallo precederà il simbolo −)
`[^...]`	complementare alla precedente, al posto dei `...` sarà inserita una lista o range di caratteri da escludere nel match
`*`	l'asterisco indica che il carattere precedente sarà in una sequenza che partirà da 0
`^`	l'accento circonflesso è un'ancora di inizio linea, cioè indica che la sequenza che lo segue dovrà matchare con l'inizio della linea
`$`	il simbolo del dollaro è un'ancora di termine linea, cioè indica che la sequenza che lo precede dovrà matchare con la fine della linea

Tabella 7.1: *Metacaratteri utilizzabili nelle* BRE

Di seguito alcune regexp:

```
1 ".at"
2 "[hc]at"
3 "[^b]at"
4 "^[hc]at"
5 "[hc]at$"
```

Esempio 7.1: *Esempi di regexp variante BRE*

che rispettivamente matchano:

- tre caratteri, di cui gli ultimi due devo essere `at`;
- le stringhe `hat` o `cat`;
- equivalente alla prima espressione eccetto che non matcherà la stringa `bat`;
- le stringhe `hat` o `cat` ma all'inizio della linea;
- le stringhe `hat` o `cat` ma al termine della linea.

7.3.2 Extended regular expression

Le **ERE** sono le ***e**xtended **r**egular **e**xpression*.
Per queste sono attivati ulteriori metacaratteri riportati nella ***Tabella 7.2***.

METACHR	DESCRIZIONE
`(...)`	*capturing group*, definisce una subespressione che potrà essere richiamata successivamente tramite `\n`, con `n` singola cifra (v. `\n`)
`\n`	con `n` singola cifra, indica la subespressione creata precedentemente (quindi gruppo indicato dall'*ennesima* parentesi)
`+`	il carattere precedente dovrà essere presente una o più volte
`?`	il carattere precedente dovrà essere presente zero o una volta
`|`	il match è per l'elemento precedente o successivo
`{m}`	l'elemento precedente dovrà essere presente esattamente `m` volte
`{m,n}`	l'elemento precedente dovrà essere presente da `m` a `n` volte
`{m,}`	l'elemento precedente dovrà essere presente almeno `m` volte
`{,n}`	l'elemento precedente dovrà essere presente massimo `n` volte

***Tabella 7.2:** Metacaratteri utilizzabili nelle* ERE

Nell'***Esempio 7.2*** sono mostrate alcune regexp che rispettivamente matchano:

- `aac` oppure `a-c` oppure `aàc`;
- sigolo carattere che dovrà essere `a` o `.` `c`;
- con `a` oppure `b` oppure `c`;
- lettera minuscolo da `a` a `z`;
- `a` o `b` o `c` o da `x` a `z`;
- identico al punto precedente;
- `a` o `b` o `c` o `-`; quest'ultimo carattere sarà trattato letteralmente se nelle parentesi quadre è posizionato all'inizio o alla fine (o successivamente al carattere ^se presente come primo carattere);
- identico al punto precedente;

```
1  a.c
2  [a.c]
3  [abc]
4  [a-z]
5  [abcx-z]
6  [a-cx-z]
7  [abc-]
8  [-abc]
9  []abc]
10 [^abc]
11 [^a-z]
12 ab*c
13 [xyz]*
14 ab+c
15 [xyz]+
16 (ab)+
17 ab?c
18 z(ab)?x
19 abc|def
20 a{3,5}
21 ba{,2}b
22 ((ab)+c\1)
23 ((ab)+c\2)
```

Esempio 7.2: *Esempi di regexp variante ERE*

- i caratteri `]` o `a` o `b` o `c`. Poiché il carattere \non ha funzione di *escape*, il carattere `]` può essere incluso nella sequenza se inserito come primo carattere (o, analogamente a quanto scritto prima, successivamente al carattere ^se presente come primo carattere);
- qualsiasi carattere ad esclusione di `a` o `b` o `c`;
- qualsiasi carattere che non sia un carattere minuscolo (dalla `a` alla `z`)
- `ac` o `abc` o `abbc` o `abbbc` o ...
- *stringa vuota* o `x` o `y` o `z` o `xx` o `xy` o ... o `zyx` o ... o `xyzzy` o ...;
- `abc` o `abbc` o `abbbc` o ... ma non `ac`
- `x` o `y` o `z` o `xx` o `xy` o ... o `zyx` o ... o `xyzzy` o ...;
- `ab` o `abab` o `abababab` o ...;
- `ac` o `abc`;
- `zx` o `zabx`;
- `abc` o `def`;
- `aaa` o `aaaa` o `aaaaa`;
- `bb` o `bab` o `baab`;
- il riferimento `\1` siè utilizzato prima che il primo raggruppamento termini, quindi la regexp non è valida;
- `\2` è il riferimento al raggruppamento `ab`. quindi `abcab` o `ababcab` o ....

7.3.3 Classi di carattere e internazionalizzazione

Questa sezione è sovrapponibile a quanto riportato in 7.3.3 a pag. 48, compreso la ***Tabella 4.5***.

7.3.4 Posix BRE e Posix ERE

Lo standard POSIX è stato inglobato sia nelle *basic regular expression* sia nelle *extended regular expression* e ciò ha comportato una convergenza delle prime verso le seconde, per cui la flessibilità espressiva delle **BRE** è divenuta pari alle **ERE** con una piccola variazione sintattica delle prime: i metacaratteri `?`, `+`, `{`, `|`, `(`, e `)` possono essere utilizzati precedendo agli stessi il carattere `\`, quindi si useranno rispettivamente `\?`, `\+`, `\{`, `\|`, `\(` e `\)`. Per chiarire il passaggio, si veda la ***Tabella 7.3***.

POSIX BRE	POSIX ERE
`\(ab\)*`	`(ab)*`
`ab\+c`	`ab+c`
`[xyz]\+`	`[xyz]+`
`\(ab\)\+`	`(ab)+`
`ab\?c`	`ab?c`
`\(ab\)\?`	`(ab)?`
`abc|def`	`abc|def`
`a\{3,5\}`	`a{3,5}`
`ba\{,2\}b`	`ba{,2}b`

Tabella 7.3: *Equivalenze tra espressioni regolari nelle varianti Posix BRE e ERE*

7.3.5 Perl compatible regular expression

Le **PCRE** hanno una sintassi più ricca[4] rispetto le *POSIX **E**xtended **R**egular **E**xpressions*. Ad esempio, con le espressioni *PCRE* è possibile stabilire se la stessa dovrà comportarsi in maniera *greedy* oppure *non greedy*[5]. Un esempio è il seguente: sia data la stringa `a bad dab`, per le regexp:

`a.*b/` $\overset{match}{\Longrightarrow}$ `a bad dab`

`a.*?b/` $\overset{match}{\Longrightarrow}$ `a b`

La differenza tra le due espressioni è nel comportamento greedy dell'entità `.*` e non greedy della stessa aggiungendo l'ultimo carattere `.*?`, quindi immediatamente dopo il quantificatore.
Questo e altri metacaratteri sono stati presi dal Perl per rendere i motori delle regexp più

[4] ancora più ricca è la sintassi utilizzabile nelle regexp del Perl, come indicato nei manuali della seconda catagoria ad inizio capitolo

[5] per maggior chiarezza, si intende la possibilità di comunicare al motore delle regexp se l'espressione dovrà essere *vorace* o meno. Nel seguito sarà fornito un esempio

potenti, pertanto i diversi linguaggi[6] e strumenti[7] si autodefiniscono *compatibili*.
Purtroppo si può constatare che le implementazioni non sono identiche e, soprattutto, implementano solo un sottoinsieme delle caratteristiche. Quindi, nonostante sia dichiarata compatibilità, la potenza espressiva è minore, quindi è lecito osservare e soprattutto utilizzare l'espressione[8] *diversamente compatibili*.

7.4 Ricerca da comando

Per la ricerca di pattern si utilizza il comando `grep`[9] le cui principali opzioni sono riportate in ***Tabella 7.4***.
Nella prossima sezione alcuni esempi riepilogativi del comando `grep` con le diverse varianti di regexp esaminate.

7.5 Esempi

Per gli esempi seguenti si partirà da questo contenuto:

```
1 $ cat testo.txt
2 THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.
3 this line is the 1st lower case line in this file.
4 This Line Has All Its First Character Of The Word With Upper Case.
5
6 Two lines above this line is empty.
7 And this is the last line.
8 $ cp testo.txt testo2.txt
```

Esempio 7.3: *Contenuto di file utilizzato nei prossimi esempi*

e l'ultima istruzione è per la copia su un secondo file che sarà poi utilizzato. Poi si crei questo file il cui contenuto è un po' maggiore:

```
1 $ cat testo3.txt
2 4. Vim Word Navigation
3
4 You may want to do several navigation in relation to the words, such as:
5
6  * e - go to the end of the current word.
7  * E - go to the end of the current WORD.
8  * b - go to the previous (before) word.
9  * B - go to the previous (before) WORD.
10 * w - go to the next word.
11 * W - go to the next WORD.
12
13 WORD - WORD consists of a sequence of non-blank characters, separated with
       white space.
14 word - word consists of a sequence of letters, digits and underscores.
15
16 Example to show the difference between WORD and word
17
18 * 192.168.1.1 - single WORD
```

[6] ad esempio Python, PHP, Ruby, Java, ...
[7] ad esempio exim, Microsoft's .NET Framework, ...
[8] chiaramente in senso dispregiativo
[9] esistono anche le varianti del comando suddetto quali `egrep`, `fgrep` e `rgrep`, ma sono deprecate a favore dell'utilizzo rispettivamente delle opzioni `-E`, `-F` e `-r` che sono in ***Tabella 7.4***

OPZIONI		DESCRIZIONE
BREVE	ESTESA	
`-e P`	`--regexp=P`	interpreta `P` come pattern. Utilizzata più volte, il match è tentato per tutti i pattern usati. L'opzione non è necessaria per definire `P`, ma è prevista per protezione verso un pattern che inizi con il carattere -
`-E`	`--extended-regexp`	interpreta il pattern come una *ERE*
`-F`	`--fixed-strings`	interpreta il pattern come una *fixed strings*, quindi non come una *regexp*
`-G`	`--basic-regexp`	interpreta il pattern come una *BRE*. Questo è il comportamento di default
`-i`	`--ignore-case`	attiva l'insensibilità tra maiuscole e minuscole
`-l`	`--files-with-matches`	mostra soltanto i nomi dei file che matchano
`-w`	`--word-regexp`	seleziona solo le linee che matchano per un'intera parola (quindi la sottostringa non dovrà essere tra caratteri alfanumerici)
`-A N`	`--after-context=N`	stampa anche le `N` linee successive al match
`-B N`	`--before-context=N`	stampa anche le `N` linee precedenti al match
`-C N`	`--context=N`	stampa sia le `N` linee precedenti, sia le `N` successive al match

Tabella 7.4: *Opzioni principali del comando* `grep`

```
19  * 192.168.1.1 - seven words.
20
21 6.1 Display N lines after match
22
23 -A is the option which prints the specified N lines after the match as
       shown below.
24
25 Syntax:
26 grep -A <N> "string" FILENAME
27
28
29 The following example prints the matched line, along with the 3 lines
       after it.
30
31 $ grep -A 3 -i "example" demo_text
32 Example to show the difference between WORD and word
33
34 * 192.168.1.1 - single WORD
```

```
35 * 192.168.1.1 - seven words.
```

Esempio 7.4: *Contenuto di file utilizzato nei prossimi esempi*

e per ultimo:

```
1 $ cat testo4.txt
2 a
3 b
4 c
5 d
```

Esempio 7.5: *Contenuto di file utilizzato nei prossimi esempi*

Esempio 1

Ricerca di una stringa in un file:

```
1 $ grep 'this' testo.txt
2 this line is the 1st lower case line in this file.
3 Two lines above this line is empty.
4 And this is the last line.
```

Esempio 7.6: *`grep` di stringa*

il cui output riporta le linee che matchano.

Esempio 2

Ricerca di una stringa su più file:

```
1 $ grep 'this' testo*.txt
2 testo2.txt:this line is the 1st lower case line in this file.
3 testo2.txt:Two lines above this line is empty.
4 testo2.txt:And this is the last line.
5 testo.txt:this line is the 1st lower case line in this file.
6 testo.txt:Two lines above this line is empty.
7 testo.txt:And this is the last line.
```

Esempio 7.7: *`grep` di stringa su più file*

in cui ciascuna linea riporta il file esaminato, poi il carattere `:` e infine la linea che matcha.

Esempio 3

Ricerca di una stringa in un file in maniera *case insensitive*:

```
1 $ grep -i 'the' testo.txt
2 THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.
3 this line is the 1st lower case line in this file.
4 This Line Has All Its First Character Of The Word With Upper Case.
5 And this is the last line.
```

Esempio 7.8: *`grep` di una stringa attivando il* case insensitive

in cui `the` fa il match anche con i caratteri maiuscoli.

Esempio 4

Ricerca di una espressione regolare in un file:

```
1 $ grep "lines.*empty" testo.txt
2 Two lines above this line is empty.
```

Esempio 7.9: *`grep` di una regexp*

in cui il match sarà con `lines above this line is empty`.

Esempio 5

Ricerca di una parola intera:

```
1 $ grep -w is testo.txt
2 this line is the 1st lower case line in this file.
3 Two lines above this line is empty.
4 And this is the last line.
```

Esempio 7.10: *`grep` di una parola intera*

in cui nelle linee riportate il parola `is` è tra caratteri di spazio. Se non fosse stata inserita l'opzione `-w` sarebbero matchate anche le parole `this` e `This`.

```
1 $ grep is testo.txt
2 this line is the 1st lower case line in this file.
3 This Line Has All Its First Character Of The Word With Upper Case.
4 Two lines above this line is empty.
5 And this is the last line.
```

Esempio 7.11: *`grep` non della parola intera*

Esempio 6

Unendo le opzioni dell'**Esempio 3** e **Esempio 5**:

```
1 $ grep -iw is testo.txt
2 THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.
3 this line is the 1st lower case line in this file.
4 Two lines above this line is empty.
5 And this is the last line.
```

Esempio 7.12: *`grep` di una parola intera attivando il case insensitive*

in cui nelle linee riportate il parola `is` maiuscola o minuscola, comunque tra caratteri di spazio.

Esempio 7

Su grossi file è spesso utile mostrare le linee dopo/prima/intorno attraverso i flag `-A`, `-B` e `-C`.

N linee dopo il match Di seguito l'esempio:

```
1 $ grep -A 2 "single WORD" testo3.txt
2  * 192.168.1.1 - single WORD
3  * 192.168.1.1 - seven words.
```

Esempio 7.13: *`grep` con linee dopo il match*

in cui solo mostrate 2 linee perché si è giunti al termine del file.

N linee prima del match Di seguito l'esempio:

```
1 $ grep -B 2 "single WORD" testo3.txt
2 Example to show the difference between WORD and word
3
4  * 192.168.1.1 - single WORD
```

Esempio 7.14: *`grep` con linee prima del match*

in cui sono anche le 2 linee precedenti quella di match.

N linee intorno al match Di seguito l'esempio:

```
1 $ grep -C 2 "single WORD" testo3.txt
2 word - word consists of a sequence of letters, digits and underscores.
3
4 Example to show the difference between WORD and word
5
6  * 192.168.1.1 - single WORD
```

Esempio 7.15: *`grep` con linee intorno al match*

in cui sono sia le 2 linee precedenti sia quella successive al match.

Esempio 8

Tramite il flag `-r` si attiverà la ricerca in tutti i file ricorsivamente

```
1 $ grep -r dada *
2 test/test1/test.txt:dadad
3 ...
```

Esempio 7.16: *`grep` ricorsivo*

e l'output sarà equivalente a quello dell'**Esempio 2**, in cui ciascuna linea riporta il file esaminato, poi il carattere `:` e infine la linea che matcha.

Esempio 9

Tramite il flag `-v` si inverte il match:

```
1 $ grep -v go test3.txt
2 4. Vim Word Navigation
3
4 You may want to do several navigation in relation to the words, such as:
5
6
7 WORD - WORD consists of a sequence of non-blank characters, separated with
      white space.
```

```
8 word - word consists of a sequence of letters, digits and underscores.
9
10 Example to show the difference between WORD and word
11
12  * 192.168.1.1 - single WORD
13  * 192.168.1.1 - seven words.
```

Esempio 7.17: *`grep -v` per invertire il match*

Esempio 10

Si cercano linee che non matchano nessuno dei pattern indicati:

```
1 $ grep -v -e a -e b -e c test4.txt
2 d
```

Esempio 7.18: *`grep` con più pattern in seguenza*

in cui è mostrata l'unica line che non ha match.

Esempio 11

Per mostrare solo i nomi dei file che matchano:

```
1 $ grep -l this test*.txt
2 testo2.txt
3 testo.txt
```

Esempio 7.19: *`grep -l` per solo i nomi dei file*

7.6 Concetti chiave

complemento: se *true*, il complemento è *false*; se *false* il complemento è *true*

classi di carattere: è una sequenza di caratteri o un intervallo o una classe prestabilita inserita in `[...]` al posto dei puntini, costituendo un insieme di alternative per il carattere da matchare

BRE: ***b**asic **r**egular **e**xpression*, v. 7.3.1

ERE: ***e**xtended **r**egular **e**xpression*, v. 7.3.2

espressione regolare: pattern definito da un linguaggio per descrivere un insieme di stringhe

greedy: l'espressione regolare manifesta la sua voracità nel matching; nelle regexp con Perl è possibile spegnere tale voracità aggiungendo il metacarattere `?` immediatamente dopo i quantificatori

PCRE: ***P**erl **c**ompatible **r**egular **e**xpression*, v. 7.3.5

regexp: contrazione di ***reg**ular **exp**ression*

grouping: è una subespressione tra `(...)` al posto dei puntini, che potrà essere richiamata successivamente tramite `\n` secondo l'ordine di creazione nella regexp

pattern: indica genericamente uno schema che solitamente distingue quanto cercato dal resto

`egrep` ⇒ deprecato a favore di `grep -E`

`grep` ⇒ deprecato a favore di `grep -F`

`grep` ⇒ ricerca pattern nei file

`perldoc` ⇒ accesso alla documentazione del linguaggio *Perl*

`rgrep` ⇒ deprecato a favore di `grep -r`

Capitolo 8

Standard I/O e redirezione

MOLTI dei comandi presentati finora hanno letto dati dallo *standard input*[1] e scritto sullo *standard output*[2]. Oltre i precedenti, gli *standard channels* sono formati da un terzo canale denominato *standard error*[3].
La differenza sostanziale tra lo standard output e error è nell'invio dell'output sul primo a meno che ci siano errori[4] i quali saranno inviati al secondo.
Più formalmente costituiscono i flussi standard (**standard streams**) per la comunicazione di un programma all'environment, entità assegnate appena inizia la sua esecuzione.

Nel capitolo essenzialmente si illustrano i metodi per la redirezione degli stessi flussi, quali ad esempio lo standard output su file, standard input prelevato da file, lo standard error buttato via, . . . , ma più importante sarà inviare lo standard output di un processo direttamente ad un altro processo, senza quindi parcheggiare il primo temporaneamente su file e poi il contenuto di quest'ultimo inoltrarlo come input al secondo processo.
Quest'ultimo passaggio permetterà di utilizzare i comandi finora presentati[5] come ingranaggi che saranno impiegati nella realizzazione di catene che costituiranno applicazioni complesse.

8.1 Standard I/O

Gli **standard channels** sono riportati in ***Tabella 8.1*** Solitamente ci si riferisce loro tramite

CHANNEL	NOME	ABBR.	DEVICE	ESEMPI D'USO
0	standard input	`stdin`	tastiera	input per programmi
1	standard output	`stdout`	monitor	output per programmi
2	standard error	`stderr`	monitor	messaggi di errori

Tabella 8.1: *Gli standard channels sono i 3 standard* file descriptors

[1] nel nostro caso la tastiera
[2] nel terminale o probabilmente nella console di una sessione grafica
[3] nella forma estesa *standard error output*
[4] ad esempio non esistenza del file di input, non permessi di accesso, errore sintattico nella composizione di comandi o espressioni, . . .
[5] ovviamente anche i successivi

l'abbreviazione (`stdin`, `stdout` e `stderr`) e nei comandi si utilizzano i numeri assegnati (rispettivamente 0, 1 e 2) che corrispondono ai valori dei **file descriptor**[6] assegnati all'avvio a ciascun processo. Tali valori numerici saranno utilizzati ripetutamente nel capitolo e diffusamente nel resto del documento.
La shell può redirigere questi canali, ad esempio può redirigere l'output di un processo su un file (eventualmente creandolo), oppure inviare il contenuto di un file come input ad un software.

8.2 Redirezione

È possibile redirigere lo `stdout` utilizzando l'operatore di shell >[7] come nell'***Esempio 8.1***.

```
1 $ ls -laF > lista_file.txt
2 $ less lista_file.txt
3 totale 8
4 drwxrwxr-x 2 ms ms 4096 nov 17 11:52 ./
5 drwxr-xr-x 8 ms ms 4096 nov 17 11:52 ../
6 -rw-rw-r-- 1 ms ms    0 nov 17 11:52 lista_file.txt
```

Esempio 8.1: *Redirezione output tramite >*

Se il file `lista_file.txt` non esiste, esso sarà creato. Se il file esiste, sarà sovrascritto. Per non sovrascrivere ma aggiungere output ad un file, si utilizza l'operatore >>[8] come nell'***Esempio 8.2***, che in ogni caso creerà il file qualora non dovesse esistere.

```
1 $ date >> lista_file.txt
2 $ cat lista_file.txt
3 totale 8
4 drwxrwxr-x 2 ms ms 4096 nov 17 11:52 ./
5 drwxr-xr-x 8 ms ms 4096 nov 17 11:52 ../
6 -rw-rw-r-- 1 ms ms    0 nov 17 11:52 lista_file.txt
7 ven 17 nov 2023, 12:01:58, CET
```

Esempio 8.2: *Redirezione output tramite >*

Per redirigere lo `stdin` si utilizza il carattere <[9]. Esso inoltrerà il contenuto di un file invece dell'input da tastiera. SI

```
1 $ cat < lista_file.txt
2 totale 8
3 drwxrwxr-x 2 ms ms 4096 nov 17 11:52 ./
4 drwxr-xr-x 8 ms ms 4096 nov 17 11:52 ../
5 -rw-rw-r-- 1 ms ms    0 nov 17 11:52 lista_file.txt
6 ven 17 nov 2023, 12:01:58, CET
```

Esempio 8.3: *Redirezione tramite <*

Ovviamente il comando `cat` avrebbe funzionato anche senza la redirezione, ma si può osservare che in questo caso non ha inviato direttamente allo `stdout` il contenuto del file, ma ha diretto quello che proveniva allo *stdin* tramite il metacarattere <, il quale ha inviato il contenuto del file. Si può notare nuovamente la questione in quest'altro esempio:

[6] in maniera estesa *standard POSIX **f**ile **d**escriptor* o abbreviata in **FD**. Saranno approfonditi in 12.6.5 a pag. 150

[7] simbolo *maggiore di*

[8] simbolo *molto maggiore di*

[9] simbolo *minore di*

```
1 $ wc lista_file.txt
2   5  35 173 lista_file.txt
3 $ wc < lista_file.txt
4   5  35 173
5 $ wc - < lista_file.txt
6   5  35 173 -
```

Esempio 8.4: *Redirezione tramite < al comando* `wc`

in cui il comando `wc` senza opzioni scrive il totale delle linee, delle parole e dei byte per ciascun file in argomento e, se nessun file è passato oppure è passato −, riporta tale conteggio per lo `stdin`.
Quindi nel primo caso ha effettuato il conteggio per il file di cui ha riportato anche il nome, nel secondo e terzo caso ha riportato il conteggio per lo *stdin* a cui ha inviato il contenuto il metacarattere <.

8.2.1 Redirezioni contemporanee

Le redirezioni riportate possono essere contemporanee come nel seguente esempio:

```
1 $ wc -lc < lista_file.txt > wc_line.txt
2 $ cat wc_line.txt
3   5 173
```

Esempio 8.5: *Redirezione contemporanee*

in cui il comando `wc` conta le linee e byte dallo `stdin` e invia il proprio `stdout` al file `wc_line.txt`.

8.2.2 Redirezione dello `stderr`

Oltre la redirezione dei canali `stdin` e `stdout`, è possibile la redirezione dello `stderr`. Come già scritto, se ad esempio durante l'esecuzione di un processo si verificheranno errori, questi saranno inviati allo `stderr`. Si potrebbe voler redirigere sia l'output sia gli errori (quindi sia lo `stdout` sia lo `stderr`) su file distinti o su un unico file, e lo si farà utilizzando i numeri associati ai canali secondo quanto riportato in ***Tabella 8.1***:

`0<`
: per lo `stdin`;

`1>`
: per lo `stdout`;

`2>`
: per lo `stderr`.

In accordo con i precedenti esempi, i numeri non sono obbligatori ad eccezione dello `stderr`. Si veda il seguente esempio:

```
1 $ find / -maxdepth 2 -ls > find_lista.txt
2 find: '/lost+found': Permesso negato
3 find: '/root': Permesso negato
```

Esempio 8.6: *Redirezione solo* `stdout` *su file*

in cui lo `stdout` è stato rediretto al file, lo `stderr` è invece visualizzato. Si può quindi ovviare anche con la redirezione dello `stderr`:

```
1 $ find / -maxdepth 2 -ls > find_lista.txt 2> find_errori.txt
2 $ cat find_errori.txt
3 find: '/lost+found': Permesso negato
4 find: '/root': Permesso negato
```

Esempio 8.7: *Redirezione anche* `stderr` *su file*

in cui il comando `cat` mostra gli errori rediretti al file, gli stessi errori mostrati allo screen nell'***Esempio 8.6***.
Inoltre, è possibile ridirigere un canale verso un altro canale utilizzando la sequenza `>&` come in questo esempio:

```
1 $ find / -maxdepth 2 -ls 1> find_maxdepth.txt 2>&1
```

Esempio 8.8: *Redirezione* `stderr` *nello* `stdout`

in cui gli errori mostrati nell'***Esempio 8.7*** sono stati inviati al file `find_maxdepth.txt` assieme all'ordinario output.
La redirezione contemporanea di `stdout` e `stderr` è molto utile nei log, ulteriori esempi si avranno nella sezione **??** a pag. **??**.

8.2.3 Altre redirezioni

Di seguito altre redirezioni possibili, utilizzate spesso in tutorial o documentazione varia online, molto utile per azioni di *copy/paste* di sequenze di comandi con eventuale costruzione di script.

Here document

Un *here document* è un flusso di input inviato ad un comando. La sintassi generica è la seguente:

```
1 $ comando [argomenti] << LABEL
2  ...
3 LABEL
```

Esempio 8.9: *Sintassi di un* here document

in cui:

`comando` può essere un generico comando che prevede immissione di stringhe;

`<<` è l'operatore di redirezione del flusso;

`LABEL` può essere una qualunque stringa che definisce i limiti del flusso.

Un esempio con `cat`:

```
1 $ cat << EOF
2 Hello
3 World!!
4 EOF
5 Hello
6 World!!
7 $
```

Esempio 8.10: *Sintassi di un* here document *con il comando* `cat`

Può essere utilizzata l'*espansione delle variabili*, ad esempio:

```
1 $ cat << EOF
2 $(echo Hello)
3 $(whoami)
4 EOF
5 Hello
6 ms
7 $
```

Esempio 8.11: *Sintassi di un* here document *con l'espansione delle variabili*

in cui solo il primo e l'ultimo carattere `$` sono inviati dal prompt, gli altri sono utilizzati per l'espansione della variabile di bash nella forma `$(variabile)`.
Si può però indicare che il flusso sia letterale, aggiungendo il *quoting*[10] al primo *label*:

```
1 $ cat << 'EOF'
2 $(echo Hello)
3 $(whoami)
4 EOF
5 $(echo Hello)
6 $(whoami)
7 $
```

Esempio 8.12: *Sintassi di un* here document *conflusso letterale*

in cui anche qui solo il primo e l'ultimo carattere `$` sono inviati dal prompt, gli altri sarebbero stati utilizzati per l'espansione della variabile di bash nella forma `$(variabile)`, ma con gli apici si è forzata la trattazione letterale.

Un esempio di creazione di uno script e sua esecuzione tramite un unico copia/incolla potrebbe essere:

```
1 $ cat << EOF > hello_world.sh
2 #!/usr/bin/bash
3
4 echo 'Hello world!!'
5 EOF
6 $ chmod +x hello_world.sh
7 $ ./hello_world.sh
8 Hello world!!
9 $
```

Esempio 8.13: *Un* here document *per creazione ed esecuzione di uno script*

in cui le fasi sono:

1. *creazione bash script* con nome file `hello_world.sh`;
2. la shell restituisce il prompt e si inserisce la linea di comando per *aggiungere il permesso di esecuzione* al file appena creato;
3. la shell ci restituisce il prompt e si inserisce la linea che *esegue* lo script;

[10] indifferentemente singoli o doppi apici

tutto da un unico copia/incolla.

Senza aggiungere ulteriori esempi, oltre l'*espansione delle variabili* si può utilizzare il *command substitution*, *espansione aritmetica*, ...[11].

Here string

L'*here strings* è una variante del precedente *here document* in cui l'input non è su più linee ma su singola linea.
Ovviamente anche qui il software dovrà essere interattivo, ad esempio si usi `bc`:

```
1 $ bc <<< 3*2
2 6
3 $
```

Esempio 8.14: *Un* here strings *con il comando* `bc`

Senza ripetersi, anche qui c'è l'*espansione delle variabili*, *command substitution*, ...[12].

8.3 Pipeline

Le *pipeline* sono un meccanismo con il quale si possono concatenare sequenze di comandi, in cui lo `stdout` del precedente costituisce lo `stdin` del successivo[13].
Per formare la pipeline, ciascun comando è connesso al successivo tramite l'operatore `|`[14], come nel seguente esempio:

```
1 $ ls -laF | less
```

Esempio 8.15: *Esempio di pipeline*

in cui non si è inviato l'output di `ls -laF` su di un file e poi consultato tramite il comando `less`, con i vantaggi:

- di scrivere tutto in maniera più compatta;
- evitare la produzione del file intermedio, evitando poi di dover cancellare il file temporaneo.

Ovviamente la sequenza di comandi in una pipeline può essere lunga (quasi) a piacere, e aumentando la lunghezza maggiore saranno i vantaggi rispetto la tediosità di aver eseguito comandi atomici e inviando per ciascuno lo `stdout` su file, poi come `stdin` al successivo, ..., fino all'ultimo comando, quindi producendo più file da dover cancellare.
Alle volte può essere utile memorizzare il flusso instradato nella pipeline in uno stadio intermedio, ad esempio perché ad un certo passaggio i dati sono utili per un altro task. Questo compito è svolto dal comando `tee` il quale girerà quanto ricevuto da `stdin` verso `stdout` e, se lanciato senza opzioni, copierà lo stesso su file in argomento sovrascrivendolo, come nell'***Esempio 8.16***.

[11] v. `man bash`, sezione `Here Documents`
[12] v. `man bash`, sezione `Here Strings`
[13] questa è la prima forma di ***i**nter-**p**rocess **c**ommunication* (**IPC**) incontrata in questo documento; l'argomento sarà affrontato al capitolo **??** a pag. **??**
[14] denominato *pipe*

```
1 $ ls -laF | tee lista_file.txt | less
2 totale 184
3 drwxrwxr-x 2 ms ms   4096 nov 19 17:00 ./
4 drwxr-xr-x 8 ms ms   4096 nov 19 14:06 ../
5 -rw-rw-r-- 1 ms ms    130 nov 19 01:18 find_errori.txt
6 -rw-rw-r-- 1 ms ms 139353 nov 19 01:18 find_lista.txt
7 -rw-rw-r-- 1 ms ms  18754 nov 19 01:26 find_maxdepth.txt
8 -rw-rw-r-- 1 ms ms    173 nov 17 12:01 lista_file.txt
9 ...
```

Esempio 8.16: *Esempio di pipeline con* `tee`

Se `tee` è lanciato con l'opzione `-a`[15], non sovrascriverà il file in argomento, ma aggiungerà quanto ricevuto in `stdin` al contenuto dello stesso file, oltre al solito invio verso lo `stdout`.

8.4 Command substitution

Il *command substitution* può essere utilizzato in linee che presentano contemporaneamente più comandi[16], assegnando precedenza ad alcuni di essi. L'interpretazione di questi produrrà per ciascuno uno *stdout* il quale sostituirà la medesima porzione valutata, successivamente sarà eseguita la linea di comando nel suo complesso. Il metodo storico per eseguire la sostituzione è il *backtick*, attivato tramite `` `...` ``, in cui la parte tra i backtick avrà precedenza. Nell'***Esempio 8.17*** si è continuato dal contenuto mostrato nell'***Esempio 8.2***

```
1 $ grep `date +%Y` lista_file.txt
2 ven 17 nov 2023, 12:01:58, CET
3 $ grep "`date '+%d %b'`" lista_file.txt
4 ven 17 nov 2023, 12:01:58, CET
```

Esempio 8.17: *Sostituzione tramite backtick*

Per ciascuna linea, la shell assegnerà al comando `date` la precedenza poiché posto tra backtick, e li eseguirà prima del resto. Successivamente l'output sostituirà i backtick e:

- il primo `grep` esegue un match sull'anno;
- il secondo esegue un match sul giorno e mese.

Quest'ultimo è stato riportato per rendere maggiormente complessa l'interpretazione del comando, rendendo la sottosequenza `"`date '+%d %b'`"` non banale a causa del doppio escaping.
Un'alternativa più moderna ma equivalente all'***Esempio 8.17*** è la sostituzione tramite l'espressione `$(...)` come di seguito:

```
1 $ grep $(date +%Y) lista_file.txt
2 ven 17 nov 2023, 12:01:58, CET
3 $ grep $(date '+%d %b') lista_file.txt
4 ven 17 nov 2023, 12:01:58, CET
```

Esempio 8.18: *Sostituzione tramite* `$()`

[15] per esteso `--append`
[16] utile anche nelle assegnazioni come vedremo in 22.1 a pag. 321

8.5 Filtri

Un'idea alla base dei sistemi *Unix-like* è mettere a disposizione tantissimi strumenti concettualmente *semplici* creati per risolvere task *minimali* con estrema efficienza. Questi software di utility possono essere considerati ingranaggi per costruire programmi più complessi. Ad esempio, poche utility incorporano procedure di ordinamento, ma molti software se ne avvalgono formando un pipeline con il comando `sort`[17].
Questa struttura modulare ha diversi vantaggi tra i quali:

- facilita il processo di sviluppo ai programmatori, i quali non necessitano di includere essi stessi routine già disponibili (ad esempio, il *sorting*);
- se un software di utilità riceverà un miglioramento[18], tutti i software che lo utilizzano usufruiranno del vantaggio, nella stragrande maggioranza dei casi senza necessità di cambiare nulla[19].

Questi strumenti che ricevono allo `stdin` e scrivono sullo `stdout` sono denominati **filtri**[20]. Se non riceveranno l'input tramite redirezione, questi software attenderanno l'input da tastiera che potrà essere terminato con un *newline*[21] e un Ctrl + D [22].

L'***Esempio 8.19*** sarà sul comando `grep -`, in cui il `-` è opzionale ma maggiormente esplicativo che il comando sarà in attesa dello `stdin`.

```
1  $ grep 12 -
2  1
3  12
4  12
5  123
6  123
7  321
8  145
9  12123
10 12123
11 as fa 12
12 as fa 12
13  Ctrl + D
```

Esempio 8.19: *Esempio del comando `grep` su `stdin` immesso da tastiera*

L'***Esempio 8.20*** mi è stato utile per ordinare le entry che saranno presentate nella sezione 8.7.

```
1 $ sort
2 |
3 2>
4 <
```

[17] si affronterà l'argomento in 9.2 a pag. 103
[18] ad esempio nell'efficienza oppure un *bug fix*
[19] si può intuire tale beneficio incontrando nei manuali le molte opzioni lasciate per retrocompatibilità, spesso lasciate lì proprio per non far modificare nulla agli utilizzatori ;-)
[20] anche *comandi di filtraggio*
[21] quindi premendo Enter
[22] intercettati dal driver del terminale, come argomentato in 15.1.3 a pag. 188, è interpretata come sequenza *end of file*. Ovviamente questa sequenza si applica solamente all'inserimento da tastiera e non su file

```
<<
<<<
>
>>
>&
`...`
$(...)
sort
tee
wc

 Ctrl + D

<
<<
<<<
>
>&
>>
`...`
|
$(...)
2>
sort
tee
wc
```

Esempio 8.20: *Esempio del comando* `sort` *su* `stdin` *immesso da tastiera*

in cui ho immesso un *newline* prima della sequenza terminale Enter e Ctrl + D semplicemente per comodità di visualizzazione.
L'***Esempio 8.21***, inserito con il medesimo scopo dell'esempio precedente, è maggiormente efficace per lo scopo dello scrivente.

```
$ sort | grep -e "[^[:alpha:]]"
|
2>
<
<<
<<<
>
>>
>&
`...`
$(...)
sort
tee
wc

 Ctrl + D

<
<<
<<<
>
>&
>>
`...`
|
$(...)
```

```
27 2>
```

Esempio 8.21: *Esempio del comando* `sort` *in pipeline con* `grep` *su* `stdin` *immesso da tastiera*

Concludendo, sono già stati esaminati alcuni filtri quali `cat`, `tac`, `rev`, `more`, `less`, `head`, `tail`, `grep`, `wc`, `tee`, nel capitolo 9 a pag. 103 ne saranno presentati ulteriori, e saranno utilizzati diffusamente nel resto del documento.

8.6 Concetti chiave

command substitution: sostituzione dello `stdout` a sequenze tra `` `...` `` oppure `$(...)`

file descriptor: per esteso *POSIX file descriptor* o abbreviato in *FD*, identificativo per ciascun processo delle risorse di input/output assegnate

here document: è un flusso di input inviato ad un comando

here strings: è un flusso di input ridotto ad una linea inviato ad un comando

IPC: è l'acronimo di ***i**nter-**p**rocess **c**ommunication*, cioè comunicazione tra processi

pipeline: successione di comandi in cui lo `stdout` del precedente costituisce lo `stdin` del successivo

standard channels: sono i flussi standard (**standard stream**) esistenti per la comunicazione tra un programma e il suo environment

stderr: abbreviazione di *standard error output*, v. *standard channels*

stdin: abbreviazione di *standard input*, v. *standard channels*

stdout: abbreviazione di *standard output error*, v. *standard channels*

8.7 Comandi introdotti nel capitolo

`<` ⇒ operatore di shell per la redirezione dello `stdin`

`<<` ⇒ operatore di redirezione per definire un *here document*

`<<<` ⇒ operatore di redirezione per definire un *here strings*

`>` ⇒ operatore di shell per la redirezione dello `stdout`

`>&` ⇒ redirezione di un canale ad un altro canale

`>>` ⇒ operatore di shell per eseguire l'append dello `stdout`

`` `...` `` ⇒ sequenza che permette il *command substitution*

`|` ⇒ operatore per la formazione della `pipeline`

`$(...)` ⇒ sequenza che permette il *command substitution*

`2>` ⇒ redirezione `stderr`

`sort` ⇒ ordina le linee di testo

`tee` ⇒ quanto ricevuto allo `stdin` lo copierà su file indicato e lo inoltrerà allo `stdout`

`wc` ⇒ scrive il numero di *newline*, parole e byte per ogni file

Capitolo 9

Data management

Si è conclusa l'ultima sezione del precedente capitolo con un elenco di filtri già incontrati. La prima sezione li riepilogherà indicando nuovamente la rispettiva funzione, le successive presenteranno altri filtri appartenenti alla medesima categoria.

9.1 Riepilogo

Di seguito il riepilogo dei filtri finora presentati:

- `cat` ⇒ concatena file e li mostra allo standard output
- `grep` ⇒ ricerca pattern nei file
- `head` ⇒ mostra la prima parte del contenuto
- `less` ⇒ pagina il contenuto consentendo lo scroll sia in forward sia in backward
- `more` ⇒ pagina il contenuto consentendo solo l'avanzamento
- `rev` ⇒ inverte l'ordine dei caratteri delle linee
- `tac` ⇒ ripropone il contenuto invertendo l'ordine delle linee
- `tail` ⇒ mostra la parte finale del contenuto
- `tee` ⇒ copia quanto ricevuto dallo `stdin` su file e lo ripropone allo `stdout`
- `wc` ⇒ riporta informazioni numeriche sul file

È già stato introdotto, ma nell'elenco manca il comando `sort` poiché sarà affrontato in maniera estesa nella prossima sezione.

9.2 sort

Il comando `sort` ordina le linee di testo secondo il criterio lessicografico ascendente in accordo a quanto descritto di seguito:

1. le linee che iniziano con un numero saranno precedenti rispetto quelle con le lettere;
2. le linee che iniziano con lettere che sono precedenti nell'alfabeto appariranno prima delle linee che iniziano con lettere successive;
3. linee con le medesima lettera alfabetica avranno precedenza se scritte in minuscolo.

Si confronti il seguente contenuto con quello mostrato nell'***Esempio 9.2***.

```
$ cat testo_da_ordinare.txt
1a riga
a
2a riga
A
3a riga
AA
4a riga
aa
5a riga
ba
6a riga
b a
7a riga
b|
8a riga
baz
9a riga
B4
10a riga
casa
11a riga
à123
12a riga
treno
13a riga
hotel
14a riga
HOTEL
15a riga
HOTel
HO tel
16a riga
HO  tel
17a riga
HO  Tel
18a riga
Z
19a riga
Zz
```

Esempio 9.1: *Comando* `cat` *su file da ordinare*

```
$ sort testo_da_ordinare.txt
10a riga
11a riga
12a riga
13a riga
14a riga
15a riga
16a riga
17a riga
18a riga
19a riga
1a riga
2a riga
3a riga
4a riga
5a riga
6a riga
```

```
18 7a riga
19 8a riga
20 9a riga
21 a
22 A
23 à123
24 aa
25 AA
26 b|
27 B4
28 b a
29 ba
30 baz
31 casa
32 hotel
33 HO  tel
34 HO tel
35 HO  Tel
36 HOTel
37 HOTEL
38 treno
39 Z
40 Zz
```

Esempio 9.2: *Comando* `sort` *su file senza opzioni*

in cui l'ordinamento è stato lessicografico tenendo conto delle variabili `locale` impostate[1]. Per ottenere un tradizionale ordinamento *native byte order*[2], si può passare la variabile `LC_ALL` con un valore differente:

```
1 $ LC_ALL=C sort testo_da_ordinare.txt | tail -n21
2 9a riga
3 A
4 AA
5 B4
6 HO  Tel
7 HO  tel
8 HO tel
9 HOTEL
10 HOTel
11 Z
12 Zz
13 a
14 aa
15 b a
16 ba
17 baz
18 b|
19 casa
20 hotel
21 treno
22 à123
```

Esempio 9.3: *Comando* `sort` *con ridefinizione della variabile* `LC_ALL`

In ***Tabella 9.1*** le opzioni principali.
Ad esempio è possibile effettuare un ordinamento numerico inverso:

[1] argomento trattato in 14.11 a pag. 179
[2] ordine ASCII, V a pag. 399

OPZIONI		DESCRIZIONE
BREVE	ESTESA	
`-d`	`--dictionary-order`	considera solo gli spazi e i caratteri alfanumerici
`-f`	`--ignore-case`	ordina i caratteri minuscoli come se fossero maiuscoli
`-n`	`--numeric-sort`	ordina secondo il calore numerico delle stringhe
`-k=K`	`--key=K`	ordina in base alla chiave `K`. La definizione della chiave può essere complessa, ma può semplicemente indicare la posizione del campo o la posizione del o dei caratteri
`-r`	`--reverse`	inverte l'ordine prima di mostrare l'elenco
`-t=S`	`--field-separator=S`	imposta il separatore `S` invece di usare la transizione da *non spazio* a *spazio*
`-u`	`--uniquq`	non riporta le linee duplicate

Tabella 9.1: *Opzioni principali del comando* `sort`

```
1 $ sort -nr testo_da_ordinare.txt | grep -E '^[0-9]' | head -n 12
2 19a riga
3 18a riga
4 17a riga
5 16a riga
6 15a riga
7 14a riga
8 13a riga
9 12a riga
10 11a riga
11 10a riga
12 9a riga
13 8a riga
```

Esempio 9.4: *Comando* `sort -n`

oppure, se il file è scritto a colonne, è possibile ordinare rispettivamente per una in particolare o per sottostringa di una colonna:

```
1 $ cat testo_da_ordinare2.txt
2 a,209,casa
3 b,115,mare
4 $ sort -n -t "," -k 2 testo_da_ordinare2.txt
5 b,115,mare
6 a,209,casa
7 $ sort -n -t "," -k 2.2,2.3 testo_da_ordinare2.txt
8 a,209,casa
9 b,115,mare
```

Esempio 9.5: *Comando* `sort -n`

in cui con `-k 2` si è impostata la chiave di confronto dal secondo campo[3], mentre con `-k 2.2,2.3` si è definita come chiave di confronto il secondo e terzo carattere del secondo campo.

9.3 uniq

Il comando `uniq` riporta o omette le linee che *adiacenti* che si ripetono. In ***Tabella 9.2*** le opzioni principali.

OPZIONI		DESCRIZIONE
BREVE	ESTESA	
`-c`	`--count`	aggiunge il numero delle occorrenze alla linea alfanumerici
`-d`	`--repeated`	mostra solo le linee che si ripetono
`-i`	`--ignore-case`	ignora la differenza tra maiuscole e minuscole
`-f=N`	`--skip-fields=N`	evita di comparare i primi `N` campi (ciascun campo è separato dal successivo da caratteri di *spazio* o *tab*)

Tabella 9.2: *Opzioni principali del comando* `uniq`

Poiché `uniq` non rileva linee duplicate se non sono adiacenti, tale comando è solitamente utilizzato nelle pipeline con `sort`. Il comando `sort` attivato con l'opzione `-u` già effettua la rimozione dei duplicati[4], ma non può:

- produce autonomamente solo le linee duplicate;
- conteggiare le linee duplicate.

Nell'esempio una tipica pipeline con `sort`:

```
1  $ cat testo_uniq.txt
2  a
3  b
4  a
5  b
6  c
7  a
8  $ sort testo_uniq.txt | uniq -cd | sort -n
9        2 b
10       3 a
```

Esempio 9.6: *Comando* `uniq` *in pipeline con* `sort`

9.4 cut

Il comando `cut` rimuove sezioni per ogni linea di file, quindi mostra solo parti selezionate indicate tramite opzioni. che *adiacenti* che si ripetono. In ***Tabella 9.3*** le opzioni principali.

[3] quindi dal secondo campo fino alla fine della linea

[4] inoltre è più flessibile poiché non pone il vincolo di considerare duplicate solo linee adiacenti

OPZIONI		DESCRIZIONE
BREVE	ESTESA	
`-b=L`	`--bytes=L`	mostra solo i byte in `L`
`-c=L`	`--characters=L`	mostra solo i caratteri in `L`
`-d=D`	`--delimiter=D`	imposta la stringa `D` come delimitatore tra campi al posto del *tab*
`-f=L`	`--fields=L`	mostra solo i campi in `L`
`-i`	`--ignore-case`	ignora la differenza tra maiuscole e minuscole
`-f=N`	`--skip-fields=N`	evita di comparare i primi `N` campi (ciascun campo è separato dal successivo da caratteri di *spazio* o *tab*)

Tabella 9.3: *Opzioni principali del comando* `cut`

Nella tabella si è utilizzato ripetutamente il carattere `L` per indicare una lista, la quale può essere espressa tramite un range o più range separati da una virgola. Ciascun range è del tipo:

`N`
: per indicare il byte/carattere/campo, iniziando da 1;

`N-`
: dall'*ennesimo* byte/carattere/campo fino al termine della linea;

`N-M`
: dal byte/carattere/campo *n* fino all'omologo *m*;

`-M`
: fino al byte/carattere/campo numero *m*.

L'esempio seguente:

```
1 $ cut -c -3
2 12
3 12
4 1234
5 123
```

Esempio 9.7: *Comando* `cut` *su* `stdin`

estrae i primi tre caratteri da ogni linea dello `stdin`.
Quest'altro esempio prende il file di input nell'***Esempio 7.5***

```
1 $ cut -b 2-5,8,20- testo.txt
2 HIS NT UPPER CASE LINE IN THIS FILE.
3 his nt lower case line in this file.
4 his nts First Character Of The Word With Upper Case.
5
6 wo les line is empty.
7 nd tst line.
```

Esempio 9.8: *Comando* `cut` *su byte*

e mostra i byte dal secondo al quinto, l'ottavo e dal ventesimo fino alla fine della linea. Nell'esempio seguente si estraggono i campi 1 e 6 del file `/etc/passwd` degli utenti che utilizzano la `bash`:

```
1 $ grep bash /etc/passwd | cut -f 1,6 -d ':'
2 root:/root
3 ms:/home/ms
```

Esempio 9.9: *Comando `cut` su `/etc/passwd` precedentemente filtrato tramite `grep`*

che riporta le username degli utenti e la relativa *home directory*.

Le capacità di `cut` sono alquanto ridotte, ad esempio non permette:

- di estrarre solo parte dei dati delle sezioni;
- non è flessibile nell'indicare il separatore[5];

pertanto si impiega solo nei casi più semplici, e si ricorre ad alternative più potenti quali lo strumento `awk`[6] o il comando `sed`[7] per ottenere maggior espressività.

[5] ad esempio non permette di specificare una sequenza dinamica di spazi, quindi non è adatto a manipolare l'output di `ls`

[6] trattato in **??** a pag. **??**

[7] trattato in **??** a pag. **??**

9.5 Concetti chiave

data management: si intende la gestione dei dati applicando gli strumenti di filtraggio

9.6 Comandi introdotti nel capitolo

`cut` ⇒ mostra solo alcune sezioni per ciascuna linea dei file di input
`sort` ⇒ ordina linee di file di testo
`uniq` ⇒ riporta o omette linee ripetute (adiacenti)

Capitolo 10

Archiviazione e compressione

QUESTO capitolo ha come argomento centrale la gestione di file e di directory (talvolta di grosse dimensioni) per una iniziale trattazione del tema della storicizzazione, affrontato nel capitolo **??** a pag.**??**.
Di seguito saranno focalizzati due processi:

archiviazione: produce di un file contenitore in cui sono immagazzinati più file (e directory) e i relativi metadati, i quali permettono l'estrazione successiva per riprodurre parzialmente o totalmente la struttura originale;

compressione[1]**:** processo algoritmico che trasforma file riducendo le dimensioni[2] permette di ridurre le dimensioni del file prodotto senza perdita di informazione, in maniera che il successivo processo di decompressione permetterà di riprodurre l'originale

Per i principali strumenti di archiviazione e compressione è utile riferirsi alla ***Tabella 10.1***. Tali strumenti saranno esaminati nelle prossime sezioni del capitolo.

SOFTWARE	USO
`bunzip2`	decompressione di file `.bz2`
`bzip2`	compressione di file in `.bz2`
`gunzip`	decompressione di file `.gz`
`gzip`	compressione di file in `.gz`
`split`	split di file in pezzi fino ad una grandezza massima
`tar`	gestione archivi (creazione ed estrazione)
`unzip`	decompressione di file *Windows-style* `.zip`
`zip`	archiviazione e compressione software simile a *PKZIP*

Tabella 10.1: *Principali software di archiviazione e compressione*

10.1 Compressione file

Una generica procedura di compressione di file prevede i seguenti passaggi:

[2] si intende almeno per una classe la cui numerosità significativa

1. processamento in memoria del file originale per la compressione;
2. creazione file omonimo con aggiunta del suffisso `.gz`;
3. rimozione dell'originale.

Questa procedura è eseguita anche se il prodotto finale sarà più grande del file originale. È possibile utilizzare diversi strumenti per la compressione e decompressione di singoli file, di seguito saranno presentati i principali.
Si segnalano anche i tool `xz`[3] e `zip`[4] discretamente utilizzati i quali non saranno argomentati[5].

10.1.1 `gzip` e `gunzip`

L'utility `gzip`[6] utilizza l'algoritmo di compressione descritto nell'*RFC 1951*[7]. In generale, nella compressione è creato un file omonimo con il suffisso `.gz`, nella decompressione al nome del file è tolto tale suffisso.
Le principali opzioni sono riportate in ***Tabella 10.2***.

OPZIONI	SIGNIFICATO
`-c`	scrive il file compresso/decompresso allo `stdout` invece di rimpiazzare il file originale (il quale rimane immutato)
`-d`	decomprime il file (alternativa a `gunzip`)
`-k`	non cancella l'originale (*keep*)
`-l`	mostra la lista e informazioni su file compressi quali nome originale dei file, grandezza originale, ...
`-r`	ricorsivo nelle subdirectory
`-v`	attiva la modalità *verbose*
`-1 ...-9`	specifica il fattore di compressione: `-1` o `--fast` processa rapidamente ma comprime meno, `-9` o `--best` comprime al massimo ma è più lento; il default è `-6`

Tabella 10.2: *Opzioni principali di* `gzip`

Un esempio di compressione:

```
1 $ gzip testo.txt
```

Esempio 10.1: *Compressione con* `gzip`

[3] per decomprimere `unxz`
[4] per decomprimere `unzip`
[5] per dettagli sono consultabili i rispettivi manuali `man xz` e `man zip`
[6] abbreviazione di ***GNU zip***
[7] v. [Deu96] - In sintesi è un successore dell'algoritmo **LZW** (***L**empel-**Z**iv-**W**elch*), alla base del software `compress`, strumento standard per le versioni proprietarie di Unix, il quale però era coperto da brevetto (scaduto il 20 giugno 2003). Il software `gzip` utilizza il metodo *DEFLATE* descritto nell'RFC che si basa su un precursore non brevettato di *LZW* denominato **LZ77**, anch'esso basato sullo schema di *codifica Huffman*, priva di brevetto. Inoltre `gzip` può decomprimere i file prodotti da `compress` poiché il brevetto copriva esclusivamente la compressione (tali file terminano con il suffisso `.Z`)

che produrrà il file `testo.txt.gz` eliminando l'originale. L'estrazione avverrà con:

```
1 $ gzip -d testo.txt.gz
```

Esempio 10.2: *Decompressione con* `gzip`

che riprodurrà il file originale `testo.txt` eliminando il compresso. Equivalentemente all'esempio precedente, è possibile utilizzare `gunzip`:

```
1 $ gunzip testo.txt.gz
```

Esempio 10.3: *Decompressione con* `gunzip`

ed anche qui si riprodurrà il file originale `testo.txt` eliminando il compresso. Aggiungendo l'opzione `-v` si attiverà la modalità *verbose* ottenendo il fattore di compressione:

```
1 $ gzip -v testo.txt
2 testo.txt:   28.3% -- replaced with testo.txt.gz
```

Esempio 10.4: *Compressione con* `gzip -v`

e ovviamente anche per la decompressione:

```
1 $ gunzip -v testo.txt.gz
2 testo.txt.gz:  28.3% -- replaced with testo.txt
```

Esempio 10.5: *Decompressione con* `gunzip -v`

Può essere utile decomprimere inviando l'output non su file ma su `stdout` aggiungendo l'opzione `-c`[8]. Equivalentemente, per la decompressione su `stdout`, si può utilizzare `zcat`.
In sintesi le seguenti linee sono identiche:

```
1 $ gunzip -c testo.txt.gz
2 $ gzip -cd testo.txt.gz
3 $ zcat testo.txt.gz
```

Esempio 10.6: *Decompressione su* `stdout`

ma, probabilmente, per questioni mnemoniche, si uilizza più spesso l'ultima.

10.1.2 `bzip2` e `bunzip2`

Il software `bzip2` utilizza un algoritmo di compressione differente dal precedente[9] ottenendo generalmente fattori di compressione migliori, ma utilizza opzioni di compressione analoghe.
In particolare il file prodotto è creato anch'esso omonimo con il suffisso `.bz2`, nella decompressione al nome del file è tolto tale suffisso.
Le principali opzioni sono riportate in ***Tabella 10.3***.
Un esempio di compressione:

```
1 $ bzip2 testo.txt
```

Esempio 10.7: *Compressione con* `bzip2`

[8] poco utile, ma può anche essere applicato per inviare su `stdout` l'output della compressione
[9] utilizza la trasformata di *Burrows-Wheeler* successivamente processata tramite l'*Huffman coding*

OPZIONI	SIGNIFICATO
`-c`	scrive il file compresso/decompresso allo `stdout` invece di rimpiazzare il file originale (il quale rimane immutato)
`-d`	decomprime il file (alternativa a `bunzip2`)
`-k`	non cancella l'originale (*keep*)
`-v`	attiva la modalità *verbose*

Tabella 10.3: *Opzioni principali di* `bzip2`

che produrrà il file `testo.txt.bz2` eliminando l'originale. L'estrazione avverrà con:

```
1 $ bzip2 -d testo.txt.bz2
```

Esempio 10.8: *Decompressione con* `bzip2`

che riprodurrà il file originale `testo.txt` eliminando il compresso. Equivalentemente all'esempio precedente, è possibile utilizzare `bunzip2`:

```
1 $ bunzip2 testo.txt.bz2
```

Esempio 10.9: *Decompressione con* `bunzip2`

ed anche qui si riprodurrà il file originale `testo.txt` eliminando il compresso. Aggiungendo l'opzione `-v` si attiverà la modalità *verbose* ottenendo il fattore di compressione:

```
1 $ bzip2 -v testo.txt
2 testo.txt: 216.225:1,  0.037 bits/byte, 99.54% saved, 27893 in, 129 out.
```

Esempio 10.10: *Compressione con* `bzip2 -v`

e ovviamente anche per la decompressione:

```
1 $ bunzip2 -v testo.txt.bz2
2 testo.txt.bz2: done
```

Esempio 10.11: *Decompressione con* `bunzip2 -v`

Può essere utile decomprimere inviando l'output non su file ma su `stdout` aggiungendo l'opzione `-c`[10]. Equivalentemente, per la decompressione su `stdout`, si può utilizzare `bzcat`.
In sintesi le seguenti linee sono identiche:

```
1 $ bunzip2 -c testo.txt.bz2
2 $ bzip2 -cd testo.txt.bz2
3 $ bzcat testo.txt.bz2
```

Esempio 10.12: *Decompressione su* `stdout`

ma, probabilmente, per questioni mnemoniche, si uilizza più spesso l'ultima.

[10] poco utile, ma può anche essere applicato per inviare su `stdout` l'output della compressione

10.2 Archiviare file

Il comando `tar` il cui nome deriva da ***t**ape **ar**chive* è un software che produce file di archivio non compressi denominati *tarball*, ideati principalmente per il backup. Già dal nome si comprende che è uno strumento storico, ma la sua utilità è immutata considerando che è lo strumento utilizzato comunemente per rendere disponibile i sorgenti dei software.
Gli archivi `tar` possono contenere file e intere directory, e la sua estrema versatilità permette di spingersi alla creazione di archivi via rete. Le funzioni principali dello strumento sono due:

creazione di archivi raggruppando file;

estrazione da archivi recuperando i file.

Gli archivi `tar` sono normalmente non compressi a meno che non si utilizzino strumenti esterni di compressione, ma taluni[11] possono anche essere attivati tramite opzioni.
I tipici suffissi utilizzati per l'archivio prodotto sono:

`.tar`
: per la semplice costruzione dello stesso;

`.tar.gz` o `.tgz`
: per indicare la successiva compressione tramite `gzip`;

`.tar.bz2` o `.tbz2`
: per indicare la successiva compressione tramite `bzip2`.

La sintassi generica del comando `tar` è la seguente:

```
1 $ tar <opzioni> [files]
```

Esempio 10.13: *Sintassi generica dello strumento di archiviazione* `tar`

con le opzioni[12] principali riportate in ***Tabella 10.4***.

OPZIONI	SIGNIFICATO
`-c`	crea il nuovo archivio
`-f`	il nome archivio da creare/da leggere
`-j`	comprime o decomprime utilizzando `bzip2`
`-M`	gestisce gli archivi multivolume
`-r`	aggiunge file all'archivio
`-t`	mostra l'*indice* del contenuto
`-u`	sostituisce i file nell'archivio con le nuove versioni sottoposte (se il file non c'è, lo creerà)
`-v`	attiva la modalità *verbose*
`-x`	estrae file e directory dall'archivio
`-z`	comprime o decomprime l'archivio usando `gzip`

Tabella 10.4: *Opzioni principali di* `tar`

Iniziamo dal seguente esempio per la creazione di un archivio:

[11] `gzip` e `bzip2`

[12] per il comando `tar` eccezionalmente le opzioni possono essere espresse senza il prefisso `-`, ma la pratica è sconsigliata

```
1 $ tar -cvf ~/dati.tar dati*
2 dati1
3 dati2
4 dati3
```

Esempio 10.14: *Creazione di un archivio di file*

il quale creerà nella home directory il file di archivio `dati.tar` contenente i file che matchano il globbing `dati*`. Ovviamente il contenuto da archiviare indicato potrà essere diversificato, quindi più argomenti[13] e più tipologie (file, directory, ...).
Le opzioni utilizzate sono `-c` per la creazione dell'archivio, `-v` per attivare la modalità *verbose*[14], `-f /dati.tar` indica il pathname dell'archivio prodotto.
Lo strumento può archiviare directory, come nel seguente esempio:

```
1 $ tar -cvf /tmp/home_ms.tar /home/ms
2 tar: Rimozione di "/" iniziale dai nomi dei membri
3 /home/ms/
4 ...
```

Esempio 10.15: *Creazione di un archivio di directory*

Per questo passaggio ci sono da notare:

- l'archivio è da produrre esternamente alla sorgente, altrimenti la sorgente conterrà l'archivio vuoto che si dovrà ancora formare, comunque materiale errato in quanto non sarebbe terminata la fase di creazione;
- la prima linea riportata dall'output ricorda che non è una buona pratica mettere un path assoluto nella creazione di un archivio per indicare ciò che è da archiviare, altrimenti l'estrazione ricreerebbe l'albero potenzialmente sovrascrivendolo (in altre parole, il carattere \è stato rimosso). Per evitare il warning sono da utilizzare path relativi.

Riprendendo l'***Esempio 10.14***, è possibile visualizzarne l'*indice* del contenuto tramite l'opzione `-t`:

```
1 $ tar -tf ~/dati.tar
2 dati1
3 dati2
4 dati3
```

Esempio 10.16: *Visualizzazione indice del contenuto*

e, per ottenere più informazioni, è possibile attivare la modalità *verbose* aggiungendo l'opzione `-v`:

```
1 $ tar -tvf ~/dati.tar
2 -rw-rw-r-- ms/ms             0 2023-11-24 11:26 dati1
3 -rw-rw-r-- ms/ms             0 2023-11-24 11:26 dati2
4 -rw-rw-r-- ms/ms             0 2023-11-24 11:26 dati3
```

Esempio 10.17: *Visualizzazione indice del contenuto con* `-v`

Per l'estrazione del contenuto dell'archivio si utilizzerà l'opzione `-x`:

[13] come già è stato effettuato attraverso il *globbing*

[14] comunicare l'attività contemporaneamente allo svolgimento, quindi a video riporta il nome dei file che aggiunge all'archivio

```
1 $ tar -xvf ~/dati.tar
2 dati1
3 dati2
4 dati3
```

Esempio 10.18: *Estrazione contenuto*

che ricreerà i file e la struttura delle directory come indicato nello stesso archivio. Se i file già esisteranno, saranno sovrascritti, per cui è consigliatissimo prima verificare il contenuto dell'archivio tramite l'opzione `-t`.

È possibile estrarre solo una parte del contenuto dall'archivio indicando il pathname (o i path) interessati:

```
1 $ tar -xvf ~/dati.tar dati2
2 dati2
```

Esempio 10.19: *Estrazione contenuto*

Riprendendo l'esempio di creazione di un archivio, successivamente lo si potrà comprimere con il software che si riterrà opportuno. Se la scelta ricadrà su `gzip`, si potrà seguire la procedura in due passaggi come di seguito:

```
1 $ tar -cvf dati.tar dati/
2 dati/
3 dati/dati2
4 dati/dati1
5 dati/dati3
6 $ gzip dati.tar
```

Esempio 10.20: *Creazione di un file compresso con `gzip` da un archivio di file*

che produrrà `dati.tar.gz`, oppure si può optare per un unico passaggio aggiungendo l'opzione `-z`:

```
1 $ tar -zcvf dati.tar.gz dati/
2 dati/
3 dati/dati2
4 dati/dati1
5 dati/dati3
```

Esempio 10.21: *Creazione di un archivio di file e compressione con `gzip`*

e per decomprimere anche qui in due passaggi applicando `gunzip` e poi `tar -x`:

```
1 $ gunzip dati.tar.gz
2 $ tar -zxvf dati.tar
3 dati/
4 dati/dati2
5 dati/dati1
6 dati/dati3
```

Esempio 10.22: *Decompressione file di archivio `gzip` e estrazione del contenuto*

oppure in un unico passaggio:

```
1 $ tar -zxvf dati.tar.gz
2 dati/
3 dati/dati2
4 dati/dati1
5 dati/dati3
```

Esempio 10.23: *Decompressione ed estrazione file di archivio* `gzip`

Analogamente a quanto mostrato dall'***Esempio 10.20*** all'***Esempio 10.23***, è possibile utilizzare lo strumento di compressione `bzip2`, quindi si potrà seguire la procedura in due passaggi come di seguito:

```
1 $ tar -cvf dati.tar dati/
2 dati/
3 dati/dati2
4 dati/dati1
5 dati/dati3
6 $ bzip2 dati.tar
```

Esempio 10.24: *Creazione di un file compresso con* `bzip2` *da un archivio di file*

che produrrà `dati.tar.bz2`, oppure si può optare per un unico passaggio aggiungendo l'opzione `-j`:

```
1 $ tar -jcvf dati.tar.bz2 dati/
2 dati/
3 dati/dati2
4 dati/dati1
5 dati/dati3
```

Esempio 10.25: *Creazione di un archivio di file e compressione con* `bzip2`

e per decomprimere anche qui in due passaggi applicando `bunzip2` e poi `tar -x`:

```
1 $ bunzip2 dati.tar.bz2
2 $ tar -xvf dati.tar
3 dati/
4 dati/dati2
5 dati/dati1
6 dati/dati3
```

Esempio 10.26: *Decompressione file di archivio* `bzip2` *e estrazione del contenuto*

oppure in un unico passaggio:

```
1 $ tar -jxvf dati.tar.bz2
2 dati/
3 dati/dati2
4 dati/dati1
5 dati/dati3
```

Esempio 10.27: *Decompressione ed estrazione file di archivio* `bzip2`

10.3 Spezzettare gli archivi

Può capitare di dover gestire file di grosse dimensioni e, ad esempio, si vogliano trasferire come copia di backup[15]. Il comando `split` è utilizzato per spezzare file di grandi dimensioni (spesso archivi) in chunk più piccoli, poi riassemblati utilizzando il comando `cat`. La sintassi generica del comando `split` è la seguente:

```
1 $ split <opzioni> file [prefix]
```

Esempio 10.28: *Sintassi generica del comando* `split`

in cui le opzioni principali sono riportate in ***Tabella 10.5***, il prefisso è il seme che si vuole assegnare ai *chunk* prodotti, solitamente omonimo al file originale. Per la dimensione del

OPZIONI	SIGNIFICATO
`-a N`	stabilisce come lunghezza del suffisso `N` (default 2)
`-b S`	dimensione `S` del singolo chunk
`-d`	utilizza un suffisso numerico (il default è alfabetico)
`-n N`	genera esattamente N chunk
`-x`	usa un suffisso esadecimale

Tabella 10.5: *Principali opzioni del comando* `split`

chunk, dopo il numero può essere utilizzata l'unità di misura[16]. Ad esempio, si abbia il file `dati.tgz` da `12G`, e lo si voglia suddividere in chunk da 100M:

```
1  $ split -b 100M dati.tgz
2  $ ls
3  dati.tgz  xai  xar  xba  xbj  xbs  xcb  xck  xct  xdc  xdl  xdu  xed
4  xaa       xaj  xas  xbb  xbk  xbt  xcc  xcl  xcu  xdd  xdm  xdv  xee
5  xab       xak  xat  xbc  xbl  xbu  xcd  xcm  xcv  xde  xdn  xdw  xef
6  xac       xal  xau  xbd  xbm  xbv  xce  xcn  xcw  xdf  xdo  xdx  xeg
7  xad       xam  xav  xbe  xbn  xbw  xcf  xco  xcx  xdg  xdp  xdy  xeh
8  xae       xan  xaw  xbf  xbo  xbx  xcg  xcp  xcy  xdh  xdq  xdz  xei
9  xaf       xao  xax  xbg  xbp  xby  xch  xcq  xcz  xdi  xdr  xea
10 xag       xap  xay  xbh  xbq  xbz  xci  xcr  xda  xdj  xds  xeb
11 xah       xaq  xaz  xbi  xbr  xca  xcj  xcs  xdb  xdk  xdt  xec
```

Esempio 10.29: `split` *da* $100M$

In alternativa è possibile definire con l'opzione `-n` il numero dei chunk:

```
1 $ split -n 3 dati.tgz
2 $ ls
3 dati.tgz  xaa  xab  xac
```

Esempio 10.30: `split` *con numero di chunk definito*

con chunk di dimensione $\sim 3.7G$.
Sicuramente è più utile l'opzione per definire la grandezza dei chunk:

[15] argomento trattato al capitolo **??** a pag. **??**

[16] il suffisso `K`, `M`, `G`, `T`, `P`, `E`, `Z`, `Y` per i multipli di 2^10, `KB`, `MB`, `GB`, ... per i multipli di 10^3 come indicato in ***Tabella 24.8*** a pag. 405

```
$ split -b 100M dati.tgz
$ ls
dati.tgz  xai  xar  xba  xbj  xbs  xcb  xck  xct  xdc  xdl  xdu  xed
xaa       xaj  xas  xbb  xbk  xbt  xcc  xcl  xcu  xdd  xdm  xdv  xee
xab       xak  xat  xbc  xbl  xbu  xcd  xcm  xcv  xde  xdn  xdw  xef
xac       xal  xau  xbd  xbm  xbv  xce  xcn  xcw  xdf  xdo  xdx  xeg
xad       xam  xav  xbe  xbn  xbw  xcf  xco  xcx  xdg  xdp  xdy  xeh
xae       xan  xaw  xbf  xbo  xbx  xcg  xcp  xcy  xdh  xdq  xdz  xei
xaf       xao  xax  xbg  xbp  xby  xch  xcq  xcz  xdi  xdr  xea
xag       xap  xay  xbh  xbq  xbz  xci  xcr  xda  xdj  xds  xeb
xah       xaq  xaz  xbi  xbr  xca  xcj  xcs  xdb  xdk  xdt  xec
```

Esempio 10.31: *`split` da* $100M$

con `dati.tgz` il file originale, ciascun `x??` da $100M$ ad esclusione dell'ultimo di `da 64M`.

Potremmo preferire che i file abbiano il nome che identifichi l'origine, quindi digiteremo il seguente comando:

```
$ split -b 100M dati.tgz dati.tgz.
$ ls
dati.tgz     dati.tgz.aw  dati.tgz.bt  dati.tgz.cq  dati.tgz.dn
dati.tgz.aa  dati.tgz.ax  dati.tgz.bu  dati.tgz.cr  dati.tgz.do
dati.tgz.ab  dati.tgz.ay  dati.tgz.bv  dati.tgz.cs  dati.tgz.dp
dati.tgz.ac  dati.tgz.az  dati.tgz.bw  dati.tgz.ct  dati.tgz.dq
dati.tgz.ad  dati.tgz.ba  dati.tgz.bx  dati.tgz.cu  dati.tgz.dr
dati.tgz.ae  dati.tgz.bb  dati.tgz.by  dati.tgz.cv  dati.tgz.ds
dati.tgz.af  dati.tgz.bc  dati.tgz.bz  dati.tgz.cw  dati.tgz.dt
dati.tgz.ag  dati.tgz.bd  dati.tgz.ca  dati.tgz.cx  dati.tgz.du
dati.tgz.ah  dati.tgz.be  dati.tgz.cb  dati.tgz.cy  dati.tgz.dv
dati.tgz.ai  dati.tgz.bf  dati.tgz.cc  dati.tgz.cz  dati.tgz.dw
dati.tgz.aj  dati.tgz.bg  dati.tgz.cd  dati.tgz.da  dati.tgz.dx
dati.tgz.ak  dati.tgz.bh  dati.tgz.ce  dati.tgz.db  dati.tgz.dy
dati.tgz.al  dati.tgz.bi  dati.tgz.cf  dati.tgz.dc  dati.tgz.dz
dati.tgz.am  dati.tgz.bj  dati.tgz.cg  dati.tgz.dd  dati.tgz.ea
dati.tgz.an  dati.tgz.bk  dati.tgz.ch  dati.tgz.de  dati.tgz.eb
dati.tgz.ao  dati.tgz.bl  dati.tgz.ci  dati.tgz.df  dati.tgz.ec
dati.tgz.ap  dati.tgz.bm  dati.tgz.cj  dati.tgz.dg  dati.tgz.ed
dati.tgz.aq  dati.tgz.bn  dati.tgz.ck  dati.tgz.dh  dati.tgz.ee
dati.tgz.ar  dati.tgz.bo  dati.tgz.cl  dati.tgz.di  dati.tgz.ef
dati.tgz.as  dati.tgz.bp  dati.tgz.cm  dati.tgz.dj  dati.tgz.eg
dati.tgz.at  dati.tgz.bq  dati.tgz.cn  dati.tgz.dk  dati.tgz.eh
dati.tgz.au  dati.tgz.br  dati.tgz.co  dati.tgz.dl  dati.tgz.ei
dati.tgz.av  dati.tgz.bs  dati.tgz.cp  dati.tgz.dm
```

Esempio 10.32: *`split` da* $100M$ *con prefisso assegnato*

con il prefisso `dati.tgz.`.

Un'opzione più elaborata probabilmente preferibile è la seguente:

```
$ split -a3 -x -b 100M dati.tgz dati.tgz.
$ ls
dati.tgz      dati.tgz.016  dati.tgz.02d  dati.tgz.044  dati.tgz.05b
dati.tgz.000  dati.tgz.017  dati.tgz.02e  dati.tgz.045  dati.tgz.05c
dati.tgz.001  dati.tgz.018  dati.tgz.02f  dati.tgz.046  dati.tgz.05d
dati.tgz.002  dati.tgz.019  dati.tgz.030  dati.tgz.047  dati.tgz.05e
dati.tgz.003  dati.tgz.01a  dati.tgz.031  dati.tgz.048  dati.tgz.05f
dati.tgz.004  dati.tgz.01b  dati.tgz.032  dati.tgz.049  dati.tgz.060
dati.tgz.005  dati.tgz.01c  dati.tgz.033  dati.tgz.04a  dati.tgz.061
dati.tgz.006  dati.tgz.01d  dati.tgz.034  dati.tgz.04b  dati.tgz.062
dati.tgz.007  dati.tgz.01e  dati.tgz.035  dati.tgz.04c  dati.tgz.063
```

```
dati.tgz.008  dati.tgz.01f  dati.tgz.036  dati.tgz.04d  dati.tgz.064
dati.tgz.009  dati.tgz.020  dati.tgz.037  dati.tgz.04e  dati.tgz.065
dati.tgz.00a  dati.tgz.021  dati.tgz.038  dati.tgz.04f  dati.tgz.066
dati.tgz.00b  dati.tgz.022  dati.tgz.039  dati.tgz.050  dati.tgz.067
dati.tgz.00c  dati.tgz.023  dati.tgz.03a  dati.tgz.051  dati.tgz.068
dati.tgz.00d  dati.tgz.024  dati.tgz.03b  dati.tgz.052  dati.tgz.069
dati.tgz.00e  dati.tgz.025  dati.tgz.03c  dati.tgz.053  dati.tgz.06a
dati.tgz.00f  dati.tgz.026  dati.tgz.03d  dati.tgz.054  dati.tgz.06b
dati.tgz.010  dati.tgz.027  dati.tgz.03e  dati.tgz.055  dati.tgz.06c
dati.tgz.011  dati.tgz.028  dati.tgz.03f  dati.tgz.056  dati.tgz.06d
dati.tgz.012  dati.tgz.029  dati.tgz.040  dati.tgz.057  dati.tgz.06e
dati.tgz.013  dati.tgz.02a  dati.tgz.041  dati.tgz.058  dati.tgz.06f
dati.tgz.014  dati.tgz.02b  dati.tgz.042  dati.tgz.059  dati.tgz.070
dati.tgz.015  dati.tgz.02c  dati.tgz.043  dati.tgz.05a
```

Esempio 10.33: *`split` da* $100M$ *con prefisso assegnato e opzioni* `-dx`

con il prefisso `dati.tgz.` e le opzioni `-a3` e `-x` per definire il suffisso rispettivamente di lunghezza 3 e esadecimale.

Per ricreare il file originale dai chunk, è sufficiente eseguire un `cat`:

```
$ cat dati.tgz.* > dati_copia.tgz
$ rm dati.tgz.*
```

Esempio 10.34: *`cat` dei chunk*

in cui:

- la prima linea ricostruisce il file originale;
- la seconda linea per pulizia dei chunk.

Per verificare che la ricostruzione sia avvenuta correttamente[17], è utile calcolare i *digest*[18] del sorgente e dell'assemblato come di seguito:

```
$ shasum dati*.tgz
c6dddae17fe2a1921bccc51ca3da1798fb425ff8  dati_copia.tgz
c6dddae17fe2a1921bccc51ca3da1798fb425ff8  dati.tgz
```

Esempio 10.35: *Calcolo del digest SHA dei due file*

in cui si è utilizzato il software `shasum` per verificare la corrispondenza dell'*SHA checksum*.

[17] lo strumento svolge egregiamente il task assegnato, ma l'errore di digitazione è sempre in agguato ☺

[18] l'argomento sarà approfondito al capitolo **??** a pag. **??**

10.4 Concetti chiave

archiviazione: immagazzinamento di più file in un file nuovo cosicché dallo stesso successivamente potrà essere riprodotto la struttura originale

chunk: pezzi solitamente equivalenti che riassemblati in ordine formano il file originale

compressione: qui *lossless*, si intende un processo algoritmico che trasforma il file originale in uno più piccolo senza perdita di informazioni, quindi il processo inverso di decompressione servirà per tornare all'originale

digest: genericamente si intente calcolo algoritmico per un'estrema sintesi di tipo numerico con alta sensibilità alle condizioni iniziali

decompressione: processo inverso alla compressione, partendo da un file compresso permette di tornare all'originale

SHA checksum: calcolo del checksum utilizzando l'algoritmo *SHA1*

verbose: se la modalità è attivata, lo strumento aggiungerà informazioni (ad esempio, comunicherà dettagli dell'attività contemporaneamente allo svolgimento)

10.5 Comandi introdotti nel capitolo

`bzcat` ⇒ decompressione di file compressi con `bzip2` su `stdout`

`bunzip` ⇒ software di decompressione, equivale a `bzip2 -d`

`bzip2` ⇒ software di compressione a blocchi che generalmente ottiene fattori di compressione migliore di quello ottenuti tramite `gzip`

`gunzip` ⇒ software di decompressione, equivale a `gzip -d`

`gzip` ⇒ software di compressione ***G****NU* ***z****ip*$basato$ su *Lempel-Ziv coding (LZ77)*

`shasum` ⇒ calcolo dell'*SHA checksum* degli argomenti

`split` ⇒ suddivide un file in chunk

`tar` ⇒ software di utilità per archiviare

`zcat` ⇒ decompressione di file compressi con `gzip` su `stdout`

Capitolo 11

Variabili di shell e environment

COME la maggior parte delle shell, *bash* ha caratteristiche comuni con molti linguaggi di programazione.
Una delle più semplici è l'uso delle *variabili* per il riuso in momenti successivi, talune sono utilizzate per la definizione dell'*environment*.

11.1 Variabili di shell

È possibile memorizzare stringhe o numeri con un comando del tipo `variable=value`, impostando la variabile `variable` alla stringa `value`. Nelle assegnazioni è importante non inserire spazi né prima né dopo il segno =.
Successivamente all'assegnazione, è possibile recuperare il valore della variabile utilizzando il simbolo `$` con il nome della variabile, come nel seguente esempio:

```
1 $ welcome=Hello
2 $ echo $welcome
3 Hello
4 $ welcome=Hello world
5 Comando «world» non trovato
6 $ welcome='Hello world'
7 $ echo $welcome
8 Hello world
```

Esempio 11.1: *Assegnazione di una variabile e utilizzo*

in cui nella seconda (tentata) assegnazione si nota il carattere *spazio*, per cui è da inserire l'*escaping* del carattere o, come si è scelto, inserire la stringa tra doppi apici.
Per cancellare una variabile, è possibile utilizzare il comando `unset` come nel seguente esempio:

```
1 $ echo $welcome
2 Hello world
3 $ unset welcome
4 $ echo $welcome
```

Esempio 11.2: *Cancellazione di una variabile*

La procedura utilizzata in questa sezione è per la definizione di una *variabile di shell* denominata anche *variabile locale* poiché la validità della stessa è nella shell medesima,

ma la variabile non esisterà né in altre successive shell che saranno aperte, né in eventuali subprocess aperti nella medesima shell.

11.2 Variabili di environment

Nell'***Esempio 11.1*** si è creata una **variabile di shell** che rimarrà valida solo per la shell in uso. È però possibile promuovere tale variabile a **environment variable** spesso definita *variabile globale* poiché sarà visibile anche nei subprocess[1] creati. Le **variabile d'ambiente** non sono visibili solo nella shell di utilizzo, ma in tutti i *processi figli*[2]. Il processo di promozione si effettua tramite il comando `export`[3] come nell'esempio che segue:

```
1 $ welcome='Hello world!!'
2 $ export welcome
```

Esempio 11.3: *Assegnazione di una variabile e esportazione*

in cui si è effettuata una assegnazione di variabile, poi la sua esportazione a variabile d'ambiente. Alternativamente si può effettuare assegnazione ed esportazione con un unico passaggio come nell'esempio che segue:

```
1 $ export welcome='Hello world!!'
```

Esempio 11.4: *Assegnazione di una variabile con esportazione*

Ci sono già diverse variabili di ambiente impostate nel nostro environment, possiamo consultarle tramite il comando `export`[4] come di seguito:

```
1 $ export
2 declare -x HOME="/home/ms"
3 declare -x LANG="it_IT.UTF-8"
4 ...
5 declare -x LOGNAME="ms"
6 ...
7 declare -x PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
      bin:/usr/games:/usr/local/games:/snap/bin"
8 declare -x PWD="/home/ms"
9 declare -x SHELL="/bin/bash"
10 ...
11 declare -x TERM="xterm-256color"
12 declare -x USER="ms"
13 ...
```

Esempio 11.5: *Visualizzazione delle variabili di ambiente*

in cui l'output è stato troncato. Le principali sono sintetizzate nella ***Tabella 11.1***,
Se cancelliamo queste variabili con il comando della precedente sezione `unset`, come già

[1] personalmente ho riportato la locuzione *variabile globale* da molti utilizzata, ma in questo contesto non la preferisco favorendo quella di *variabile di environment* poiché la visibilità sarà esclusivamente nei subprocess e non in eventuali shell create parallelemente

[2] tali processi non dovranno essere necessariamente shell, ma sono ad esempio tutti i processi lanciati con comandi esterni

[3] similmente può essere utilizzato il comando `declare`, ma per gli scopi qui trattati sono equivalenti

[4] se al comando non è passato nessun parametro, sarà attivata l'opzione di default `-p` (quindi può non essere digitata)

NOME	DESCRIZIONE
HOME	*home directory* dell'utente
LOGNAME	alias dell'utente utilizzato nei log (solitamente coincidente con USER)
PATH	lista delle directory contenente gli eseguibili che possono essere eseguiti come comandi esterni
PS1	template del prompt della shell
PWD	nome della working directory
TERM	nome del terminale utilizzato
UID	*user id*
USER	*username* dell'utente

Tabella 11.1: *Descrizione delle principali variabili di environment*

scritto sarà cancellata come variabile di shell, quindi anche d'ambiente.
Per rimuoverla dall'ambiente ma non cancellare il valore della variabile di shell, si utilizza il comando `export -n` come di seguito:

```
1 $ echo $welcome
2 Hello world!!
3 $ export | grep welcome
4 declare -x welcome="Hello world!!"
5 $ export -n welcome
6 $ export | grep welcome
7 $ echo $welcome
8 Hello world!!
```

Esempio 11.6: *Rimozione della variabile dall'ambiente*

in cui:

- le prime due linee mostrano che la variabile `welcome` è sia di shell sia di ambiente;
- la terza linea per rimuovere la stessa dall'ambiente;
- la quarta per verificare che effettivamente non è più una variabile d'ambiente;
- la quinta per verificare che è rimasta la variabile di shell definita.

Le variabili di environment acquistano particolare importanza poiché ogni processo che sarà eseguito dalla shell *erediterà* le variabili d'ambiente impostate[5].

11.2.1 Stampa variabili d'ambiente^P

Nell'***Esempio 11.7*** è mostrato il semplice script `environment_variables.pl` per visualizzare le variabili di ambiente.

```
1 foreach my $key (sort keys %ENV) {
2     printf "%s => %s\n", $key, $ENV{$key};
3 }
```

Esempio 11.7: *Script Perl per visualizzare le variabili d'ambiente*

[5] ciò sarà mostrato nel seguito della sezione e tramite script in 11.2.1

L'output dello script è mostrato di seguito:

```
1 $ perl environment_variables.pl
2 HOME => /home/ms
3 LANG => it_IT.UTF-8
4 ...
5 LOGNAME => ms
6 PATH => /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
      games:/usr/local/games:/snap/bin
7 PWD => /home/ms
8 SHELL => /bin/bash
9 TERM => xterm-256color
10 USER => ms
11 ...
12 welcome => Hello world!!
```

Esempio 11.8: *Output dello script dell'***Esempio 11.7**

in cui si nota l'`export` della variabile `welcome` come suggerito in **??**.

11.3 `type` reloaded

Il comando `type` è già stato trattato in 2.3.2 a pag, 19, il quale ha lo scopo di distinguere tra comandi interni ed esterni. Ovviamente la shell conosce i comandi interni, ad esempio:

```
1 $ type type
2 type è un comando interno di shell
```

Esempio 11.9: *Esempio di comando interno*

ma è opportuno tornare sull'argomento in relazione ai comandi esterni.
I comandi esterni sono programmi eseguibili che la shell dovrà trovare, e la ricerca sarà effettuata in base al valore della variabile di environment `PATH`. Di seguito un tipico valore per `PATH`:

```
1 $ echo $PATH
2 /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/
      usr/local/games:/snap/bin
```

Esempio 11.10: *Variabile* `PATH`

La stringa fornita è una lista di directory, ciascuna è separata dall'altra dal carattere `:`, quindi nel precedente caso la lista è formata da 9 directory.
Ad esempio, se digitiamo il comando `find` ed Enter, non essendo un comando interno, la shell inizierà la ricerca di un comando omonimo tra i vari *path* elencati, quindi verificherà in sequenza se esiste un eseguibile concatenando ciascuna directory e il comando, come nella lista seguente:

```
1 $ find Enter
2 /usr/local/sbin/find  ⇒ no
3 /usr/local/bin/find  ⇒ no
4 /usr/sbin/find  ⇒ no
5 /usr/bin/find  ⇒ trovato!!
6 /sbin/find
7 /bin/find
8 /usr/games/find
9 /usr/local/games/find
10 /snap/bin/find
```

Esempio 11.11: *Ricerca del comando* `find` *secondo quanto indicato nella variabile* `PATH`

Quindi la ricerca:

- nei primi tre casi non ha avuto esito positivo;
- nel quarto caso ha trovato il riscontro e quello sarà l'eseguibile associato al comando;
- negli ultimi casi non è effettuata, per cui rimarranno percorsi non testati.

La ricerca effettuata è un processo lungo, per abbreviare i successivi utilizzi del comando, memorizza il *pathname* associato al comando, quindi memorizzerà l'associazione:

`comando` → `eseguibile`

che nel caso in oggetto sarà:

`find` → `/usr/bin/find`

Questo processo è denominato *hashing*, e si può verificare che è stato applicato digitando il `type` del comando:

```
1 $ type find
2 hash effettuato su find (/usr/bin/find)
```

Esempio 11.12: `type` *di* `find` *per verificare l'*hashing

Per verificare tutte le corrispondenze possiamo usare il comando `hash`:

```
1 $ hash
2 rich. comando
3    3  /usr/bin/which
4    1  /usr/bin/mv
5    2  /usr/bin/cp
6    2  /usr/bin/apropos
7    7  /usr/bin/find
8 ...
```

Esempio 11.13: *Comando* `hash`

Lo stesso comando senza parametri fornisce il numero di utilizzi del comando con l'hash memorizzato, ma è possibile gestire le singole entry, quindi rimuoverle o ricrearle. Ad esempio:

```
1 $ hash | wc -l
2 13
3 $ hash -r
4 $ hash
5 hash tabella di hash vuota
```

Esempio 11.14: *Rimozione entry dell'hash*

Come già visto nell'***Esempio 2.9*** a pag. 20, per verificare quale software utilizza un comando esterno, è possibile ricorrere al comando `which`:

```
1 $ which find
2 /usr/bin/find
```

Esempio 11.15: *Comando* `which` *di* `find`

che esegue lo stesso metodo della shell per cercare il pathname associato al comando[6], ma non utilizza il processo di hashing. Per verificarlo, è sufficiente digitare i seguenti comandi:

```
1 $ hash | grep find
2    1  /usr/bin/find
3 $ which find
4 /usr/bin/find
5 $ hash | grep find
6    1  /usr/bin/find
```

Esempio 11.16: *Verifica del non uso dell'hashing del comando* `which`

È però necessario porre attenzione che il comando `which` non conosce i comandi interni della shell, per cui anche se nella ricerca di un comando troverà un riscontro di eseguibile tra i path, i comandi interni avranno la precedenza[7]. Ad esempio, per il comando `echo`:

```
1 $ type echo
2 echo è un comando interno di shell
3 $ which echo
4 /usr/bin/echo
```

Esempio 11.17: `type` *vs* `which` *del comando* `echo`

la prima linea mostra che la shell tratta il comando come interno, ma `which` ha trovato un riscontro che non sarà richiamato utilizzando il comando `echo`, e lo si utilizzerà da shell solo richiamandone il path.

[6] quindi tramite la variabile `PATH`

[7] a meno che non si indichi il comando ma proprio il pathname da utilizzare

11.4 Concetti chiave

assegnazione: è effettuata tramite il carattere =

environment variable: v. *variabile d'ambiente*

hashing: per l'utilizzo di un comando esterno, la shell memorizza il pathname dell'eseguibile utilizzato, cosicché per il successivo utilizzo eviterà il processo di ricerca utilizzando direttamente la corrispondenza precedentemente memorizzata

path: sono i percorsi definiti nella variabile d'ambiente `PATH` in cui la shell cercherà gli eseguibili

variabili d'ambiente: variabili si shell promosse a *environment variable*, la cui visibilità è estesa anche ai processi generati nell'ambiente medesimo

variabili di shell: le variabili che sono state create tramite assegnazione

11.5 Comandi introdotti nel capitolo

= ⇒ assegnazione, non vanno inseriti caratteri di spazio né prima né dopo

`declare` ⇒ più flessibile del comando `export`, ma per usi base sono equivalenti

`hash` ⇒ gestisce il database di corrispondenze tra comando e pathname dell'eseguibile

`export` ⇒ promuove le variabili di shell a variabili di ambiente

`unset` ⇒ cancellazione di unavariabile di shell

Capitolo 12

Introduzione al filesystem

QESTO capitolo intende proseguire e argomentare con maggior profondità la trattazione precedente sui file e sulle directory[1].

12.1 Terminologia

Il termine *file* genericamente indica un contenitore di dati[2] immagazzinato su storage.
Il *filesystem* predispone il device (Hard disk, USB Key, ...) per l'immagazzinamento dei file e ne determina la gestione.

Per chiarire ulteriormente, nello storage sono memorizzati dati in forma di byte che il sistema dovrà successivamente reperire, inoltre dovranno essere organizzati in maniera il più efficiente e flessibile, anche con la particolarità di dover gestire file di dimensioni eterogenee[3]. Questi[4] dettagli sono gestiti dai filesystem, e i sistemi GNU/Linux conoscono molti tipi di filesystem[5] i quali si presenteranno all'utilizzatore in maniera molto simile: una gerarchia ad albero di file e directory, con file di differente tipologia.
L'operazione che permette di unire filesystem al sistema si definisce *montaggio*, in particolare il sistema eseguirà un *mounting* creando una nuova ramificazione al filesystem esistente attraverso un nuovo *path* il quale permetterà l'accesso alla nuova risorsa.

12.2 Tipi di file

Per individuare il contenuto di un file, anche se può essere un'indicazione l'estensione, possiamo utilizzare il comando `file` il cui argomento sarà la lista di file cui dovrà indicarne il tipo. Si vada l'esempio:

```
1 $ file /bin/grep /usr/bin/shasum /etc/passwd
2 /bin/grep:       ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
3                  dynamically linked, interpreter
4                  /lib64/ld-linux-x86-64.so.2,
```

[1] riferimento al capitolo 4 a pag. 31
[2] quindi semplice testo, immagini, audio, software o altro
[3] piccola, media, grande o grandissima
[4] e molti altri
[5] ad esempio *ext2*, *ext3*, *ext4*, *ReiserFS*, *XFS*, *JFFS2*, *btrfs*. Saranno argomentati in **??** a pag. **??**

```
5                   BuildID[sha1]=f4564437ed282494b69a191fcb07a56235cce8ff,
6                   for GNU/Linux 3.2.0, stripped
7 /usr/bin/shasum: Perl script text executable
8 /etc/passwd:     ASCII text
```

Esempio 12.1: *Determina il contenuto del file tramite il comando* `file`

in cui `file`[6] ha individuato per:

`/bin/grep`
: un compilato eseguibile;

`/usr/bin/shasum`
: script Perl eseguibile[7];

`/etc/passwd`
: un testo ASCII.

Ma se accettiamo la premessa utilizzata diffusamente nel documento *Tutti i file sono file*, quelli precedenti sono da inquadrare come *regular file*. Se si ignora il contenuto e si procede verso la medesima suddivisione effettuata dal filesystem per tipologia, la distinzione sarà:

regular: sono inclusi testo, immagini, audio, video, eseguibili, script, . . . ;

directory: sono essenzialmente tabelle per la la ricostruzione della struttura ad albero. Ogni record memorizzerà almeno i nomi dei file e gli *inode number*;

symbolic link: indica il path di redirezione per l'accesso al file[8];

device: è l'interfaccia ai vari device. Ad esempio `/dev/sda` è solitamente l'hard disk: ogni accesso in lettura o scrittura su file di questo storage è effettuato proprio a questo device;

FIFO: anche denominati *named pipe*, analogamente alle *pipe* via shell permettono una comunicazione tra processi, in cui un processo scrive alla *FIFO*, un secondo legge dalla FIFO[9];

Unix domain socket: analogamente alle FIFO costituiscono un metodo di **IPC**[10], ma a differenza delle precedenti è necessario uno strato di programmazione per utilizzarle e, soprattutto, permettono *bidirezionalità* nella comunicazione[11]

Nella ***Tabella 12.1*** sono riassunte le tipologie indicate.

[6] le regole per riconoscere il contenuto dei file sono indicare nella directory `/usr/share/file`. I dettagli possono essere consultati nel manuale tramite `man magic`

[7] è stato utilizzato per l'***Esempio 10.35***

[8] v. ***Esempio 4.27*** a pag. 44

[9] v. ***Esempio 12.6.3*** a pag. 148

[10] ***i**nter-**p**rocess **c**ommunication*, già incontrata in 8.3 a pag. 96

[11] v. ***Esempio 12.6.4*** a pag. 148

TIPO	`ls -l`	`ls -F`	CREATO CON
regular	`-`	`nome`	*vari programmi*
directory	`d`	`nome/`	`mkdir`
link simbolici	`l`	`nome@`	`ln -s`
device	`b o c`	`nome@`	`mknod`
FIFO (named pipe)	`p`	`nome|`	`mkfifo`
Unix domain socket	`s`	`nome=`	*programmazione*

Tabella 12.1: *Descrizione delle varie tipologie di file (si è utilizzato* `nome` *come placeholder)*

12.3 The Linux Directory Tree

Un sistema GNU/Linux è formata da diverse centinaia di migliaia di file ed è utile seguire alcune convenzioni per mantenere ordinata la struttura delle directory e dei file. La maggior parte delle distribuzioni segue lo standard definito dal ***F**ilesystem **H**ierarchy **S**tandard* (**FHS**) con piccole variazioni.

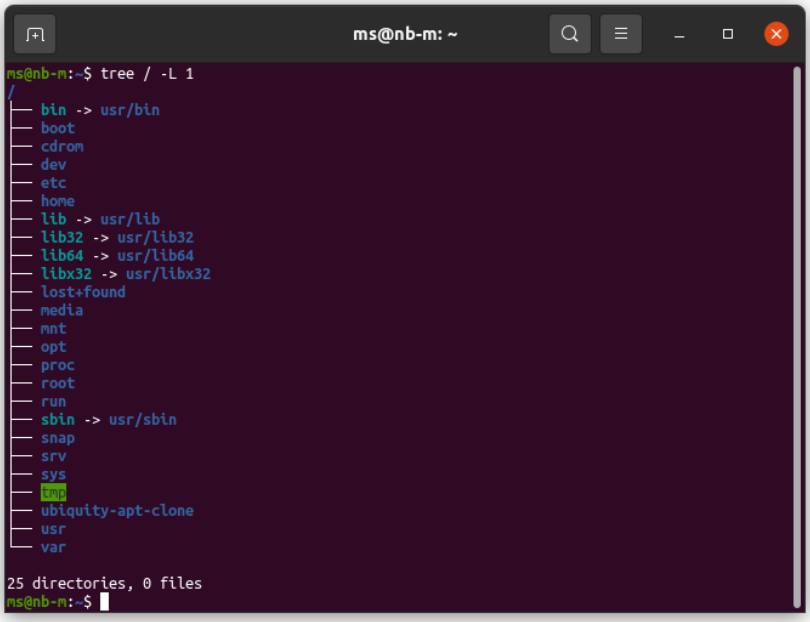


Figura 12.1: *Comando* `tree -L 1` *digitato sul pc dell'autore*

Partendo dalla *root directory*[12] `/`, la struttura di primo livello sarà simile a quella riportata in ***Figura 12.1***.
Di seguito saranno esaminate in maniera rigorosamente disordinata.

[12] anche definito livello 0

12.3.1 /boot

La directory /boot conterrà principalmente file relativi alla procedura di boot, in particolare:

boot loader: dipenderà dal *boot loader* installato, i più comuni sono *GRUB* e *LILO*;

vmlinuz: il *kernel* di Linux. Solitamente esistono diverse release e il file è un link simbolico[13] all'ultima;

initrd: un file system temporaneo utilizzato nella fase preliminare (precedente al caricamento del kernel)[14];

System.map: è un file di *symbol table*[15] usata dal kernel.

12.3.2 /bin e /usr/bin

Le directory /bin e /usr/bin[16] contengono molti eseguibili rispettivamente essenziali e non essenziali per la fase di boot. Ad esempio troveremo i binari per cat, ls, cp, mkdir, ...

12.3.3 /sbin e /usr/sbin

Simili a /bin e /usr/bin[17], contengono molti eseguibili rispettivamente essenziali e non essenziali per il sistema, come ad esempio fsck[18], mkfs[19], init[20] e route[21]. Sono per l'uso del amministratore di sistema, gli utenti ordinari solitamente possono solo usare tali strumenti per info sul sistema e non modifiche.

12.3.4 /lib e /usr/lib

Sono le *shared library* utilizzate dai vari software in /bin e /sbin per la prima directory, /usr/bin e /usr/sbin per la seconda directory[22]. Come suggerisce il termine *shared*, sono librerie richiamate da più software per cui l'essere esterne pone più vantaggi, ad esempio:

- molti processi utilizzano stessi moduli quindi, non essendo interni ai vari software, non saranno caricati in memoria multiple volte;
- se si presenta un bug su di una funzionalità di queste librerie, il bug fix dovrà essere effettuato in un unico segmento e tutti i software otterranno tale beneficio.

[13] equivalentemente per *initrd* e *System.map*

[14] più precisamente, **initrd** (***init****ial* ***ram*****d***isk*) è lo schema per il caricamento temporaneo di un *root file system* in memoria

[15] una *symbol table* è una *look-up table* tra simboli utilizzati (variabili, funzioni, ...) e il loro indirizzo in memoria

[16] come in questo caso, spesso la prima directory è un link simbolico alla seconda

[17] quindi anche in questo caso spesso la prima directory è un link simbolico alla seconda

[18] ***f****ile****s****ystem* ***c****hec****k***

[19] ***m****a****k****e* ***f****ile****s****ystem*

[20] ***init****ialization* è il primo processo lanciato dal kernel durante la fase di boot. Esso continua ad essere in esecuzione fino allo *shutdown*, ed è direttamente o indirettamente padre di tutti i processi

[21] è il comando per visualizzare e modificare la *IP routing table*, argomento trattato in **??** a pag. **??**

[22] e ovviamente ci sarà il link simbolico qualora sarà impostato per le suddette directory

Tra le tante subdirectory è presente `/lib/modules` che contiene i *moduli del kernel*[23] non necessariamente caricati e in uso.

12.3.5 `/dev`

Questa directory e le sue subdirectory contengono una notevole quantità di *special file*, ciascuno è una interfaccia per l'utilizzatore[24] al *device driver* del kernel, e spesso terminano con un numero definito *device number* il quale è l'identificativo del device.
Una delle principali distinzioni tra questi *file speciali* va effettuata tra:

character device: è un device che fornisce accesso diretto all'hardware, quindi processa singoli caratteri/byte senza *buffering* da parte del kernel. Ad esempio lo sono il terminale, il mouse, ...
Tali device sono anche conosciuti come *raw device*, per rimarcare la modalità di accesso diretto all'hardware;

block device: è un device che fornisce accesso al device fisico non diretto ma tramite *buffering*. Ovviamente la gestione dei dati può essere effettuata in lettura/scrittura di blocchi di qualsiasi dimensione (quindi anche singoli byte), ma poiché il flusso avviene tramite buffer, non è possibile conoscere quando il dato passa dal buffer al dispositivo fisico. Ad esempio lo sono tutti i dispositivi di storage: hard disk, USB Key, ... Per un elenco completo è utile il comando `lsblk`.

Un piccolo approfondimento su tali device è presentato in 12.6.2.
Per avere un esempio, digitando `ls -l` nella directory avremo una lunga lista di cui di seguito è riportato uno spezzone:

```
1  ...
2  crw-rw-rw- 1 root root 1, 7 nov 21 21:01 /dev/full
3  crw-r----- 1 root kmem 1, 1 nov 21 21:01 /dev/mem
4  crw-rw-rw- 1 root root 1, 3 nov 21 21:01 /dev/null
5  crw-rw-rw- 1 root root 1, 8 nov 21 21:01 /dev/random
6  brw-rw---- 1 root disk 8, 0 nov 21 21:01 /dev/sda
7  brw-rw---- 1 root disk 8, 1 nov 21 21:01 /dev/sda1
8  lrwxrwxrwx 1 root root   15 nov 21 21:01 /dev/stderr -> /proc/self/fd/2
9  lrwxrwxrwx 1 root root   15 nov 21 21:01 /dev/stdin -> /proc/self/fd/0
10 lrwxrwxrwx 1 root root   15 nov 21 21:01 /dev/stdout -> /proc/self/fd/1
11 crw-rw-rw- 1 root tty  5, 0 nov 30 15:21 /dev/tty
12 crw-rw-rw- 1 root root 1, 9 nov 21 21:01 /dev/urandom
13 crw-rw-rw- 1 root root 1, 5 nov 21 21:01 /dev/zero
14 ...
```

Esempio 12.2: *Alcuni file contenuti in* `/dev`

in cui, escludendo i link simbolici, nella prima colonna i file riporteranno come primo carattere una `c` o `b` in accordo alla distinzione appena effettuata, quindi rispettivamente device a *caratteri* o a *blocchi*.
Inoltre si nota che nell'elenco i file non mostrano la dimensione, ma due numeri che rappresentano i *device number*:

major device number: è il primo, specifica il tipo di device e indica quale driver del kernel gestirà tale device (ad esempio gli *HD SCSI* hanno 8 come *major device number*;

[23] moduli quali *device driver*, *filesystem*, *protocolli di network*, ...
[24] inteso utente o processo

minor device number: è il secondo, utilizzato dal driver per distinzioni tra dispositivi simili o, ad esempio, individuare la partizione[25] del disco.

Alcuni device maggiori sono i seguenti:

`/dev/mem`
: è il device della main memory

`/dev/hd`
: è l'*IDE driver*[26] per lo storage con:

  `/dev/hda`
  : per il device registrato sul master del primo canale IDE;

    `/dev/hda1`
    : per la prima partizione sul primo device;

    `/dev/hda2`
    : per la seconda partizione sul primo device;

    ...;

  `/dev/hdb`
  : per il device registrato sullo slave del primo canale;

  ...[27]

`/dev/nvme`
: per i moderni drive *NVMe*[28];

`/dev/sd`
: è lo *SCSI driver*[29] per lo storage con:

  `/dev/sda`
  : per il primo device registrato;

    `/dev/sda1`
    : per la prima partizione sul primo device;

    `/dev/sda2`
    : per la seconda partizione sul primo device;

    ...;

  `/dev/sdb`
  : per il secondo device registrato;

  ...

`/dev/sg`
: per *generic SCSI layer*;

`/dev/sr`
: per *ROM* driver (dischi ottici);

---

[25] quest'ultimo si nota in `/dev/sda` e `/dev/sda1`, argomento accennato in 12.4 e trattato in **??** a pag. **??**

[26] precedentemente usato per gli hard disk *ATA*, *ATAPI optical disc*, ...

[27] la tecnologie IDE arriva fino a `hdd`, slave del secondo canale

[28] ***N**on-**V**olatile **M**emory **E**xpress* controller device driver

[29] utilizzato anche da *libATA*, *USB*, ...

`/dev/st`
: per *magnetic tape* driver;

`/dev/ttyS`
: serial port driver;

`/dev/ttyUSB`
: serial port converter;

`/dev/xvd`
: per lo storage su cloud[30].

Pseudo device

Nella lista ci sono diversi device virtuali[31] notevoli esaminati di seguito.

`/dev/stdin`, `/dev/stdout` e `/dev/stderr` Già incontrati in 8.1 a pag. 91 forniscono gli accessi ai flussi standard del processo, rispettivamente `stdin`, `stdout` e `stderr`.

`/dev/null` Anche denominato *null device*, in gergo *cestino* o *buco nero* di bit, è un'entità a cui è possibile effettuare scritture di quantità arbitrarie di dati che saranno scartate, mentre in lettura restituirà un *end-of-file*. Ad esempio:

```
1 $ ./myprogram 2>/dev/null
2 $ ./myprogram >/dev/null 2>&1
```

Esempio 12.3: *Redirezione su* `/dev/null`

- la prima riga redirige lo `stderr` su `/dev/null`, quindi non riporterà mai errori in output;
- la seconda riga riga redirige `stdout` su `/dev/null`, `stderr` su `stdout`, quindi non riporterà nulla in output.

Per maggiori dettagli è utile consultare `man 4 null`.

`/dev/random` e `/dev/urandom` Questi device restituiscono byte randomici creati da rumore del sistema. Nella maggioranza degli scopi è sufficiente la pseudorandomicità fornita da `/dev/urandom`, ma la massima entropia (quindi randomicità di qualità crittografica) è fornita da `/dev/random`. Per maggiori dettagli è utile consultare `man 5 random` e `man 4 random`.

`/dev/tty` Il device `/dev/tty` rappresenta il terminale del processo corrente. Se ad esempio si digita:

```
1 $ echo 'Hello world!!'  > /dev/tty
2 Hello world!!
```

Esempio 12.4: `echo 1` *su di* `/dev/tty`

[30] essenzialmente identificato come (*virtual disk*

[31] nel senso che non corrispondono a nulla di fisico, ma sono device logici

si nota che la stringa inviata apparirà allo schermo. Se si digita:

```
1 $ cat /dev/tty
2 Hello world!!
3 Hello world!!
```

Esempio 12.5: *`cat` su `/dev/tty`*

l'input sarà duplicato e per terminare la solita sequenza Ctrl + D. Per maggiori dettagli è utile consultare `man 4 tty`.

`/dev/zero` Il device `/dev/zero` è un fornitore illimitato di byte ciascuno contenente il carattere `\0` (byte con valore zero[32]), byte con valore zero). Per la scrittura si comporta come `/dev/null`, quindi eventuali dati scritti saranno scartati.
Questo device è ad esempio utile utile per la creazione o la sovrascrittura con il comando `dd` come di seguito:

```
1 $ dd if=/dev/zero of=/tmp/a.txt count=1000 bs=4096
2 1000+0 record dentro
3 1000+0 record fuori
4 4096000 bytes (4,1 MB, 3,9 MiB) copied, 0,0137487 s, 298 MB/s
```

Esempio 12.6: *Uso di `dd` con fornitore `/dev/zero`*

in cui il comando:

- ha letto 1000 blocchi da `/dev/zero`
- ha scritto 1000 blocchi su `/tmp/a`;
- ciascun blocco è formato da 4096 byte (parametro **bs**), quindi ha scritto 4096000 byte;
- ha completato la copia in $\sim 14ms$, con una velocità calcolata di 298MB/s[33].

Per maggiori dettagli sul device, è utile consultare `man 4 zero`.

`/dev/full` Il lettura produce un flusso continuo di carattere *null* (byte di valore 0), quindi si comporta equivalentemente a `/dev/zero`.
Tentando la scrittura è generato un errore di tipo *disco pieno*[34], di qui la sua utilità: può essere testato da uno sviluppatore nella gestione di disco pieno.
Per maggiori dettagli sul device, è utile consultare `man 4 full`.

12.3.6 `/etc`

Questa directory contiene i file di configurazione per la maggior parte dei software del sistema. Di seguito alcuni file maggiori:

`/etc/fstab`[35]
: contrazione di ***filesystem table***, tipicamente elenca le partizioni e altri file system, le loro proprietà e come integrarli nella struttura del filesystem esistente;

[32] anche chiamato carattere *null*
[33] essendo l'intervallo temporale brevissimo, quest'ultimo valore è solo indicativo
[34] ENOSPC error
[35] v. `man 5 fstab`

`/etc/mtab`[36]
: mantiene la lista di tutti i filesystem correntemente integrati nel sistema;

`/etc/hosts`[37]
: fornisce una mappa statica degli indirizzi ip e i nomi assegnati agli host;

`/etc/init.d`
: questa directory contiene gli *init script* per vari servizi del sistema, solitamente utilizzati per eseguire lo *start* o lo *shutdown* dei servizi quando il sistema rispettivamente si sta avviando o spegnendo;

`/etc/passwd`[38]
: c'è la lista di tutti gli utenti conosciuti dal sistema, l'*user id*, *group id*, la sua *home directory*, interprete dei comandi, ...;

`/etc/shadow`[39]
: è il file contenente la lista di user, password cifrate[40] e altre informazioni correlate

`/etc/lsb-release`
: contiene informazioni relative alla distribuzione. È connesso al comando `lsb_release`, ad esempio:

```
1 $ cat /etc/lsb-release
2 DISTRIB\_ID=Ubuntu
3 DISTRIB\_RELEASE=22.04
4 DISTRIB\_CODENAME=jammy
5 DISTRIB\_DESCRIPTION="Ubuntu 22.04.3 LTS"
6 $ lsb_release -a
7 No LSB modules are available.
8 Distributor ID: Ubuntu
9 Description:  Ubuntu 22.04.3 LTS
10 Release:  22.04
11 Codename: jammy
```

Esempio 12.7: *Informazioni sulla release tramite* `lsb_release`

12.3.7 `/opt`

Con distribuzione appena installata, questa directory sarà vuota. Essa è stata predisposta per il software di terze parti (package preparati da fornitori, eventualmente da installare senza che creino conflitti con la distribuzione). Tali software andrebbero installati in `/opt/<package>`.

12.3.8 `/usr`

Contiene varie subdirectory contenenti programmi e dati non essenziali per il boot. Il contenuto per le più importanti è:

`/usr/bin`
: binari per gli utenti;

[36] v. `man 5 mtab`
[37] v. `man 5 hosts`
[38] v. `man 5 passwd`
[39] v. `man 5 shadow`
[40] v. `man 3 crypt`

`/usr/sbin`
: binari per l'utente `root`;

`/usr/lib`
: ulteriori librerie (non per il software contenuto in `/bin` o `/sbin`);

`/usr/local`
: eventuale software installato dall'amministratore di sistema[41];

`/usr/share`
: dati indipendenti dall'architettura. In particolare:

  `/usr/share/doc`
  : documentazione, ad esempio *HOWTO*;

  `/usr/share/info`
  : le pagine `info`

  `/usr/share/man`
  : le pagine di manuale consultate tramite `man`;

`/usr/src`
: sorgenti per il kernel e altro software;

12.3.9 /proc

La directory `/proc` è in realtà uno pseudofilesystem[42] poiché non occupa spazio su disco, ma i file e le subdirectory sono creati e aggiornati dal kernel e mantenuti in memoria. Sono qui contenute molte informazioni sui processi in esecuzione e altre che il kernel traccia in relazione all'hardware. Il contenuto più importante è:

`/proc/cpuinfo`
: informazioni relativo alle CPU (tipo, frequenza di clock, *core*, ...);

`/proc/devices`
: lista dei dispositivi supportati dal kernel, con incluso il *major device number*;

`/proc/interrupts`
: una lista degli interrupt hardware[43];

`/proc/ioports`
: equivalente al precedente ma per le porte di I/O;

`/proc/loadavg`
: informazioni numeriche sul carico medio[44], del sistema, in particolare i primi tre campi sono gli stessi che restituisce il comando `uptime` come nell'esempio:

```
1 $ cat /proc/loadavg
2 0.48 0.45 0.45 1/2148 896249
3 $ uptime
4  18:21:31 up 10 days, 21:19,  2 users,  load average: 0,48, 0,45,
       0,45
```

[41] come per `/opt`, qui la distribuzione non preinstalla nulla

[42] v. `man 5 proc`

[43] un interrupt appare nella lista solo se è presente un driver nel kernel che lo gestisce

[44] v. **??** a pag. **??**

Esempio 12.8: *Informazioni sul carico sul sistema tramite* `/proc/loadavg` *e il comando* `uptimed`

`/proc/meminfo`
: informazioni su memoria e swap. Il file è usato dal software `free`;

`/proc/mounts`
: informazioni relative ai filesystem montati, contenuto sovrapponibile a `/etc/mtab`;

`/proc/scsi`
: directory con le informazioni relative ai dispositivi SCSI disponibili;

`/proc/version`
: numero di versione e data di compilazione del kernel corrente;

`/proc/self/fd` e `/dev/fd`
: approfonditi in 12.6.5.

La fruizione di tale informazioni avverrà nel seguito del documento, a partire dal capitolo **??** a pag. 193.

12.3.10 `/sys`

Sin dalla versione 2.6, il kernel crea il contenuto di tale directory coerentemente ai dispositivi hardware. Probabilmente il trend che si sta seguendo è la migrazione del contenuto di `/proc` di ciò che non è relativo a singoli processi verso questa directory.
Come per la sezione precedente, la fruizione di tale informazioni avverrà nel seguito del documento, a partire dal capitolo **??** a pag. 193.

12.3.11 `/var`

La directory contiene directory tematiche i cui file cambiano continuamente con l'evoluzione del sistema, dei processi e più in generale di eventuali servizi offerti dal sistema. Alcune directory interessanti sono:

`/var/mail`
: conterrà le email non lette dagli utenti

`/var/spool`
: directory contenente subdirectory le quali memorizzano temporaneamente file, ad esempio:

  `/var/spool/cron`
  : contiene informazioni relative ai job che sono eseguiti in maniera automatica secondo un calendario[45];

  `/var/spool/cups`
  : quando un documento è stampato da un software, i dati da stampare sono prima memorizzati in questa directory e saranno cancellati solo quando la stampante li avrà processati;

[45] si tratterà l'argomento in **??** a pag. **??**

`/var/log`
: mantiene i record di eventi di sistema quali, ad esempio, *login* e *logout*, aggiornamenti automatici, o se il sistema fornisce servizi remoti qui è solitamente creata una directory che manterrà lo storico, ... Nel complesso ci riferisce genericamente al contenuto come *log files*[46]. Alcuni file/directory per tracce importanti sono:

  `/var/log/dmesg`
  : kernel message, accessibili tramite il comando `dmesg`[47];

  `/var/log/syslog`
  : log di sistema

12.3.12 /tmp

Molti software richiedono spazio temporaneo. In questa directory tutti i software potranno depositare il contenuto[48], talvolta solo transitoriamente considerando che molte distribuzioni effettuano una cancellazione dei dati in fase di *boot*.
Forse meno conosciuta è la directory `/var/tmp` che è equivalente alla precedente (tutti possono scrivere), ma tradizionalmente non è svuotata durante la fase di *startup*[49].

12.3.13 /srv

Tale directory è predisposta per eventuali demoni[50].
Questa directory è relativamente nuova per cui tradizionalmente ci si è appoggiati su altre posizioni nel filesystem secondo quanto predisposto dai system administrator che talvolta seguono il default degli autori di software.

12.3.14 /media

Il contenuto di questa directory è spesso generato in maniera automatica, in particolare solitamente avviene un *mounting*[51] connettendo una USB Key o altro dispositivo con memoria di massa.
Operativamente sarà definito un nuovo `path` che permetterà l'accesso alla memoria del nuovo device. Tale path sarà `/media/dispositivo` con `dispositivo` sarà una label che talvolta richiama il dispositivo, la marca o il modello dello stesso.

12.3.15 /mnt

Molto simile alla precedente directory, anch'essa pensata per il *montaggio* di dispositivi di storage. Alcune distribuzioni[52] utilizzano questa directory al posto della precedente.

---

[46] anche *file di log*, un piccolo approfondimento in 12.6 a pag. 144

[47] sarà successivamente esaminato in 24.2.1 a pag. 368

[48] questo comportamente può essere al momento considerato come un'*eccezione*, sarà trattato in 18 a pag. 227

[49] è utile consultare la documentazione della distribuzione per verificare tale comportamento che può deviare da quanto appena scritto

[50] approfondimento in **??** a pag. **??**

[51] l'operazione sarà illustrata in 12.6.1 a pag. 144

[52] storicamente la *Red Hat*

12.3.16 /home

Questa directory contiene le home directory degli utenti ad eccezione dell'utente *root*, quindi:

```
1 $ ls -C1 /home
2 /home/user1
3 /home/user2
4 /home/user3
5 ...
```

Esempio 12.9: *Elenco generico di home degli utenti*

Per realtà con centinaia di utenti, potrebbe aver maggior praticità aggiungere, tra la directory `/home` e la username dell'utente, il nome del gruppo, quindi un esempio potrebbe essere:

```
1 $ find /home/ -maxdepth 2 -mindepth 2 -type d | sort
2 /home/develop/user1
3 /home/develop/user2
4 /home/develop/user4
5 /home/hr/user3
6 /home/hr/user5
7 /home/support/user6
8 /home/support/user7
9 ...
```

Esempio 12.10: *Elenco generico di home degli utenti*

12.3.17 /root

Questa è la home directory dell'amministratore di sistema. La ragione per la separazione di quest'home con quella degli altri utenti deriva dalla considerazione che talvolta la `/home` è su un differente filesystem (partizione, hard disk, ...).

12.3.18 /lost+found

È una directory che contiene eventuali oggetti smarriti. Operativamente si utilizza il comando `fsck`[53] per verificare l'integrità del filesystem. Se il tool trova frammenti di dati che ritiene verosimile appartengano a file, ma non c'è un file che li referenzi[54], e se è stato eseguito aggiungendo l'opzione per la riparazione, il checker trasforma questi *quasi file* in file, ma non avendo più né un nome né una directory di appartenenza, saranno depositati nella directory `lost+found`[55].

12.4 Albero delle directory e filesystem

L'albero delle directory ha origine dal filesystem inizialmente caricato dalla partizione iniziale di boot, ma spesso si estende. Quest'integrazione avverrà sempre integrando il nuovo filesystem legandolo ad un path a partire dalla `/`.
Una scelta comune è separare la `/home` in una partizione separata dal sistema cosicché,

[53] ***file**system **ch**eck*, un esempio in **??** a pag. **??**

[54] ad esempio trova un inode che non ha un corrispondente nome

[55] il norme deriva proprio da questa azione: dopo essere stati persi (*lost*), sono stati ritrovati *found*

se si vorrà reinstallare da zero il SO (ad esempio provare una nuova distribuzione), non si dovrà preoccupare della `/home` in fase di installazione[56].
Il file `/etc/fstab` descrive come globalmente il filesystem è *assemblato*, quindi come le varie aggiunte di filesystem saranno montate all'albero iniziale (in maniera automatica) innestata proprio al *mount point* definito[57].
Le directory corrispondenti al mount point non devono essere necessariamente vuote, ma al momento del *mounting* il loro contenuto non sarà più accessibile fino al successivo *unmounting*, proprio perché accedendo alla medesima directory non si sarà più rediretti al filesystem del nuovo dispositivo, ma si rimarrà nello stesso.

12.5 Approfondimenti

12.6 File di log

I file di log sono solitamente gestiti da un servizio denominato *syslog*[58]. In maniera sintetica, il servizio `syslogd` accetta messaggi da altri software e li categorizza in base ad origine e priorità (*debugging*, *error*, ...), dopodiché li scriverà su file, li invierà ad un servizio di storicizzazione eventualmente remoto o li scriverà in console in base alla configurazione voluta.
L'eccezione è per i messaggi generati nella fase iniziale dal kernel, prima che parta il servizio di management suddetto. Per questi messaggi, il kernel il mantiene internamente e sono consultabili tramite il comando `dmesg`.

12.6.1 Montaggio manuale di un filesystem

L'operazione di montaggio di un device esterno dotato di memoria di massa permette l'innesto del nuovo filesystem al filesystem del sistema. Possiamo suddividere la procedura in tre fasi:

individuazione del dispositivo e partizione: saranno elencate diverse alternative per individuare il dispositivo e la partizione interessate;

mount: si eseguirà il montaggio della partizione su un path già esistente;

unmount: al termine si eseguirà lo smontaggio della partizione.

Individuazione del dispositivo

Il primo passaggio è l'individuazione del dispositivo, di seguito esaminiamo alcune procedure alternative tramite *syslog*, il comando `lsblk`, il comando `dmesg` oppure tramite consultazione dell'utility `fdisk`.
Per essere maggiormente precisi, le alternative sono per l'amministratore di sistema, l'utente potrà invocarle tutte solo se avrà accesso a tali privilegi[59].

[56] ad esclusione del momento in cui si individua la partizione da reinstallare, v. ***Esempio 12.15*** *a pag. 146* e ***Esempio ??*** *a pag.* **??**
[57] v. 12.6.1 per un montaggio manuale
[58] v. 24.3 a pag. 371
[59] ad esempio invocando `sudo`, argomento trattato in 16.3.2 a pag. 196

`syslog` Un'alternativa è mettersi in ascolto con `tail -f` del file `/var/log/syslog` prima di inserire il dispositivo, dopodiché inserirlo e leggere l'incremento di log come di seguito:

```
$ tail -f /var/log/syslog
...
Dec  4 23:37:47 nb-m kernel: [882762.751869] usb 1-1: new high-speed USB
    device number 6 using xhci_hcd
Dec  4 23:37:47 nb-m kernel: [882762.904174] usb 1-1: New USB device found
    , idVendor=abcd, idProduct=1234, bcdDevice= 1.00
Dec  4 23:37:47 nb-m kernel: [882762.904188] usb 1-1: New USB device
    strings: Mfr=1, Product=2, SerialNumber=3
Dec  4 23:37:47 nb-m kernel: [882762.904194] usb 1-1: Product: UDisk
Dec  4 23:37:47 nb-m kernel: [882762.904198] usb 1-1: Manufacturer:
    General
Dec  4 23:37:47 nb-m kernel: [882762.904202] usb 1-1: SerialNumber: #
Dec  4 23:37:47 nb-m kernel: [882762.905308] usb-storage 1-1:1.0: USB Mass
     Storage device detected
Dec  4 23:37:47 nb-m kernel: [882762.905779] scsi host2: usb-storage
    1-1:1.0
Dec  4 23:37:38 nb-m rtkit-daemon[4650]: Supervising 1 threads of 1
    processes of 1 users.
Dec  4 23:37:47 nb-m mtp-probe: checking bus 1, device 6: "/sys/devices/
    pci0000:00/0000:00:14.0/usb1/1-1"
Dec  4 23:37:47 nb-m mtp-probe: bus: 1, device: 6 was not an MTP device
Dec  4 23:37:47 nb-m mtp-probe: checking bus 1, device 6: "/sys/devices/
    pci0000:00/0000:00:14.0/usb1/1-1"
Dec  4 23:37:47 nb-m mtp-probe: bus: 1, device: 6 was not an MTP device
Dec  4 23:37:48 nb-m kernel: [882763.912222] scsi 2:0:0:0: Direct-Access
      General  UDisk            5.00 PQ: 0 ANSI: 2
Dec  4 23:37:48 nb-m kernel: [882763.912791] sd 2:0:0:0: Attached scsi
    generic sg2 type 0
Dec  4 23:37:48 nb-m kernel: [882763.913068] sd 2:0:0:0: [sdb] 31457280
    512-byte logical blocks: (16.1 GB/15.0 GiB)
Dec  4 23:37:48 nb-m kernel: [882763.913215] sd 2:0:0:0: [sdb] Write
    Protect is off
Dec  4 23:37:48 nb-m kernel: [882763.913221] sd 2:0:0:0: [sdb] Mode Sense:
     0b 00 00 08
Dec  4 23:37:48 nb-m kernel: [882763.913347] sd 2:0:0:0: [sdb] No Caching
    mode page found
Dec  4 23:37:48 nb-m kernel: [882763.913355] sd 2:0:0:0: [sdb] Assuming
    drive cache: write through
Dec  4 23:37:48 nb-m kernel: [882763.917459]  sdb: sdb1
Dec  4 23:37:48 nb-m kernel: [882763.918844] sd 2:0:0:0: [sdb] Attached
    SCSI removable disk
```

Esempio 12.11: *In ascolto del* `syslog`

ed è evidente che il dispositivo è `sdb` con la partizione da montare `sdb1`.

`lsblk` Una procedura utile può essere elencare i block device prima l'inserimento del dispositivo:

```
$ lsblk
...
sda      8:0    0 465,8G  0 disk
|-sda1   8:1    0   260M  0 part
|-sda2   8:2    0    16M  0 part
|-sda3   8:3    0 185,6G  0 part
|-sda4   8:4    0  16,3G  0 part /opt/bak
|-sda6   8:6    0 259,8G  0 part /
```

```
9 --sda7   8:7    0   3,9G  0 part [SWAP]
10 sr0     11:0    1 141,9M  0 rom
```

Esempio 12.12: *Lista dei dispositivi a blocchi prima dell'inserimento*

e dopo l'inserimento del dispositivo:

```
1 $ lsblk
2 ...
3 sda      8:0    0 465,8G  0 disk
4 |-sda1   8:1    0   260M  0 part
5 |-sda2   8:2    0    16M  0 part
6 |-sda3   8:3    0 185,6G  0 part
7 |-sda4   8:4    0  16,3G  0 part /opt/bak
8 |-sda6   8:6    0 259,8G  0 part /
9 --sda7   8:7    0   3,9G  0 part [SWAP]
10 sdb      8:16   1    15G  0 disk
11 --sdb1   8:17   1    15G  0 part
12 sr0     11:0    1 141,9M  0 rom
```

Esempio 12.13: *Lista dei dispositivi a blocchi dopo l'inserimento*

in cui è apparso il dispositivo `sdb` e la partizione `sdb1`.

`dmesg` Appena si inserisce la USB Key, digitando `dmesg` avremo:

```
1 $ dmesg
2 ...
3 [881746.940889] usb 1-1: New USB device found, idVendor=abcd, idProduct
      =1234, bcdDevice= 1.00
4 [881746.940905] usb 1-1: New USB device strings: Mfr=1, Product=2,
      SerialNumber=3
5 [881746.940911] usb 1-1: Product: UDisk
6 [881746.940915] usb 1-1: Manufacturer: General
7 [881746.940918] usb 1-1: SerialNumber: #
8 [881746.942038] usb-storage 1-1:1.0: USB Mass Storage device detected
9 [881746.942505] scsi host2: usb-storage 1-1:1.0
10 [881747.944798] scsi 2:0:0:0: Direct-Access     General  UDisk
      5.00 PQ: 0 ANSI: 2
11 [881747.945807] sd 2:0:0:0: [sdb] 31457280 512-byte logical blocks: (16.1
      GB/15.0 GiB)
12 [881747.945985] sd 2:0:0:0: [sdb] Write Protect is off
13 [881747.946005] sd 2:0:0:0: [sdb] Mode Sense: 0b 00 00 08
14 [881747.946024] sd 2:0:0:0: Attached scsi generic sg2 type 0
15 [881747.946203] sd 2:0:0:0: [sdb] No Caching mode page found
16 [881747.946220] sd 2:0:0:0: [sdb] Assuming drive cache: write through
17 [881747.948313]  sdb: sdb1
18 [881747.953029] sd 2:0:0:0: [sdb] Attached SCSI removable disk
```

Esempio 12.14: *Individuazione dispositivo e partizione tramite `dmesg`*

quindi è evidente che il dispositivo è `sdb` con la partizione da montare `sdb1`.

`fdisk -l` Digitando il comando `fdisk -l` avremo:

```
1 $ fdisk -l
2 ...
3
4 Disk /dev/sdb: 15 GiB, 16106127360 bytes, 31457280 sectors
5 Disk model: UDisk
6 Units: sectors of 1 * 512 = 512 bytes
```

```
7 Sector size (logical/physical): 512 bytes / 512 bytes
8 I/O size (minimum/optimal): 512 bytes / 512 bytes
9 Disklabel type: dos
10 Disk identifier: 0x26040b26
11
12 Dispositivo Avvio Start     Fine  Settori Size Id Tipo
13 /dev/sdb1   *      2048 31457279 31455232  15G  c W95 FAT32 (LBA)
```

Esempio 12.15: *Individuazione dispositivo e partizione tramite* `fdisk`

quindi è evidente che il dispositivo è `sdb` con la partizione da montare `sdb1`.

Mounting

Operativamente sarà definito un punto definito *mount point*. Sul filesistem del sistema è già predefinito il path `/mnt/usb`[60]:

```
1 $ mount /dev/sdb1 /mnt/usb
2 $ ls /mnt/usb
3 ...
```

Esempio 12.16: *Mounting manuale di una USB Key attraverso il comando* `mount`

in cui è visibile il contenuto del filesystem appena montato.

Unmounting

Sarà l'operazione opposta alla precedente. Prima ricontrolliamo il *mount point*, poi si procede all'effettivo smontaggio tramite il comando `umount`:

```
1 $ mount
2 ...
3 sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
4 proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
5 /dev/sda6 on / type ext4 (rw,relatime,errors=remount-ro)
6 ...
7 /dev/sdb1 on /mnt/usb type vfat (rw,relatime,uid=1000,gid=1000,fmask=0022,
      dmask=0022,codepage=437,iocharset=iso8859-1,shortname=mixed,showexec,
      utf8,flush,errors=remount-ro)
8 $ umount /mnt/usb
9 $ mount | grep usb
```

Esempio 12.17: *Unmounting manuale di una USB Key attraverso il comando* `umount`

in cui:

- il primo comando è per verificare i filesystem montati (e si individuerà il dispositivo da smontare);
- il comando `umount` effettuerà lo smontaggio dal filesystem;
- l'ultimo per verificare che non è più tra i dispositivi montati.

Equivalentemente si poteva optare per smontare non il *mount point* assegnato ma il dispositivo, quindi `umount /dev/sdb1`.

[60] altrimenti sarebbe stato necessario crearne uno

12.6.2 Device via buffer

Alcuni device necessitano[61] che il trasferimento dei dati non avvenga istantaneamente ma attraverso un buffer. Questo comporterà un'asincronicità nel trasferimento per cui non sarà più possibile stabilire quando il trasferimento al device fisico avvenga.
Inoltre, se lo stesso hardware è esposto sia tramite interfaccia a caratteri sia a blocchi. c'è il rischio di incoerenza di dati poiché gli utilizzatori del dispositivo a caratteri non sono a conoscenza delle modifiche apportate al buffer del dispositivo a blocchi.
Molti sistemi creano contemporaneamente interfacce sia a blocchi sia a caratteri per dispositivi di storage. FreeBSD e Linux hanno scelto di non seguire questa metodologia, ma hanno seguito strade diverse: il primo ha rimosso il supporto per dispositivi a blocchi, il secondo crea solo interfacce a blocchi.

12.6.3 Esempio di FIFO

Le *named pipe* sono uno speciale tipo si file che permettono la comunicazione tra processi. Per creare tale file si utilizza il comando `mkfifo` che semplicemente creerà il canale. Nell'esempio:

```
1 $ mkfifo pipe_here
2 $ file pipe_here
3 pipe_here: fifo (named pipe)
```

Esempio 12.18: *Creazione di una named pipe tramite* `mkfifo`

in cui il primo comando ha creato la *named pipe*, il secondo mostra che effettivamente è stata creata. Dopodiché nella stessa console metteremo del contenuto nella pipe:

```
1 $ echo "Scrivo nella named pipe" > pipe_here
```

Esempio 12.19: *Scrittura sulla named pipe*

e il processo rimarrà in attesa del destinatario (e del suo invio dell'*end-of-file*), quindi da un'altra console:

```
1 $ cat pipe_here
2 Scrivo nella named pipe
```

Esempio 12.20: *Lettura dalla named pipe*

in cui il processo destinatario riceverà il messaggio e un *end-of-file* dal mittente, il destinatario invierà il messaggio allo `stdout` e invierà il proprio *end-of-file*, quindi l'esecuzione per entrambi potrà terminare.
L'esempio di comunicazione si può complicare ponendo più scrittori contemporanei sulla medesima pipe, in cui l'atomicità dei messaggi ricevuti non è più garantita se la grandezza dei messaggi è superiore alla dimensione del buffer utilizzato denominato `PIPE_BUF`.

Per ulteriori si consulti il manuale `man 7 fifo`, `man 7 pipe`.

12.6.4 Esempio di Unix domain socket

L'*Unix domain socket* è un metodo di *IPC* che permette la comunicazione tra processi dello stesso host. Le differenze principali con le FIFO sono le seguenti:

[61] o semplicemente è stata preferita algoritmicamente tale via

bidirezionalità: consentono comunicazioni bidirezionali, quindi entrambi i processi possono inviare o ricevere dati;

programmazione: per la creazione dell'interfaccia necessitano di un livello aggiuntivo di complessità che che sarà colmato attraverso la programmazione.

Operativamente la differenza è l'interfaccia con cui si lavora, nelle FIFO si ha un *file interface*, nelle socket si ha una *Unix socket interface*.

Ci sono tantissimi modi per creare una *unix socket*, probabilmente il più semplice è ricorrere al comando `nc`[62], ad esempio permette di creare una socket:

```
1 $ nc -U /tmp/test.sock -l
```

Esempio 12.21: *Creazione di una Unix domain socket tramite* `nc`

con:

`-U /tmp/test.sock`
: creerà una Unix socket come indicato

`-l`
: è il parametro per *listen for an incoming connection*, quindi si comporterà da server.

Per verificare che il file sia stato creato ma soprattutto che la tipologia sia corretta, da una nuova console si ricorrerà al comando `lsof` che elencherà i file aperti, aggiungendo come argomento il file riporterà i dettagli dello stesso:

```
1 $ lsof /tmp/test.sock 2>/dev/null
2 COMMAND     PID USER   FD   TYPE             DEVICE SIZE/OFF     NODE NAME
3 nc      1425676   ms    3u  unix 0x0000000000000000      0t0 48737589 /tmp
      /test.sock type=STREAM
```

Esempio 12.22: *Verifica della socket tramite* `lsof`

in cui le principali colonne sono:

`COMMAND`
: abbiamo il comando eseguito;

`PID`
: il *process ID*[63];

`FD`
: il *file descriptor*[64];

`TYPE`
: `unix` per indicare il tipo *Unix domain socket*;

`NAME`
: il nome del file (che in questo caso è ovvio poiché è stato filtrato), `type=STREAM` indica che il socket è di tipo `SOCK_STREAM` quindi orientato alla connessione.
Tra le varie tipologie, l'alternativa principale è `SOCK_DGRAM` la quale è senza connessione. Questa criptica differenza sarà mostrata nell'***Esempio* ??**, ma sarà più chiara al capitolo 20 a pag. 273.

[62] equivalentemente `netcat`, utility estremamente versatile. Per ulteriori v. **??** a pag. **??**
[63] sarà esaminato in 16.5 a pag. 198
[64] esaminato in 12.6.5

Dalla nuova console si eseguirà il client con identica chiamata ad eccezione dell'opzione `-l`, digitando poi una stringa e premento Enter come di seguito:

```
1 $ nc -U /tmp/test.sock
2 Hello world!!
```

Esempio 12.23: *Connessione alla Unix domain socket creata precedentemente*

la stessa stringa appartirà anche all'altro capo della socket. Chiudendo una qualunque delle socket, si chiuderà anche l'altra poiché la socket è stata aperta con tipologia `STREAM`.

Ripercorrendo gli esempi di questa sezione non con la tipologia `SOCK_STREAM` che è orientata alla connessione, ma con `SOCK_DGRAM` non orientata alla connessione. Quindi lato server si aprirà la connessione con

```
1 $ nc -U /tmp/test.sock -l -u
```

Esempio 12.24: *Creazione di una Unix domain socket tramite `nc` ma non orientata alla connessione*

poi aprendo una seconda console si verificherà la stessa:

```
1 $ lsof /tmp/test.sock 2>/dev/null
2 COMMAND     PID USER   FD   TYPE             DEVICE SIZE/OFF     NODE NAME
3 nc      1574734   ms    3u  unix 0x0000000000000000      0t0 52646037 /tmp
      /test.sock type=DGRAM
```

Esempio 12.25: *Verifica della socket non orientata alla connessione tramite `lsof`*

in cui la principale colonna da notare è l'ultima con:

`NAME`
: il nome del file (che in questo caso è ovvio poiché è stato filtrato), `type=DGRAM` indica che il socket è di tipo `SOCK_DGRAM`, quindi non è orientato alla connessione.

Dalla nuova console si eseguirà il client con identica chiamata ad eccezione dell'opzione `-l`, digitando poi una stringa e premento Enter come di seguito:

```
1 $ nc -U /tmp/test.sock -u
2 Hello world!!
```

Esempio 12.26: *Connessione alla Unix domain socket creata precedentemente*

la stessa stringa appartirà anche all'altro capo della socket. Però questa volta chiudendo una qualunque delle socket, non si chiuderà anche l'altra in quanto la socket è stata aperta con tipologia `SOCK_DGRAM`.

12.6.5 I file descriptor

I ***file descriptor*** (**FD**) sono già stati incontrati in 8.3 a pag.96, in particolare i valori 0, 1 e 2 sono assegnati all'avvio ad ogni processo rispettivamente allo `stdin`, `stdout` e `stderr`.
Più in generale i *file descriptor* sono un identificativo univoco per ogni processo per i file

o altre risorse di input/output, quindi file speciali (*block device* o *character device*, pipe, socket.
Gli *FD* assegnati sono indicizzati in una tabella denominata **file descriptor table** che sarà creata per ogni processo, ciascuna sarà una partizione di una sezione di una tabella più generale denominata **file table** mantenuta dal kernel a livello di sistema. Essa sarà composta da tre sezioni:

file descriptor table: in cui saranno indicati processo e valore numerico;

modalità di accesso: *read*, *write*, *read/write* e eventualmente altre modalità;

inode table: indirizzo della risorsa.

Per ogni nuovo processo alla **file descriptor table** saranno aggiunte 3 entry per i *file descriptor* assegnati agli *standard streams*, successivamente nuove entry saranno aggiunte o cancellate a seconda della richiesta di risorse. Per eseguire attività di input o output, il processo passerà il valore di *FD* al kernel tramite una *system call*, ed il kernel accederà al file per conto del processo[65].
L'insieme degli *FD* di un processo sono elencati in `/proc/$PID/fd/` in cui `$PID` è l'identificativo del processo. Quindi un processo `$PID` appena avviato avrà assegnati `/proc/$PID/fd/0` per `stdin`, `/proc/$PID/fd/1` per `stdout` e `/proc/$PID/fd/2` per `stderr`. Inoltre Linux crea alcune scorciatoie in quanto ogni processo potrà accedere ai propri *FD* attraverso le directory `/proc/self/fd` e dal suo link simbolico `/dev/fd`.

Analogamente alla redirezione dei *file descriptor* standard[66], è possibile aprirne di nuovi con:

`FD<` per la lettura da `FD`;

`FD>` per la scrittura su `FD`;

`FD>>` per l'append su `FD`;

`FD<>` per la lettura/scrittura da/su `FD`;

in cui:

`FD` è un intero che rappresenta il nuovo *file descriptor*;

`...` può esserci sia un file sia un altro *file descriptor* preceduto dal carattere `&`[67].

È quindi necessario determinare a quale file (o *file descriptor* se preceduto da `&`) il nuovo *file descriptor* va associato. Il modo più semplice è utilizzare l'istruzione `exec` senza argomenti, e dopo l'uso andrà chiuso attraverso l'espressione `&-` dopo l'espressione equivalente all'apertura.
Di seguito un esempio:

```
1 $ exec 3> file.txt
2 $ ls -al >&3
3 $ ls -al
4 ...
5 $ exec 3>&-
```

Esempio 12.27: *Esempio di uso di* file descriptor

[65] in altri termini, il processo non potrà accedere direttamente alla risorsa
[66] in 8.2.2 a pag. 93
[67] indicando quindi la concatenazione

in cui:

`exec 3> file.txt` ha aperto l'*FD* in scrittura associandolo a `file.txt`;

`ls -al >&3` ha inviato l'output del comando al *FD*;

`exec 3>&-` ha chiuso il *FD*.

Da considerare che non è garantita la scrittura sul file finché il *file descriptor* associato non sarà chiuso.

12.7 Concetti chiave

buffer: genericamente è un'entità di memoria usata per la conservazione temporanea di dati destinati ad un altro device oppure per una rielaborazione

block device: è un device che fornisce accesso al device fisico non diretto ma tramite *buffering*.Lo sono tutti i dispositivi di storage: hard disk, USB Key, ...

boot loader: è un software che permette il booting del computer. Quando un sistema si avvia (*bootstrap*), il *boot loader* è deputato a caricare il kernel del sistema operativo dalla memoria non volatile alla memoria primaria

character device: è un device che fornisce accesso diretto all'hardware, quindi senza *buffering* da parte del kernel. Ad esempio lo sono il terminale, il mouse, ...

device: un dispositivo genericamente può interfacciarsi con un sistema tramite un *device driver*

device driver: tipicamente un modulo che controlla un particolare tipologia di device

end-of-file: anche **EOF**;

FD: v. *file descriptor*

filesystem: genericamente predispone il device per l'immagazzinamento dei file e ne determina la loro gestione

file descriptor: per esteso *POSIX file descriptor* o abbreviato in *FD*, identificativo per ciascun processo delle risorse di input/output assegnate

special file: anche *file speciali*, sono interfacce all'hardware del sistema e si distinguono in *block device* o *character device*

FIFO: file speciale di tipo ***f**irst-**i**n **f**irst-**o**ut*, anche denominato *named pipe*

kernel: è il core di un SO (ed è il secondo termine nel nome GNU/Linux) ed ha il completo controllo di tutte cià che accade nel sistema. È la porzione sempre residente in memoria e permette le interazioni tra le componenti hardware e software

link simbolici: tipologia di file (v. anche 4.6.2 a pag. 43)

mounting: *montaggio*, è l'operazione che permette di unire filesystem al sistema, in particolare il sistema creerà una nuova ramificazione al filesystem esistente attraverso un nuovo *path* deputato all'accesso alla nuova risorsa

mount point: operazione per innestare il filesystem di un dispositivo al filesystem del sistema, definendo un nuovo `path` su quest'ultimo affinché si crei una nuova ramificazione che permetterà l'accesso alla memoria del nuovo device

named pipe: v. FIFO

raw device: device che fornisce accesso diretto all'hardware (v. *character device*)

regular: è un tipo di file (di testo, audio, immagini, video, ...). È denominato *regular* per distinguerlo dalle altre tipologie considerate speciali

socket: termine generico, nel capitolo utilizzato per *Unix domain socket*

system call: meccanismo utilizzato da un processo a livello utente o livello applicativo per richiedere un servizio a livello kernel del SO

storage: dispositivi di memorizzazione persistente

Unix domain socket: anche *IPC socket* (***i**nter-**p**rocess **c**ommunication socket*) è un estremitàdi un canale di comunicazione per lo scambio tra processi dello stesso host

12.8 Comandi introdotti nel capitolo

`dmesg` ⇒ mostra i messaggi del kernel dal boot

`exec` ⇒ nel capitolo utilizzato per gestione dei *file descriptor*

`fdisk` ⇒ manipola la tabella delle partizioni del disco

`file` ⇒ determina il tipo di file

`FD<` ⇒ lettura da `FD`

`FD>` ⇒ scrittura su `FD`

`FD>>` ⇒ append su `FD`

`FD<>` ⇒ lettura/scrittura da/su `FD`

`fsck` ⇒ check e riparazione di filesystem

`free` ⇒ mostra la quantità di memoria libera, totale e usata

`lsof` ⇒ mostra la lista dei file aperti, indicando il tipo (regular file, directory, block file, character file, Unix domain socket, Internet socket, . . .)

`lsb_release` ⇒ mostra informazionirelative alla distribuzione

`lsblk` ⇒ elenca i *block devices*

`mkfifo` ⇒ crea *FIFO* (named pipe)

`mknod` ⇒ crea file a *character* o *block device*

`mount` ⇒ monta un filesystem

`nc` ⇒ o `netcat`, nel capitolo utilizzato per la realizzazione di *Unix domain sockets*

`netcat` ⇒ alias di `nc`

`umount` ⇒ smonta un filesystem

`uptime` ⇒ mostra da quanto tempo il sistema è running

Capitolo 13

La shell Bash

QUESTO capitolo affronta argomenti inerenti la shell *Bash*, alcuni dei quali saranno già stati incontrati più o meno casualmente dallo sperimentatore.
È utile segnalare che alcune sezioni possono adattarsi alle altre shell con piccole variazioni.

13.1 Prima del comando

In 2.3.1 si era esaminata una generica sintassi di un comando riportata di seguito:

```
1 $ command [OPTIONS] arguments
```

Esempio 13.1: *Esecuzione di una sequenza di comandi*

In realtà è possibile far precedere il comando da una moltitudine di variabili con assegnazione; esse avranno precedenza rispetto le variabili di environment per il comando medesimo.
Ad esempio, in questa linea:

```
1 $ PWD=3 SHELL=2 USER=1 comando
```

Esempio 13.2: *Sequenza di variabili prima del comando*

il `comando` avrà impostate nel proprio environment tali variabili aggiungendole all'environment ereditato dal sistema, ma sono necessarie due precisazioni:

- hanno precedenza rispetto le variabili di *environment del sistema*: in caso di omonimia, i valori delle variabili dell'*environment del comando* saranno quelli appena settati;
- sono *temporanee*: terminata l'esecuzione del comando, ovviamente sparirà l'*environment del comando* e di conseguenza la disponibilità delle variabili.

In 13.8.1 a pag. 166 sarà mostrato un esempio.

13.2 Singola linea

Alle volte per brevità è comodo inserire più comandi in una singola linea. Solitamente si utilizza come separatore il carattere `;` come mostrato di seguito:

```
1 $ echo; date; echo; ncal; echo
2
3 gio 14 dic 2023, 17:54:49, CET
4
5     Dicembre 2023
6 lu      4 11 18 25
7 ma      5 12 19 26
8 me      6 13 20 27
9 gi      7 14 21 28
10 ve  1  8 15 22 29
11 sa  2  9 16 23 30
12 do  3 10 17 24 31
```

Esempio 13.3: *Sequenza di comandi su singola linea*

in cui nella successione i comandi sono eseguiti al termine del precedente, in particolare:

`echo` senza argomenti è utilizzato per andare a capo;

`date` mostra la data e orario odierni;

`ncal` mostra il calendario.

Talvolta è utile eseguire un comando solo se quello che lo precede è stato eseguito correttamente.

Breve approfondimento sugli *exit status*[1] In generale tutti i processi del sistema terminano restituendo un valore che identifica se siano terminati correttamente o se un qualche errore sia avvenuto. In estrema sintesi, nel primo caso il valore restituito sarà 0, nell'altro un valore diverso da 0.
È possibile richiedere da shell se un processo appena terminato sia terminato correttamente ispezionando la variabile `$?`. Ad esempio:

```
1 $ bash
2 $ exit 25
3 exit
4 $ echo $?
5 25
```

Esempio 13.4: *Verifica dell'*exit code

in cui:

`bash` esegue una nuova shell *bash*;

`exit 25` esce con l'*exit code* 25

`echo $?` mostra l'*exit code* del processo appena terminato.

Per l'***Esempio 13.4***, è da notare che il primo e il terzo prompt sono restituiti dal processo *bash* iniziale (definito *parent*), il secondo prompt è restituito dal comando `bash` eseguito dalla prima linea (definito *child*)[2].

Tornando alla questione della sequenza di comandi, supponiamo si voglia eseguire un secondo comando solo se il primo avrà terminato con successo. Per tale scopo, non si usa il separatore precedente, ma i caratteri `&&`[3] come di seguito:

[1] si affronterà l'argomento in **??** a pag. **??**
[2] si affronterà l'argomento in 16.5 a pag. 198
[3] l'operatore sarà meglio argomentato in 22.5.10 a pag. 330

```
1 $ bash -c 'exit 0' && echo 'Ok!!'
2 Ok!!
3 $ bash -c 'exit 25' && echo 'Ok!!'
```

Esempio 13.5: *Esecuzione di comandi separati da* `&&`

in cui:

`bash -c` esegue l'argomento, quindi nella shell che sarà apertà eseguirà il comando `exit`;

`exit 0` e `exit 25` rispettivamente terminano l'esecuzione con l'*exit code* indicata;

`&&` esegue il comando a destra solo se quello a sinistra ha restituito *exit code* 0, infatti nella seconda sequenza non è stampato nulla.

In maniera complementare si comporta il separatore `||`, in cui il comando a destra sarà eseguito solo se quello a sinistra restituisce un *exit code* diverso da 0.
Ad esempio:

```
1 $ bash -c 'exit 0' || echo 'Problem!!'
2 $ bash -c 'exit 25' || echo 'Problem!!'
3 Problem!!
```

Esempio 13.6: *Esecuzione di comandi separati da* `||`

in cui solo la seconda sequenza arriverà al termine mostrando il messaggio (che ovviamente poteva essere un qualunque altro comando).

13.3 `alias` e `unalias`

In 4.4 a pag. 36 si erano già incontrati alcuni `alias` impostati nella mia shell. Ricapitolando, impostare un'*alias* significa che la shell (successivamente all'operazione stessa) sostituirà in nome dell'*alias* con un altro comando e eventuali parametri aggiuntivi o addirittura una successione di comandi.
Digitando quanto segue:

```
1 $ alias nome_alias='comando -opzioni argomenti'
```

Esempio 13.7: *Generica impostazione di un alias*

si imposta un *alias*. Un esempio potrebbe essere:

```
1 $ alias hi='Hello world!!'
```

Esempio 13.8: *Impostazione dell'alias* `hi`

e sin da subito è possibile digitare l'alias appena creato:

```
1 $ hi
2 Hello world!!
```

Esempio 13.9: *Esecuzione dell'alias* `hi`

Digitando il comando `alias` senza parametri si ottiene la lista degli *alias* già impostati, e nella mia shell sono i seguenti:

```
1 $ alias
2 ...
3 alias egrep='egrep --color=auto'
4 alias fgrep='fgrep --color=auto'
5 alias grep='grep --color=auto'
6 alias hello='echo "Hello world!!"'
7 alias hi='echo "Hello world!!"'
8 alias l='ls -CF'
9 alias la='ls -A'
10 alias ll='ls -alF'
11 alias ls='ls --color=auto'
12 alias ncal='ncal -w'
```

Esempio 13.10: *Impostazione di un alias* `hi`

in cui nell'elenco si noterà quello impostato, quindi ne imposteremo uno più significativo:

```
1 $ alias lm='ls -al | more'
2 $ lm
3 ...
```

Esempio 13.11: *Impostazione dell'alias* `lm`

in cui al posto dei `...` ci sarà l'elenco fornito da `ls -l` in pipe con `more`. Un altro alias simpatico potrebbe essere il seguente:

```
1 $ alias rm='rm -i'
2 $ touch a.txt
3 $ rm a.txt
4 rm:  rimuovere file regolare vuoto 'a.txt'?
```

Esempio 13.12: *Impostazione dell'alias* `rm`

Un ultimo esempio:

```
1 $ echo "Hello $USER"
2 Hello ms
3 $ alias hi='echo "Hello $USER"'
4 $ hi
5 Hello ms
```

Esempio 13.13: *Nuova impostazione dell'alias* `hi`

In ogni caso gli *alias* hanno un'utilità molto limitata, poiché, ad esempio, *bash*:

- non espande gli argomenti passati come argomenti ma li concatena;
- non si espande ricorsivamente.

Un esempio della limitazione:

```
1 $ alias mkcd='mkdir "$1"; cd "$1"'
2 $ mkcd test
3 mkdir: impossibile creare la directory "": File o directory non esistente
```

Esempio 13.14: *Alias che non funziona come si potrebbe ipotizzare*

L'*alias* che ci si propone di creare è risolvibile associando allo stesso una funzione[4] come di seguito:

[4] argomento trattato in 22.7 a pag. 340

```
1 ~$ alias mkcd='function f_mkcd(){ mkdir "$1"; cd "$1"; }; f_mkcd'
2 ~$ mkcd test
3 ~/test$
```

Esempio 13.15: *Alias che non funziona come si potrebbe ipotizzare*

in cui si è riportata anche l'espansione della shell per evidenziare il nuovo path.

Se apriamo una nuova console, gli *alias* appena creati non ci saranno, quindi servirà un meccanismo affinché gli stessi siano esportati ad ogni shell. Questo aspetto sarà esaminato in 13.6. Comunque, qualora si voglia cancellare un *alias* appena creato, si ricorre al comando `unalias` come si seguito:

```
1 $ unalias lm
```

Esempio 13.16: *Cancellazione dell'alias* `hi`

ed abbiamo la prova richiedendo nuovamente la lista degli alias impostati per la shell:

```
1 $ alias
2 ...
3 alias egrep='egrep --color=auto'
4 alias fgrep='fgrep --color=auto'
5 alias grep='grep --color=auto'
6 alias hello='echo "Hello world!!"'
7 alias hi='echo "Hello world!!"'
8 alias l='ls -CF'
9 alias la='ls -A'
10 alias ll='ls -alF'
11 alias ls='ls --color=auto'
12 alias mkcd='function f_mkcd(){ mkdir "$1"; cd "$1"; }; f_mkcd'
13 alias rm='rm -i'
14 alias ncal='ncal -w'
```

Esempio 13.17: *Impostazione di un alias* `hi`

o semplicemente si può digitare l'alias cancellato:

```
1 $ lm
2 lm: comando non trovato
```

Esempio 13.18: *Esecuzione dell'alias* `hi` *appena cancellato*

13.4 Storico

La shell ricorda i comandi più recenti attraverso lo *storico* consultabile tramite il comando `history`. Ci si può spostare nello storico utilizzando i tasti ↑ e ↓ e, appena trovato il comando cercato, si può scrivere/cancellare o inviare attraverso il tasto ↩.
È anche possibile ricercare utilizzando la sequenza Ctrl + r che permetterà una ricerca mirata digitando una sequenza di caratteri. Possiamo affinare la ricerca digitandone ulteriormente caratteri, oppure tornare all'immediato precedente (sempre rispettando il filtro impostato) ancora tramite la sequenza Ctrl + r.
Ad esempio:

```
1 $ Ctrl + r
2 digitando  m
3 (reverse-i-search)'m': man man
4 digitando  y
5 (reverse-i-search)'my': mysql
6 digitando   Ctrl + r
7 cd lib ; dbicdump -o debug=1 -o components='[qw( InflateColumn::DateTime )
      ]' ...
8 ...
```

Esempio 13.19: *Ricerca nello storico tramite* Ctrl + r

si arriveràal comando voluto, eventualmente da modificare e poi premendo ↩, oppure un Ctrl + c se non si è trovato ciò di cui si necessita.

Per visualizzare la cronologia completa, è sufficiente digitare il comando `history` come di seguito:

```
1 $ history | tail -n5
2 2224 comando_n-4
3 2225 comando_n-3
4 2226 comando_n-2
5 2227 comando_n-1
6 2228 comando_n
```

Esempio 13.20: *Stampa della cronologia*

in cui si è scelto di riportare solo gli ultimi 5 comandi.
Per richiamare direttamente un comando dalla cronologia è possibile utilizzare il metacarattere `!`, in particolare:

`!n` richiama il comando corrispondente ad `n`; quindi se si volesse richiamare `comando_n-2` si digiterà `!2226`;

`!-n` per richiamare l'ultimo comando immesso sarà `!-1`, il penultimo `!-2`, e così via;

`!!` si riferisce al precedente comando, quindi è un sinonimo di `!-1`;

`!string` si riferisce al comando più recente che inizia con `string`.

Per ulteriori sostituzioni, si veda `man history`.

Per avere una *history* aggiornata, quando l'utente di disconnette dalla console di sistema, la shell trasferisce il buffer della *cronologia dei comandi* immessi al file definito dalla variabile `$HISTFILE`. Solitamente questa variabile punta al file `.bash_history` nella propria home. Ad esempio:

```
1 $ echo $HISTFILE
2 /home/ms/.bash_history
```

Esempio 13.21: *Valore di $HISTFILE*

che sarà popolato in *append* alla disconnessione, cosicché alla riconnessione l'utente avrà la cronologia riportata nel file aggiornata.
Le altre variabili di cui tener conto sono due:

`$HISTFILESIZE` indica il numero soglia di comandi che possono essere memorizzati nel file `.bash_history`;

`$HISTSIZE` è il numero che indica la soglia di comandi memorizzati nel buffer della cronologia, quindi, al raggiungimento di tale valore, i comandi più vecchi saranno scartati a favore dei nuovi.

È utile vedere il valore di default delle due variabili:

```
1 $ echo $HISTSIZE
2 1000
3 $ echo $HISTFILESIZE
4 2000
```

Esempio 13.22: *Valori di $HISTSIZE e $HISTFILESIZE*

che ovviamente possono essere modificati reimpostandoli[5]. Per verificare il loro effetto è sufficiente digitare:

```
1 $ history | wc -l
2 1000
3 $ cat $HISTFILE | wc -l
4 2000
```

Esempio 13.23: *Verifica del comportamento $HISTSIZE e $HISTFILESIZE*

Qualora si volesse modificare la cronologia passata, sarebbe banale modificarla[6].
È anche possibile evitare che un comando finisca nel buffer della cronologia (e successivamente popoli il file suddetto), immettendo uno spazio ad inizio della linea del comando.
Per evitare che tutta la sessione sia memorizzata, è possibile eseguire *unset* della variabile precedente `$HISTFILE`, quindi:

```
1 $ unset HISTFILE
```

Esempio 13.24: *Valore di $HISTFILE*

13.5 Directory stack

Questa sezione avrà per oggetto lo spostamento tra directory per cui si è preferito mostrare la directory corrente anche nel prompt.
Talvolta un'operazione ripetitiva è doversi spostare frequentemente da una directory di lavoro ad un'altra, spesso rimbalzando tra esse. È possibile velocizzare il passaggio tra directory di lavoro già visitate sfruttando lo *stack di directory* attraverso i comandi builtin:

`dirs` per visualizzare il contenuto dello stack;

`pushd` per aggiungere directory allo stack;

`popd` per eliminare directory dallo stack.

Lo stack è inizierà sempre dalla directory corrente, quindi:

```
1 ~$ dirs
2 ~
```

Esempio 13.25: *Stack iniziale*

[5] v. 13.6
[6] il contenuto di `.bash_history` è modificabile con un qualunque editor

il comando `dirs`[7] mostrerà la *home directory*. Entrando ad esempio nella directory `mytest/`, si avrà:

```
1 ~$ cd mytest
2 ~/mytest$ dirs
3 ~/mytest
```

Esempio 13.26: *Spostamento su altra directory e verifica che lo stack inizia ancora con la current directory*

il contenuto iniziale dello stack è mutato alla nuova current directory.

Ora è conveniente tornare alla directory precedente digitando `cd -`, poi tornare nuovamente alla directory `mytest/` ma tramite il comando `pushd`[8] come mostrato:

```
1 ~/mytest$ cd -
2 /home/ms
3 ~$ pushd mytest
4 ~/mytest ~
5 ~/mytest$
```

Esempio 13.27: *Spostamento su altra directory tramite `pushd`*

in cui il `pushd`:

- ha aggiunto la directory corrente allo stack;
- sposato la *current directory* al nuovo path;
- mostrato lo stack.

Si nota che lo stack ora si compone di due entry:

$\sim$ la directory da cui abbiamo lanciato il comando `pushd`

$\sim$`/mytest` la directory corrente.

In sintesi il comando `pushd` prima ha salvato la *current directory* nella parte iniziale dello stack, poi ha cambiato directory.
Lo stack mostrato è equivalente a quello ottenuto tramite il comando `dirs` senza parametri, ma è possibile una visualizzazione migliore dello stack aggiungendo l'opzione `-v` al comando `dirs` come mostrato:

```
1 ~/mytest$ dirs -v
2  0  ~/mytest
3  1  ~
```

Esempio 13.28: *Visualizzazione dello stack tramite `dirs -v`*

o potrebbe essere preferibile aggiungere anche l'opzione `-l` la quale specificatamente non utilizza il prefisso $\sim$ delle directory relative alla propria *home*:

```
1 ~/mytest$ dirs -v -l
2  0  /home/ms/mytest
3  1  /home/ms
```

Esempio 13.29: *Visualizzazione dello stack tramite `dirs -v -l`*

[7] per la sintassi completa, v. `dirs --help`
[8] per la sintassi completa, v. `pushd --help`

Come già scritto prima, la directory corrente risiede sempre alla posizione 0 dello stack, per cui cambierà nuovamente cambiando posizione:

```
1 ~/mytest$ cd test1
2 ~/mytest/test1$ dirs -v
3  0  ~/mytest/test1
4  1  ~
```

Esempio 13.30: *Visualizzazione dello stack tramite* `dirs -v`

Si possono aggiungere un po' di directory allo stack senza spostarsi in esse aggiungendo l'opzione `-n` al comando `pushd`:

```
1 ~/mytest/test1$ cd
2 ~$ dirs
3 ~ ~
4 ~$ pushd -n /var/log/
5 ~ /var/log/ ~
6 ~$ pushd -n /etc
7 ~ /etc/ /var/log/ ~
8 ~$ pushd -n /etc/default
9 ~ /etc/default /etc/ /var/log/ ~
10 ~$ pushd -n /usr/src/
11 ~ /usr/src/ /etc/default /etc/ /var/log/ ~
```

Esempio 13.31: *Aggiunta di directory allo stack tramite* `pushd -n`

in cui sono state aggiunte altre quattro directory allo stack. Per una miglior visualizzazione:

```
1 ~$ dirs -v
2  0  ~
3  1  /usr/src/
4  2  /etc/default
5  3  /etc/
6  4  /var/log/
7  5  ~
```

Esempio 13.32: *Visualizzazione stack tramite* `dirs -v`

Per eliminare directory dallo stack, si utilizza il comando popd[9].
Ad esempio, se si vuole eliminare dallo stack la directory `/etc/default` è sufficiente digitare:

```
1 ~$ popd +2
2 ~$ dirs -v
3  0  ~
4  1  /usr/src/
5  2  /etc/
6  3  /var/log/
7  4  ~
```

Esempio 13.33: *Eliminazione dallo stack di* `/etc/defaults`

in cui `+2` era la posizione della directory rimossa dallo stack, le successive sono scalate di una posizione.

Successivamente all'inserimento delle directory nello stack, per spostarsi sarà sufficiente digitare il comando `cd ~n` in cui `n` è il valore indicato, quindi ad esempio digitando:

[9] per la sintassi completa, v. `popd --help`

```
1 ~$ cd ~2
2 /etc$ dirs -v
3  0  /etc
4  1  /usr/src/
5  2  /etc/
6  3  /var/log/
7  4  ~
```

Esempio 13.34: *Spostamento tramite lo stack*

Effettuata la panoramica di funzionamento, di seguito sono elencati ulteriore esempi sintattici:

`pushd .`
: aggiungerà la directory corrente nello stack alla posizione 1;

`pushd directory`
: si sposterà nella directory `directory` la quale diventerà l'elemento 0 dello stack aggiungendolo a tutti gli elementi precedenti (che quindi scalano di una posizione);

`pushd`
: senza argomenti si sposterà sull'elemento dello stack con posizione 1 (quindi effettua lo swap degli elementi 0 e 1 dello stack);

`pushd +N`
: eseguirà una rotazione dello stack in maniera che l'*N-esimo* sia a 0[10];

`pushd -N`
: eseguirà una rotazione dello stack in maniera che l'*N-esimo* contando dalla fine sia a 0[11];

`popd`
: senza argomenti, si sposterà nella directory di indice 1 trasformandolo in elemento 0 dello stack;

`popd -n`
: non cambierà la directory corrente e cancella l'elemento di indice 1 dello stack.

`popd +N`
: rimuoverà l'elemento *N-esimo*[12]

`popd -N`
: rimuoverà l'elemento *N-esimo* partendo dall'ultimo[13]

[10] sposterà l'elemento *N-esimo* a 0, poi i successivi sino all'ultimo, al termine inizierà da quello che era 0 fino ad arrivare al *(N-1)-esimo*

[11] sposterà l'elemento *N-esimo* dalla fine a 0, poi i successivi sino all'ultimo, al termine inizierà da quello che era 0 fino ad arrivare al *(N-1)-esimo* sempre da sinistra

[12] ad esempio `popd +0` rimuoverà l'elemento 0, `popd +1` rimuoverà l'elemento 1, ...

[13] ad esempio `popd -0` rimuoverà l'ultimo elemento, `popd -1` rimuoverà il penultimo elemento, ...

Approfondimento^P Per concludere l'argomento dello *stack di directory*, è utile comprenderne la struttura dati per la memorizzazione. La memorizzazione avviene nella variabile d'ambiente `DIRSTACK` che è un array[14] di stringhe, quindi i comandi di questa sezione permettono di gestire (visualizzare, aggiungere, rimuovere, ruotare) l'array.
Per visualizzare la variabile è sufficiente digitare:

```
1 $ echo ${DIRSTACK[@]}
2 ...
```

Esempio 13.35: *Variabile d'ambiente* `DIRSTACK`

il quale restituisce lo stesso output del comando `dirs` senza parametri.
Ogni elemento dell'array è accessibile tramite il rispettivo indice, quindi:

```
1 $ echo ${DIRSTACK[0]}
2 ...
3 $ echo ${DIRSTACK[1]}
4 ...
5 $ echo ${DIRSTACK[${#DIRSTACK[@]}-1]}
6 ...
```

Esempio 13.36: *Variabile d'ambiente* `DIRSTACK`

in cuil'ultimo comando mostrerà l'ultima directory memorizzato nell'array.

13.6 Impostazione all'avvio

Se la shell associata ad un utente è *bash*, quando si collegherà al sistema saranno eseguiti diversi script:

`/etc/profile`
: file di sistema;

`.bash_profile` o `.bash_login` o `.profile`,
: solo il primo che sarà disponibile nella sequenza indicata, tutti nella home dell'utente;

`.bashrc`
: nella home dell'utente.

Se è caricata una shell interattiva ma non di login, sarà caricato solo quest'ultimo file. Esso contiene una moltitudine di configurazioni quali *completamento automatico*, *history*, *alias* e altro. L'utente potrà modificare i file nella propria home per personalizzare il proprio ambiente di lavoro.
Esaminando il file, le impostazioni più interessanti sono:

`HISTCONTROL`
: per non duplicare le linee nell'*history* e non inserire le linee che iniziano con uno *spazio* nel medesimo file;

`shopt -s histappend`
: per settare la modalità di scrittura sull'*history file* (quindi `.history_bash`), impostata su *append* e non *overwrite*;

[14] gli array saranno affrontati in 22.6 a pag. 339

`HISTSIZE=1000` e `HISTFILESIZE=2000`
: v. 13.4;

`alias`
: la maggior parte impostati per il comando `ls` e `grep`;

`PS1`
: per impostare il prompt dei comandi della forma interessata[15].

13.7 `source`

Il comando `source` è un comando interno di shell utilizzato per eseguire comandi da file o script per modificare l'environment corrente. Equivalentemente si utilizza il carattere `.` e poi uno spazio per la lettura di uno script. Spesso lo si trova ripetuto nel file `.bashrc` esaminato in 13.6 proprio per impostare l'environment dell'utente. Per esempio:

```
1 $ grep source ~/.bashrc | grep -v '^#'
2 source ~/perl5/perlbrew/etc/bashrc
```

Esempio 13.37: *Il comando `source` utilizzato per modificare aspetti dell'environment dell'utente*

utilizzato da `perlbrew`, un *Perl environment manager*.

13.8 Approfondimenti

13.8.1 Stampa variabili d'ambiente /2^P

In 11.2.1 a pag. 125 si e già stampato l'environment definito dalle variabili d'ambiente. Rimanendo lo script 11.7 immutato ma ponendo in un'unica linea sia assegnazione di varibili sia esecuzione dello script:

```
1  $ PWD=3 SHELL=2 USER=1 perl environment_variables.pl
2  HOME => /home/ms
3  LANG => it_IT.UTF-8
4  ...
5  LOGNAME => ms
6  PATH => /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
       games:/usr/local/games:/snap/bin
7  PWD => 3
8  SHELL => 2
9  TERM => xterm-256color
10 USER => 1
11 ...
12 welcome => Hello world!!
```

Esempio 13.38: *Output dello script dell'***Esempio 11.7 che non mosrtrerà quanto indicato nella stessa sezione**

si noterà che per il software le variabili assegnate hanno sovrascritto quelle dell'environment di sistema[16], e solo al termine saranno scordate le assegnazioni e per il successivo comando sarà tornato l'environment preassegnazioni.

[15] spesso si personalizza questo, il default `'\u@\h:\w\$ '` indica l'*utente*, il carattere `@`, l'*host*, il carattere `:`, la *working directory*, il carattere `$` e uno spazio

[16] indicate in 11.8

13.9 Concetti chiave

alias: creazione di un nome di comando che la shell sostituirà con un comando (anche omonimo) con eventuali parametri aggiuntivi

`.bashrc`: se la shell associata all'utente è *bash*, qui sarà la configurazione caricata quando si collegherà al sistema

`.bash_history`: è memorizzata la cronologia dei comandi

directory stack: lo stack delle directory

`exit code`: codice numerico di uscita (se 0 è tutto ok ☺)

environment del sistema: l'environment definito dalle variabili di ambiente

environment del comando: eredita l'environment del sistema dal quale discende, con eventuale aggiunta di variabili se assegnate nella stessa linea di comando

child: processo che è stato generato da un *parent*

parent: processo che genererà uno o più *child*.

13.10 Comandi introdotti nel capitolo

Ctrl + r ⇒ richiama e permette la navigazione della cronologia dei comandi

`&&` ⇒ separatore tra comandi con valore *and*

`$?` ⇒ ispeziona l'*exit code* dell'ultimo comando

`alias` ⇒ imposta un *alias*

`dirs` ⇒ mostra lo stack delle directory

`history` ⇒ mostra la cronologia dei comandi

`ncal` ⇒ mostra un calendario

`pushd` ⇒ aggiunge una directory allo stack delle directory

`popd` ⇒ elimina una directory dallo stack delle directory

`source` ⇒ esegue comandi da un file nella shell corrente

`unalias` ⇒ cancella un *alias*

Capitolo 14

Altri comandi utili

QUESTO capitolo ha l'obiettivo di aggiungere ulteriori comandi utili nella *cassetta degli attrezzi*. L'ordine con cui gli argomenti sono presentati è rigorosamente a mio giudizio.

14.1 watch

Il comando `watch` esegue comandi a intervalli regolari. Di default ad intervalli di due secondi cancella il contenuto della console e mostrerà l'output del comando un'intestazione con data e orario di sistema. Per terminare l'update periodico, si eseguirà la sequenza Ctrl + c .
In ***Tabella 14.1*** sono presenti le principali opzioni. Alcuni comandi possono essere:

OPZIONI		DESCRIZIONE
BREVE	ESTESA	
`-n`	`--interval`	specifica l'intervallo temporale tra due update
`-d`	`--differences`	evidenzia le differenze tra due update successivi
`-t`	`--no-title`	non mostra l'intestazione

Tabella 14.1: *Opzioni principali per il comando* `watch`

`watch -n 5 date`
: mostra la data e l'orario di sistema ogni 5 secondi (v. ***Figura 14.1***);

`watch -d date`
: mostra la data e l'orario di sistema ogni 2 secondi, evidenziando i caratteri che cambiano nella schermata di aggiornamento (v. ***Figura 14.2***);

`watch -t date`
: mostra la data e l'orario di sistema ogni due secondi senza inserisce l'intestazione (v. ***Figura 14.3***).

Altre opzioni e altri esempi nel manuale relativo[1].

[1] man watch

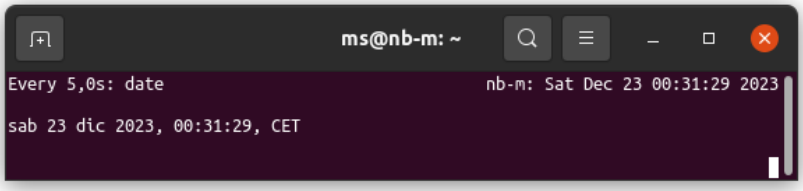


Figura 14.1: *Esempio di comando* `watch -n 5`*, la console si aggiornerà ogni 5 secondi*

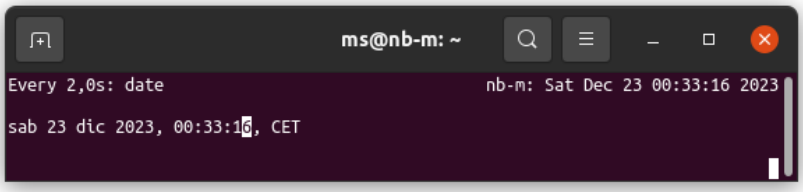


Figura 14.2: *Esempio di comando* `watch -d`

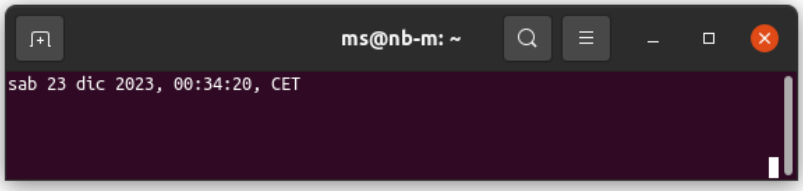


Figura 14.3: *Esempio di comando* `watch -t`

14.2 xargs

Il comando `xargs` è già stato utilizzato negli esempi di 4.9.2 a pag. 54.
Esso è utile per creare una pipeline per l'esecuzione dello `stdin`: alcuni strumenti riescono ad elaborare nativamente lo `stdin`[2], altri come `echo`, `mkdir`, `rmdir`, ... non elaborano questo flusso.
L'utilizzo è molto semplice, ad esempio si vogliano creare le directory ricevute allo `stdin`:

```
1 $ echo '1 2 3'| xargs mkdir
2 $ ls -dF 1 2 3
3 1/  2/  3/
```

Esempio 14.1: *Esempio creazione di directory tramite* xargs

Come è mostrato dall'elenco, sono state create le 3 directory.
Spesso è utilizzata in pipeline con `find` per poi passare l'output al comando `rm`, ad esempio:

[2] `cat`, `grep`, `sort`, ...; genericamente le utility di *filtro* (v. capitolo 9 a pag. 103

```
1 $ find /tmp -name core -type f -print | xargs /bin/rm -f
```

Esempio 14.2: *Esempio rimozione di file tramite* xargs

in cui si cercano i file con nome `core` nella directory `/tmp`, l'output è passato a `rm` che provvederà alla cancellazione.
Questa linea non si comporterà correttamente qualora nei file ci fossero ad esempio spazi, per cui è migliore questa espressione:

```
1 $ find /tmp -name core -type f -print0 | xargs -0 /bin/rm -f
```

Esempio 14.3: *Esempio rimozione di file tramite* xargs -0

in cui si sono attivate l'opzione `-print0` per il `find` e `-0` per `xargs` per un eventuale gestione corretta di spazi o *newline*.

Per la cancellazione si potrebbe optare per l'utilizzo dell'`-exec`:

```
1 $ find /tmp -name core -type f -exec rm  \;
```

Esempio 14.4: *Esempio rimozione di file tramite* find -exec

ma si preferisce utilizzare la modalità con `xargs` per una questione di efficienza, ma sarebbe ancor più efficiente:

```
1 $ find /tmp -name core -type f -delete
```

Esempio 14.5: *Esempio rimozione di file tramite* xargs -0

poiché eviterebbe l'uso implicito di *fork* ed *exec*[3].

Se volessimo la lista degli utenti di sistema presentata in maniera compatta[4], potremmo utilizzare la pipe con `echo`;

```
1 $ cut -d:  -f1 < /etc/passwd | sort | xargs echo
2 _apt bin games ... ms ...
```

Esempio 14.6: *Elenco compatto di utenti di sistema*

14.3 sleep

Il comando `sleep` attende per il tempo specificato. La sintassi generale è la seguente:

```
1 $ sleep NUMBER[SUFFIX]
```

Esempio 14.7: *Sintassi generica del comando* `sleep`

in cui:

`sleep` è il comando in oggetto;

`NUMBER` è la durata di attesa del comando, è utilizzabile un numero decimale;

[3] argomento trattato in **??** a pag. **??**

[4] se la pipe si fosse fermata a `sort`, avremmo una lista di utenze separate da *newline*

`SUFFIX` è l'eventuale suffisso da aggiungere in accordo con la ***Tabella 14.2*** (se non specificato, il default è l'unità di misura del secondo).

Ad esempio:

```
1 $ sleep 1
2 $ sleep 1.5
3 $ sleep 0.1m
```

Esempio 14.8: *Sequenza di comandi* `sleep`

in cui:

`sleep 1` attenderà un secondo

`sleep 1.5s` attenderà 1.5 secondi;

`sleep 0.1m` 0.1 minuti (6 secondi).

I suffissi che si possono utilizzare sono riportati in ***Tabella 14.2***.

SUFFISSO	DESCRIZIONE
s	secondi (default)
m	minuti
h	ore
d	day

Tabella 14.2: *Suffissi utilizzabili per il comando* `sleep`

14.4 `time`

Il comando `time` esegue programmi e mostra le risorse del sistema utilizzate. Ad esempio:

```
1 $ time sleep 6
2
3 real  0m6,003s
4 user  0m0,000s
5 sys 0m0,002s
```

Esempio 14.9: *Il comando* `time` *con il software* `sleep`

Ovviamente è utile utilizzarlo in altri contesti, in questo èstato utile a dimostrare che lo `sleep` funzioni correttamente.

14.5 `timeout`

Il comando `timeout` esegue un comando impostando un limite temporale. La sintassi generale è:

```
1 $ timeout [OPTION] DURATION COMMAND [ARG]...
```

Esempio 14.10: *Sintassi generica del comando* `timeout`

in cui:

`timeout` è il comando in oggetto;

`OPTION` sono le eventuali opzioni al comando;

`DURATION` è l'intervallo limite del comando che segue secondo lo standard utilizzabile per il comando `sleep`[5]

`COMMAND` è il comando che si desidera eseguire a cui si applicherà il timeout qualora sia necessario;

`[ARG]...` sono gli argomenti del comando successivo e altro.

Ad esempio:

```
1 $ timeout 2 sleep 2.5
2 $ echo $?
3 124
4 $ timeout 2.1 sleep 2
5 $ echo $?
6 0
```

Esempio 14.11: *Esempio del comando* `timeout` *con il software* `sleep`

in cui:

`timeout 2 sleep 2.5` ha stabilito un timeout di 2 secondi per uno `sleep` di 2.5 secondi, quindi agirà;

`timeout 2.1 sleep 2` ha stabilito un timeout di 2.1 secondi per uno `sleep` di 2 secondi, quindi non agirà.

Per verificare che abbia agito, si è interrogata la variabile `$?` che riporta l'*exit status* associato, in particolare:

124 è l'*exit status* che riporta quando ha agito;

0 è l'*exit status* del comando `sleep` che ha terminato senza problemi.

Qualora si volesse l'*exit status* del comando che si è lanciato indipendentemente dall'eventuale azione del software *timeout*, si può attivare l'opzione `--preserve-status` come nel seguente esempio:

```
1 $ timeout -preserve-status 2 sleep 2.5
2 $ echo $?
3 143
```

Esempio 14.12: *Esempio del comando* `timeout` *con l'opzione* `--preserve-status`

in cui l'*exit status* è restituito dal comando `sleep`.

Per verifica ulteriore che la soglia stabilita dal comando *timeout* sia affidabile, è possibile eseguire quanto di seguito:

[5] quindi è utilizzabile un numero decimale con i suffissi in ***Tabella 14.2*** (anche qui, se non specificato, il default è l'unità del secondo)

```
1 $ time timeout 2 sleep 2.5
2
3 real  0m2,003s
4 user  0m0,002s
5 sys 0m0,002s
6 $ echo $?
7 124
```

Esempio 14.13: *Esempio del comando* `time` *che esegue* `timeout` *il quale esegue* `sleep`

in cui è mostrato che effettivamente la soglia stabilita da `timeout` è stata rispettata, con l'*exit status* restituito dallo stesso comando.

14.6 Background e foreground

Inviare un comando in *background* equivale ad eseguire il task medesimo *dietro le scene*[6], quindi senza un successivo intervento dell'utente. Se l'utente vorrà inviare il job in background, all'esecuzione del comando aggiungerà il simbolo `&` come nell'esempio:

```
1 $ comando &
2 [SPEC_JOB] PID
```

Esempio 14.14: *Invio di un generico comando in background*

in cui:

`SPEC_ID` o *job number* è un id numerico assegnato al job, identificativo relativamente alla nostra shell;

`PID` è l'acronimo di ***process ID***[7], identificativo valido a livello di sistema.

Quindi il sistema si occuperà di eseguire il *job* mentre, dopo la comunicazione di questi parametri, il prompt della shell tornerà disponibile per un nuovo comando, eventualmente inviandolo in background. Ad esempio:

```
1 $ sleep 10 &
2 [1] 809753
3 $ sleep 5 &
4 [2] 809754
```

Esempio 14.15: *Invio di due* `sleep` *in background*

in cui `[1] 809753` e `[2] 809754` sono i parametri assegnati ai rispettivi comandi. Aspettando un paio di secondi:

```
1 $ sleep 15 &
2 [3] 809755
3 [1]   Completato              sleep 10
4 [2]   Completato              sleep 5
```

Esempio 14.16: *Invio di un nuovo* `sleep` *in background*

in cui:

[6] maggiori dettagli in **??** a pag. **??**

[7] argomentato in 16.5 a pag. 16.5

`[3] 809755` sono i parametri assegnati al nuovo comando;

`[1] Completato` e

`[2] Completato` è la comunicazione del completamento dei job precedenti.

Possiamo continuare a lavorare e sarà la shell che comunicherà in maniera asincrona il completamento dei job in coda, ad esempio per quest'ultimo, attendendo 15 secondi, inviando semplicemente Enter alla shell, si ottiene:

```
1 $ Enter
2 [3]+  Completato              sleep 15
3 $
```

Esempio 14.17: *Attesa di job in background*

e dal simbolo + si ha comunicazione che il buffer dei job in coda si è svuotato poiché l'ultimo è stato `Completato`. Quindi inviando nuovamente un job, il numero relativo `SPEC_ID` ripartirà da `1`.

Possiamo avere informazioni relativamente ai job in coda tramite il comando `jobs`, ad esempio:

```
1 $ jobs
2 $ sleep 10 &
3 [1] 810857
4 $ sleep 2 &
5 [2] 810858
6 $ jobs
7 [1]-  In esecuzione           sleep 10 &
8 [2]+  Completato              sleep 2
9 $
```

Esempio 14.18: *Check dei job in coda*

È possibile attendere esplicitamente il completamento di un job in coda inviando un esplicito comando `wait %SPEC_ID`, o tutti i job senza parametro aggiuntivo, ad esempio:

```
1 $ sleep 10 &
2 [1] 811249
3 $ sleep 5 &
4 [2] 811254
5 $ wait
6 [1]-  Completato              sleep 10
7 [2]+  Completato              sleep 5
8 $
```

Esempio 14.19: *Wait dei job in coda*

In ultimo, possiamo recuperare in job in background spostandolo in foreground utilizzando il comando `fg`, ad esempio:

```
1 $ sleep 10 &
2 [1] 811646
3 $ sleep 5 &
4 [2] 811662
5 $ fg
6 sleep 5
7 $
```

Esempio 14.20: *Spostamento in foreground del job*

oppure si può riportare in foregroud un job in particolare sempre tramite il suo `%SPEC_ID` utilizzandolo come parametro del comando `fg`.

14.7 bc

Il comando `bc` è un software che supporta il linguaggio *bc* per la trattazione numerica tramite esecuzione interattiva degli statement forniti. La potenza del linguaggio è nella trattazione numerica con precisione arbitraria.
Senza entrare nello specifico[8], di seguito alcuni esempi per intuirne la potenza:

```
1  $ echo "10+5" | bc
2  15
3  $ x=`echo "10+5" | bc`; echo $x
4  15
5  $ x=`echo "var=10;var^=3;var" | bc`; echo $x
6  1000
7  $ x=`echo "var=51;var%=7;var" | bc`; echo $x
8  2
9  $ echo "var=10;++var" | bc
10 11
11 $ echo "var=10;var--" | bc
12 10
13 $ echo "3>2" | bc
14 1
15 $ echo "2 && 3" | bc
16 1
17 $ echo "0 || 1" | bc
18 1
19 $ echo "!  0" | bc
20 1
```

Esempio 14.21: *Esempi di* bc

ma ci si può spingere molto oltre utilizzando la libreria matematica integrata attraverso l'opzione `-l`:

```
1  $ pi=`echo "scale=20; 4*a(1)" | bc -l`; echo $pi
2  3.14159265358979323844
3  $ echo "c($pi/3)" | bc -l
4  .50000000000000000001
5  $ echo "s($pi/6)" | bc -l
6  .49999999999999999999
7  $ echo "e(1)" | bc -l
8  2.71828182845904523536
9  $ echo "l(e(1))" | bc -l
10 .99999999999999999999
11 $ echo "obase=2;18" | bc -l
12 10010
13 $ echo "obase=16;31" | bc -l
14 1F
```

Esempio 14.22: *Sintassi di un* bc

in cui:

`a(x)` definisce l'*arcotangente* di *x* (restituito in radianti), `scale` indica il numero di cifre decimali da calcolare (per cui sono mostrate 20 cifre decimali del π);

[8] il linguaggio è espressivamente simile al linguaggio *C*, v. `man bc`

`c(x)` per il *coseno* di *x*, *x* in radianti;

`s(x)` per il *seno* di *x*, *x* in radianti;

`e(x)` per e^x;

`l(x)` per $ln(x)$;

`obase=2,n` per esprimere in binario *n* (default decimale);

`obase=16,n` per esprimere in esadecimale *n*;

Il software può essere anche eseguito in maniera interattiva:

```
1 $ bc
2 bc 1.07.1
3 Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free
      Software Foundation, Inc.
4 This is free software with ABSOLUTELY NO WARRANTY.
5 For details type `warranty'.
6 3^4
7 81
```

Esempio 14.23: *Sintassi di un* bc

14.8 convmv

In 4.1.1 a pag, 31 nel paragrafo **Caratteri speciali** si era menzionato il software `convmv`: è un programma Perl che converte nomi di file e directory tra diverse codifiche di caratteri. Per ottenere la lista dei nomi dei set di caratteri utilizzabili:

```
1 $ convmv --list
2 7bit-jis
3 AdobeStandardEncoding
4 AdobeSymbol
5 AdobeZdingbat
6 ascii
7 ascii-ctrl
8 big5-eten
9 big5-hkscs
10 cp1006
11 ...
12 UTF-32
13 UTF-32BE
14 UTF-32LE
15 UTF-7
16 utf-8-strict
17 utf8
18 viscii
19 $ convmv -list | wc -l
20 124
```

Esempio 14.24: *Elenco* set di caratteri *utilizzabili*

quindi sul mio sistema ben 124 set di caratteri disponibili.
Per procedere alla conversione:

```
1 $ convmv -f iso-8859-1 -t utf8 DIRECTORY
2 ...
3 $ convmv -f euc-jp -t utf8 DIRECTORY
```

```
4 ...
```

Esempio 14.25: *Esempi di conversione da un* set di caratteri *ad un altro in* test mode

in cui:

-f iso-8859-1 attiva la conversione da *ISO8859-1*[9] dei filename;

-f euc-jp attiva la conversione da *EUCJP* dei filename;

entrambi alla codifica *UTF-8* della directory *DIRECTORY*.
È anche possibile convertire la codifica dell'intero filesystem in *UTF-8*:

```
1 $ convmv -f iso-8859-7 -t utf8 -r /home
2 ...
```

Esempio 14.26: *Esempi di conversione da un* set di caratteri *ad un altro in* test mode *dell'intera* `/home`

in cui tutta la `/home` sarà riconvertita ricorsivamente.
Tutti i test precedenti saranno effettuati in *test mode* perché è tale il comportamento di default del software. Se quanto mostrato risulta coerente con l'obiettivo, la conversione potrà avvenire attivando l'opzione `--notest`.

14.9 pr

Il comando `pr` converte file di testo per la stampante. Ad esempio:

```
1 $ cut -d:  -f1 /etc/passwd | sort | pr -2 >/tmp/userlist.txt
```

Esempio 14.27: *Elenco degli utenti di sistema su due colonne*

invierà l'elenco degli utenti a `sort` che li ordinerà lessicograficamente il quale invierà l'output al comando `pr` che lo stamperà su due colonne inviandolo a `/tmp/userlist.txt`.

14.10 whereis

Il comando `whereis` è utilizzato per localizzare sorgenti, binari e manuali di un comando. Ad esempio:

```
1 $ whereis md5sum
2 md5sum: /usr/bin/md5sum.textutils /usr/bin/md5sum /usr/share/man/man1/
      md5sum.1.gz
3 $ whereis passwd
4 passwd: /usr/bin/passwd /etc/passwd /usr/share/man/man1/passwd.1.gz /usr/
      share/man/man1/passwd.1ssl.gz /usr/share/man/man5/passwd.5.gz
```

Esempio 14.28: *Esempi del comando* `whereis`

[9] legacy Western European

14.11 locale

Senza parametri il comando `locale` mostra le variabili di environment che definiscono il *locale*, come ad esempio la lingua, la nazione, l'encoding utilizzato, ..., informazioni condivise sia dalle librerie e sia applicazioni *locale-aware*.
Ad esempio:

```
1  $ locale
2  LANG=it_IT.UTF-8
3  LANGUAGE=
4  LC_CTYPE="it_IT.UTF-8"
5  LC_NUMERIC="it_IT.UTF-8"
6  LC_TIME="it_IT.UTF-8"
7  LC_COLLATE="it_IT.UTF-8"
8  LC_MONETARY="it_IT.UTF-8"
9  LC_MESSAGES="it_IT.UTF-8"
10 LC_PAPER="it_IT.UTF-8"
11 LC_NAME="it_IT.UTF-8"
12 LC_ADDRESS="it_IT.UTF-8"
13 LC_TELEPHONE="it_IT.UTF-8"
14 LC_MEASUREMENT="it_IT.UTF-8"
15 LC_IDENTIFICATION="it_IT.UTF-8"
16 LC_ALL=
```

Esempio 14.29: *Esempi del comando* `locale`

Tali impostazioni hanno effetto, sul formata di data e orario, primo giorno della settimana, valuta e altri valori da formattare in relazione alle impostazioni di lingua/region/country.
Per ulteriori si consulti `man locale`.

14.12 tr

Il comando `tr` esegue il translate o la cancellazione di caratteri ricevuti dallo `stdin` e li mostra sullo `stdout`.
Ad esempio,

```
1 $ echo 'Hello world!!'  | tr Ho ha
2 hella warld!!
```

Esempio 14.30: *Esempio di comando* `tr`

in cui sono stati sostituiti i caratteri `H` e `o` rispettivamente con `h` e `a`.
La sintassi generale è la seguente:

```
1 $ tr [OPTION] SET1 [SET2]
```

Esempio 14.31: *Sintassi del comando* `tr`

in cui le opzioni principali sono sintetizzate in ***Tabella 14.3***. Nel caso non sia data l'opzione `-d` per la cancellazione, entrambi i `SET` devono essere digitati.
I vari `SET` sono stringhe di caratteri costituite da sequenze la cui interpretazione è riportata in ***Tabella 14.4***.

SEQUENZA	INTERPRETAZIONE

\NNN	caratteri con valore ottale NNN (da 1 a 3 cifre ottali)
\\	
\a	*BEL* sonoro
\b	*backspace*
\f	*form feed*
\n	*newline*
\r	*return*
\t	*tab* orizzontale
\v	*tab* verticale
[CHAR1-CHAR2]	tutti i caratteri da CHAR1 a CHAR2 in ordine ascendente
[CHAR*]	inSET2, copie del carattere CHAR fino ad arrivare alla lunghezza diSET1
[CHAR*REPEAT]	REPEAT copied CHAR (REPEAT ottale se inizia con 0)
[:alnum:]	qualsiasi lettera o cifra
[:alpha:]	qualsiasi lettera
[:blank:]	qualsiasi spazio orizzontale
[:cntrl:]	qualsiasi carattere di controllo
[:digit:]	qualsiasi cifra
[:graph:]	qualsiasi carattere stampabile ad esclusione dello spazio
[:lower:]	qualsiasi carattere minuscolo
[:print:]	qualsiasi carattere stampabile (incluso lo spazio)
[:punct:]	qualsiasi carattere di punteggiatura
[:space:]	qualsiasi carattere di spaziatura orizzontale o verticale
[:upper:]	qualsiasi carattere maiuscolo
[:xdigit:]	qualsiasi carattere esadecimale
[=CHAR=]	qualsiasi carattere equivalente a CHAR

Tabella 14.4: *Interpretazione delle sequenze nei* `SET`

Di seguito alcuni esempi.

Esempio 1

Trasforma le parentesi graffe in tonde.

```
1 $ echo '{}' | tr '{}' '()'
2 ()
```

Esempio 14.32: `tr` *di* `{}` *in* `()`

Esempio 2

Trasforma le parentesi graffe in quadre.

```
1 $ echo '{}' | tr '{}' '[]'
2 []
```

Esempio 14.33: `tr` *di* `{}` *in* `[]`

Per una maggior portabilità, la sequenza del comando `tr` si sarebbe dovuta scrivere come segue:

OPZIONI		DESCRIZIONE
BREVE	ESTESA	
-c, -C	--complement	usa il complemento di SET1
-d	--delete	evidenzia le differenze tra due update successivi
-s	--squeeze-repeats	sostituisce ogni sequenza di un carattere ripetuto elencato nell'ultimo SET con una singola occorrenza del carattere
-t	--truncate-set1	troncamento del SET1alla lunghezza di SET2

Tabella 14.3: *Opzioni principali per il comando* `tr`

```
1 $ echo '{}' | tr '{}' '\[]'
2 []
```

Esempio 14.34: `tr` *di* `{}` *in* `[]` *corretta*

quindi si sarebbe dovuto far precedere il metacarattere \ al carattere [per esplicitare che, nel contesto in oggetto, quest'ultimo non è un metacarattere.

Esempio 3

Trasformare i caratteri maiuscoli in minuscoli.

Esempio 3a Di seguito la modalità con range di caratteri.

```
1 $ echo 'Hello WORLD!!'  | tr A-Z a-z
2 hello world!!
```

Esempio 14.35: `tr` *per sostituzione di maiuscole in minuscole tramite range*

Esempio 3b Di seguito la modalità con classi di carattere.

```
1 $ echo 'Hello WORLD!!'  | tr [:upper:]  [:lower:]
2 hello world!!
```

Esempio 14.36: `tr` *per sostituzione di maiuscole in minuscole tramite classe*

Esempio 4

Cancellare sequenze numeriche.

```
1 $ echo 'Numero segreto:  123456' | tr -d [:digit:]
2 Numero segreto:
```

Esempio 14.37: `tr` *per cancellare sequenze di numeri*

Esempio 5

Mostrare solo i numeri.

```
1 $ echo 'Numero segreto:  123456' | tr -cd [:digit:]
2 123456
```

Esempio 14.38: *`tr` per cancellare sequenze non numeriche*

Esempio 6 Di seguito più sequenze:

```
1 $ tr -d '\0' < src_file > dest_file
2 $ tr -s '\n' < src_file > dest_file
3 $ tr -c '[:print:][:ctrl:]'  '[?*]' < src_file > dest_file
```

Esempio 14.39: *Altri esempi di `tr`*

in cui:

`tr -d '\0'`
: cancella tutte le occorrenze di `'\0'`[10];

`tr -s '\n'`
: sostituisce ogni sequenza di uno o più *newline* con un singolo *newline*;

`tr -c '[:print:][:ctrl:]'  '[?*]`
: sostituisce ciascun carattere non stampabile (che non siano caratteri di controllo validi) con il carattere `?`;

tutte dal file sorgente `src_file` al file destinazione `dest_file`.

14.13 figlet

Mostra i caratteri ingranditi secondo uno stile personalizzabile sul terminale. Se eseguito senza argomenti, aspetta l'immissione da terminale:

```
1 $ figlet
2 a
3
4   __ _
5  / _` |
6 | (_| |
7  \__,_|
8
9 Hello
10  _   _       _ _
11 | | | | ___| | | ___
12 | |_| |/ _ \ | |/ _ \
13 |  _  |  __/ | | (_) |
14 |_| |_|\___|_|_|\___/
```

Esempio 14.40: *Semplici esempi di `figlet`*

e il solito Ctrl + d per terminare.

È possibile utilizzare un font personalizzato con l'opzione `-f` e il nome del font, ad esempio:

[10] NULL character

```
$ figlet -f script
a



 __,
/  |
\_/|_/


B
 , __
/|/  \
 | __/
 |   \
 |(__/
```

Esempio 14.41: *Esempio di* `figlet -f`

oppure centrare la scritta con l'opzione `-c`:

```
$ figlet -c
Ciao
                                      ____ _
                                     / ___(_) __ _  ___
                                    | |   | |/ _` |/ _ \
                                    | |___| | (_| | (_) |
                                     \____|_|\__,_|\___/
```

Esempio 14.42: *Esempio di* `figlet -c`

oppure aggiungere dello spazio tra i *FIGcaratteri* utilizzati con l'opzione `-k`:

```
$ figlet -kc
Ciao
                                     ____  _
                                    / ___|(_)  __ _   ___
                                   | |     | | / _` | / _ \
                                   | |___  | || (_| || (_) |
                                    \____||_| \__,_| \___/
```

Esempio 14.43: *Esempio di* `figlet -c`

Ci sono tante altre opzioni utili come:

`-p` per attivare il *paragraph mode* il quale elimina dall'input i *newline* non necessari (utile nella lettura di testi con `figlet -p < testo.txt`);

per cui è utile ricorrere al manuale[11].

Per concludere, un'opzione simpatica potrebbe essere la seguente:

```
$ date +%T|figlet -kc
                    _   ___      ____   ___       _  _      ___
                   / | ( _ ) _ | ___| / _ \  _  | || |    ( _ )
                   | | / _ \(_)|___ \| (_) |(_)| || |_  / _ \
                   | || (_) |_  ___) |\__, | _ |__   _|| (_) |
                   |_| \___/(_)|____/   /_/ (_)    |_|   \___/
```

Esempio 14.44: *Esempio di* `figlet -kc` *il cui* `stdin` *proviene dal comando* `date`

[11] `man figlet` ricca di esempi

e per ultimo il comando:

```
1 $ watch -t -n1 "date +%T|figlet -kc"
```

Esempio 14.45: *Esempio di* `figlet -kc` *il cui* `stdin` *proviene dal comando* `date` *eseguito con* `watch`

il cui effetto non è reso da tale mezzo ma andrà eseguito da terminale.

14.14 Concetti chiave

background: in esecuzione *dietro le quinte*

foreground: è l'applicativo che l'utente sta eseguedo

label: arbitraria etichetta, soltitamente è una sequenza alfanumerica

locale: si intende la localizzazione dell'ambiente tramite variabili di environment

locale aware: sono le applicazioni che gestiscono la localizzazione, quindi si presentano differentemente in accordo a tali variabili

set di caratteri: il *charset* è un codifica che associa ad un insieme di grafemi (solitemente caratteri) ad un insieme di numeri

14.15 Comandi introdotti nel capitolo

`&` ⇒ `metacarattere per l'invio di un comando in background`
`bc` ⇒ `software che supporta linguaggio di calcoli per precisione arbitraria`
`convmv` ⇒ `converte nomi di file e directory tra diverse codifiche di caratteri`
`fg` ⇒ `sposta unjob in primo piano`
`figlet` ⇒ `mostra i caratteri ingranditi secondo uno stile personalizzabile`
`jobs` ⇒ `visualizza lo stato dei jobs`
`sleep` ⇒ `attende per il tempo` *n* `specificato (di default in secondi)`
`time` ⇒ `esegue programmi e mostra le risorse del sistema utilizzate`
`timeout` ⇒ `esegue un comando impostando un tempo limite`
`tr` ⇒ `esegue il translate di caratteri`
`wait` ⇒ `attende il completamento dei job`
`watch` ⇒ `esegue un programma periodicamente`
`whereis` ⇒ localizza sorgenti, binari e manuali del comando
`xargs` ⇒ permette l'esecuzione di linee di comando dallo `stdin`

Capitolo 15

Approfondimenti

QUESTO capitolo è dedicato ad approfondimenti per i quali non ho ritenuto opportuno una trattazione ravvicinata all'argomento omologo, per cui sono stati confinati in quest'ultimo segmento.

15.1 Procedura di *logout*

Per la procedura di *logout* da terminale, nella sezione 2.1.2 a pag. 16, si sono mostrate tre alternative analizzate di seguito.

15.1.1 logout

Digitando:

```
1 $ type logout
2 logout è un comando interno di shell
3 $ logout --help
4 logout: logout [n]
5     Esce da una shell di login.
6
7     Esce da una shell di login con stato di uscita N.
8     Restituisce un errore se non eseguito  in una shell di login.
```

Esempio 15.1: *Comandi* `type logout` *e* `logout --help`

la risposta è esplicativa, il comando ha esattamente la funzionalità omonima, per cui non c'è altro da aggiungere ad esclusione che, a differenza del prossimo comando, non lo si incontra in altri ambiti.

15.1.2 exit

Digitando:

```
1 $ type exit
2 exit è un comando interno di shell
3 $ exit --help
4 exit: exit [n]
5     Esce dalla shell.
6
7     Esce dalla shell con uno stato N. Se N è omesso lo stato
```

```
8     di uscita è quello dell'ultimo comando eseguito.
```

Esempio 15.2: *Comandi type exit e exit –help*

si nota una piccola differenza dal precedente, in particolare non si contestualizza il concetto di *login*. Ovviamente la generalità del comando la si nota digitando:

```
1 $ man exit
2 EXIT(3)                                Linux Programmer's Manual
3
4 NAME
5        exit - cause normal process termination
```

Esempio 15.3: *Comando man exit*

e infatti per un programmatore `exit` è la funzione di terminazione di un processo. Quindi la si può definire correttamente una chiamata universale, in altri termini la si può decontestualizzare dalla procedura di *login/logout* e, più in generale, dall'oggetto *shell*.

15.1.3 Ctrl + D

Per comprendere la sequenza Ctrl + D si può digitare:

```
1 $ stty -a
2 speed 38400 baud; rows 24; columns 80; line = 0;
3 intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
4 eol2 = <undef>; swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt =
      ^R;
5 werase = ^W; lnext = ^V; discard = ^O; min = 1; time = 0;
6 -parenb -parodd -cmspar cs8 -hupcl -cstopb cread -clocal -crtscts
7 -ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -
      ixoff
8 -iuclc -ixany -imaxbel iutf8
9 opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0
      vt0 ff0
10 isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -
      echoprt
11 echoctl echoke -flusho -extproc
```

Esempio 15.4: *stty -a*

in cui scorrendo nell'output si nota `eof = ^D`. Quindi la sequenza Ctrl + D (in *ASCII* e *Unicode* il carattere ha la codifica `U+0004 <control-0004>` definito nello stesso *Unicode* come *END OF TRANSMISSION*) è il valore di default assegnato al carattere di controllo `EOF`[1].
In generale il driver del terminale riceve ed elabora le sequenze dopo che sia stato premuto il carattere Enter, il quale invierà la linea immessa con il carattere di controllo `LF`[2], e il terminale procederà all'elaborazione della stessa ritornando il carattere di controllo `CR`[3].
Esistono però altri caratteri di controllo intercettati dal driver del terminale, quali ad esempio ⟵ oppure Ctrl + U, rispettivamente il primo per cancellare un carattere a sinistra, il secondo per cancellare l'intera linea e terminare con un `CR`.
Per inviare la linea immessa senza premere Enter, è possibile utilizzare il carattere di

[1] *end of file*
[2] line feed
[3] carriage return

controllo `EOF` attivandolo, per quanto appena scritto, attraverso la sequenza Ctrl + D . Se la linea non presenta nessun carattere immesso[4], allora il driver interpreterà il carattere di controllo come l'effettivo *end of file*[5].

[4] come già scritto, una linea immessa con termine Enter presenterà il carattere `LF`

[5] il nome deriva dal posizionamento dello stesso carattere di controllo all'interno di un file, in altri termini in un file non ci saranno caratteri successivi a questo, per cui esso identifica esattamente il carattere di *end of file*

15.2 Concetti chiave

carriage return: sia in ASCII sia Unicode è assegnato il valore 13_{10} (in esadecimale D_{16}), in molti linguaggi di programmazione e nelle regexp è assegnato la sequenza `\r`

end of file: v, **EOF**

EOF: ***e**nd **o**f **f**ile* indica il termine della sorgente (file o stream)

EOT: ***e**nd **o**f **t**rasmission*, nelle telecomunicazione il carattere di controllo indica la conclusione delle trasmissioni.

line feed: anche *newline*, *line ending*, ***e**nd **o**f **l**ine* (**EOL**), o *line break*, è un carattere di controllo codificato sia in ASCII sia Unicode con valore 10_{10} (in esadecimale A_{16}), in molti linguaggi di programmazione e nelle regexp è assegnata la sequenza `\n`

15.3 Comandi introdotti nel capitolo

Ctrl + D ⇒ è la sequenza che trasmette il carattere di controllo `EOF`

Ctrl + U ⇒ cancella l'intera linea e terminare con un `CR`

`exit` ⇒ causa la terminazione di un processo

`logout` ⇒ comando interno alla Bash per eseguire il *logout*

`stty` ⇒ cambia e, senza parametri, stampa i *line settings* del terminale

Parte II

Principiante

Capitolo 16

Introduzione all'amministrazione di sistema

FINORA il documento ha presentato alcuni concetti, strumenti e procedure valide per un qualunque utente. In questo capitolo (e diffusamente nel seguito) c'è necessità di procedere attraverso la figura dell'*amministratore di sistema* il quale ha il *privilegio* di eseguire diversi compiti alcuni dei quali sono:

- configurazione del computer;
- installare/aggiornare/rimuovere software;
- aggiungere (o rimuovere) utenti al sistema;
- backup/ripristino del filesystem;

Questi e altri task sono ovviamente *oneri* per l'amministratore, ma si è utilizzato il termine *privilegio* proprio per rimarcare il concetto di *acquisizione di privilegi* per poter svolgere determinati compiti.

Si è già accennato in 12.3.9 a pag. 140 e in 12.3.10 a pag. 141 che nei pseudofilesystem rispettivamente `/proc` e `/sys` sono contenute le informazioni dinamiche sui processi e sullo stato del sistema. Dalla sezione 16.5 si introdurranno alcuni comandi utili per la fruizione di tali informazioni.

16.1 Il superutente

Per il *system administrator* è designato un account speciale denominato **root** per il quale non saranno effettuati taluni check affinché gli siano permesse operazioni critiche nel sistema, operazioni vietate agli utenti ordinari. Ad esempio, da utente ordinario non si avrà accesso alla directory assegnata all'utente *root*:

```
1 ms@nb-m:~$ ls /root/
2 ls:  impossibile aprire la directory '/root/':  Permesso negato
```

Esempio 16.1: *`ls` di `/root` da utente ordinario*

ma l'utente `root` non riscontrerà il medesimo problema:

```
1 root@nb-m:~# ls ~
2 root@nb-m:~# ls -R /home/
3 ...
```

Esempio 16.2: *`ls` di `/root` e dell'albero `/home` da amministratore*

in cui l'amministratore non ha ovviamente riscontrato problemi nell'eseguire il comando `ls` della propria *home directory*[1], ma nemmeno delle *home degli utenti*.
Si è voluta riportare la forma estesa del prompt dei comandi[2], nel seguito sarà mostrato il carattere `$` per gli utenti ordinari, `#` per il superutente.

16.2 Agire responsabilmente

Non sarebbe necessario, ma è utile rimarcare due concetti relativamente alla concessione tecnica appena esaminata:

- entrare nella *home degli utenti* è illegale (figuriamoci ispezionare il materiale contenuto) se non per ragioni tecniche[3], operazioni che dovranno sempre essere concordate con l'interessato e esplicitamente permesse;
- il *superutente* potrà muoversi e operare con estrema disinvoltura, ciò conduce all'immediato effetto nefasto di permettere l'eventuale danneggiamento del sistema stesso.

Quanto appena scritto è condensato negli aforismi posti nell'esergo del documento.

16.3 Diventare superutente

Accedere direttamente come utente *root* è quasi sempre una pessima idea, talvolta per ragioni di sicurezza l'amministratore potrà inibirlo. Pertanto è opportuno loggarsi al sistema come utente ordinario, successivamente effettuare *escalation dei privilegi* all'utente *root*. Ciò è possibile utilizzando diverse alternative:

`su`
: il comando è l'acronimo di ***s**ubstitute **u**ser*[4]

`sudo`
: il comando è la sigla di ***s**uper**u**ser **do***

`doas`
: il comando è autoesplicativo - ***do as***

Nelle prossime sezioni saranno esaminate le diverse alternative, saranno comparate in 16.9.1 a pag. 206.

[1] per l'utente *root* la propria home è `/root`, quindi si è utilizzato lo *shortcut* ~, ma ovviamente non era necessario esplicitarlo poiché dal prompt si nota che la *working directory* era già quella, per cui il primo comando poteva equivalentemente essere troncato a `ls` senza esplicitare l'argomento

[2] v. ***Esempio 2.6*** a pag. 18

[3] ad esempio un backup

[4] talvolta identificato anche con ***s**witch **u**ser*

16.3.1 switch user

Possiamo utilizzare il comando `su` che ha lo scopo di eseguire comandi con utenza differente dalla propria, se non espressi sarà avviata interattivamente la shell associata all'utenza.
La sintassi generale[5] è la seguente:

```
1 $ su [option] [-] [user [argument...]]
```

Esempio 16.3: *Sintassi generale del comando* `su`

in cui

`option`
: sono le opzioni, le principali sono mostrate in ***Tabella 16.1***

`user`
: indica su quale utente si vorrà assumere l'identità, se non espresso si intenderà `root`.

OPZIONI		DESCRIZIONE
BREVE	ESTESA	
`-c`	`--command`	per eseguire uno specifico comando
`-l`	`--login`	avvia la shell caricando l'environment molto simile a quello che si sarebbe ottenuto tramite procedura di login

Tabella 16.1: *Opzioni principali per il comando* `su`

Operativamente è spesso utilizzata la sintassi più breve:

```
1 $ su - [user]
```

Esempio 16.4: *Alternativa sintattica principale del comando* `su` *con caricamento dell'environment associato all'utente target*

oppure

```
1 $ su [user]
```

Esempio 16.5: *Alternativa sintattica principale del comando* `su` *senza caricamento dell'environment associato all'utente target*

in cui il carattere - è equivalente ad attivare l'opzione `--login`, quindi caricando l'environment molto simile[6] a quello che sarebbe ottenuto tramite procedura di login dell'utente target.
Per entrambi gli esempi è possibile non definire il parametro `user` che di default sarà l'utente *root*.
Un esempio è il seguente:

[5] v. `man su`
[6] per le eccezioni v. `man su`

```
1 $ su -
2 Password:
3 #
```

Esempio 16.6: *Utilizzo di* `su -` *per diventare* root

in cui il sistema proietterà in una sessione di login dell'utente target, quindi chiederà la password associata a tale utente (nel caso in oggetto del superutente) e, se digitata correttamente, si assumerà la nuova identità.
Per uscire dalla sessione dell'utente target si potrà digitare la sequenza Ctrl + d [7].
Anche il superutente potrà eseguire lo switch di utenza come mostrato di seguito:

```
1 # su - ms
2 $
```

Esempio 16.7: *Utilizzo di* `su -` *da* root

in cui si nota che dall'utente *root* non sarà richiesta la password di autenticazione per il cambio di identità.

16.3.2 superuser do

Alcune distribuzioni supportano un metodo alternativo per il cambio di identità, in particolare tramite il comando `sudo` è possibile effettuare il cambio di identità o l'escalation dei privilegi al superutente senza possederne le credenziali. La sintassi del comando è molto ricca, senza entrare nella trattazione generale[8], solitamente l'utilizzo avviene con due sintassi principali:

`sudo [-u user ]command`
: per l'esecuzione del singolo comando con l'identità associata a `user`. Se non definito i privilegi saranno quelli del superutente.
Ad esempio:

```
1 $ sudo ls -a /root
2 [sudo] password di ms:
3 ..
```

Esempio 16.8: *Esempio di* `sudo command`

`sudo [-u user] -i`
: in cui sarà eseguita la shell dell'utente target, quindi senza `user` la shell sarà di root.
Ad esempio:

```
1 $ sudo -i
2 [sudo] password di ms:
3 #
```

Esempio 16.9: *Esempio di* `sudo -i`

[7] oppure le alternative espresse in 2.1.2 a pag. 16
[8] v. `man sudo`

La questione da focalizzare è la password inserita per mutare i privilegi concessi: non è quella dell'utente target, ma la propria. Questo aspetto induce a pensare che non a tutti gli utenti è concessa la facoltà di mutare identità tramite `sudo`, ma solo ad alcuni[9]. Se non si è tra gli eletti, avverrà quanto di seguito:

```
1 $ sudo -i
2 [sudo] password di user_xxx:
3 user_xxx non è nel file sudoers. Questo evento verrà segnalato.
```

Esempio 16.10: *Esempio di `sudo -i` non*

in cui, dopo l'inserimento della propria password, all'utente `user_xxx` non è stato concesso l'uso del comando ed è segnalato l'incidente di sicurezza.
Al momento è sufficiente sapere che l'utente installatore del sistema avrà la possibilità di utilizzare `sudo`.

16.3.3 doas

L'utility `doas`[10] è simile a `sudo` ma meno popolare. Anch'essa permette ad un utente di eseguire comandi impersonando un altro utente, e in effetti è considerata un *replacement minimalista*, da [Ove]:

> «95% of the features of sudo with a fraction of the codebase»

potenzialmente con un miglioramento della sicurezza.
Il software ha una semplice configurazione[11], le due principali sintassi sono:

`doas [-u user] command`
: per l'esecuzione del singolo comando con l'idendità associata a `user`. Se non definito i privilegi saranno quelli del superutente.
Ad esempio:

```
1 $ doas whoami
2 root
```

Esempio 16.11: *Esempio di `doas command`*

`doas [-u user] -s`
: in cui sarà eseguita la shell dell'utente target, senza definire lo `user` la shell sarà di root.
Ad esempio:

```
1 $ doas -s
2 [sudo] password di ms:
3 #
```

Esempio 16.12: *Esempio di `doas -s`*

Come nel caso del comando `sudo`, la password (se dovrà essere inserita) per mutare i privilegi concessi non sarà quella dell'utente target, ma la propria.

[9] v. **??** a pag. **??**
[10] il software è un porting dell'omonimo comando *OpenBSD*
[11] v. 16.9.2 a pag. 207

16.4 Configurazione di sistema

Un sistema GNU/Linux è configurato perlopiù tramite file di testo contenuti nella directory `/etc`[12], quindi per natura la configurazione è ben definita nella localizzazione ma non è centralizzata. Essendo distribuita su più file, si evita l'effetto *bit rot*[13], per cui il superutente (tramite il proprio editor o altri strumenti del caso) modificherà o aggiungerà file di configurazione. Esempi tipici sono:

- aggiungere un utente[14], in cui sarà essenzialmente aggiunto un record nei file `/etc/passwd`, `/etc/shadow` e `/etc/group` e sarà aggiunta la *home directory* associata all'utente (a cui saranno aggiunti alcuni file preconfigurati);
- un nuovo *hard disk* collegato sarà disponibile aggiungendo un record al file `/etc/fstab`;

È difficile fornire indicazioni generali poiché i file di configurazioni sono estremamente eterogenei[15], le uniche sono:

stabilità nel tempo: difficilmente saranno rivoluzionate nel breve periodo, solitamente sono modificate in maniera minore con il susseguirsi delle release, spesso la differenza sarà minima o nulla se la maturazione del software sarà elevata;

ricchezza di note: oltre il manuale, gli stessi sviluppatori creeranno file di configurazione ricchi di note, con commenti e suggerimenti per semplificare le eventuali modifiche o per indicare casi d'uso specifici;

16.5 Processi[16]

Un programma in esecuzione è denominato *processo*. Oltre il codice del programma[17], un processo include dati quali:

- variabili d'ambiente (user, *working directory*)[18];
- *ID del processo* denominato *PID*[19], numero univoco per l'identificazione del processo nel sistema.

Per ottenere informazioni sui processi attivi del sistema si può utilizzare il comando `ps`, ad esempio senza opzioni aggiuntive mostrerà i processi attivi nel terminale corrente:

```
1 $ ps
2     PID TTY          TIME CMD
3   37648 pts/2    00:00:00 bash
4   37660 pts/2    00:00:00 ps
```

Esempio 16.13: *`ps` senza opzioni aggiuntive*

in cui:

[12] v. 12.3.6 a pag. 138

[13] un esempio opposto di centralizzazione sono i *registri di sistema* usati in altri SO

[14] procedura descritta in 17.4.1 a pag. 218

[15] alcuni saranno strutturati con linee del tipo *key* = *value*, altri in sezioni, altre saranno directory con più file, altri script, altri file *JSON* o *XML*, ...

[16] questa sezione è solo un accenno, sarà argomentata in **??** a pag. **??**

[17] inteso nel linguaggio macchina del processore

[18] v. 11.2 a pag. 124

[19] acronimo di ***process ID***

`PID`
: è l'identificativo del processo;

`TTY`
: indica il nome del terminale[20]

`TIME`
: indica il *tempo di CPU* utilizzato dal processo;

`CMD`
: è il comando eseguito.

È possibile aumentare la verbosità dell'output, ad esempio:

```
1 $ ps -x
2     PID TTY      STAT   TIME COMMAND
3   37648 pts/2    S      0:00 -bash
4   37661 pts/2    R+     0:00 ps -x
```

Esempio 16.14: `ps x`

in cui è cambiato leggermente l'output[21] inoltre è stata aggiunta la colonna `STAT` per identificare il *process state*. Nel sistema un processo è sempre associato ad uno *stato*, i principali[22] sono:

R: Running o **Runnable**, al processo è assegnato o può essere assegnato *tempo di CPU*[23];

S: Sleeping, il processo è in attesa di completamento di un evento[24] (tipicamente un input da tastiera o dati da disco);

D: (in deep) uninterruptible sleep, il processo è inattesa di un evento (tipicamente *IO*) e non può essere disturbato. I processi non dovrebbero rimanere in questo stato per troppo tempo poiché potranno essere forzatamente rimossi da tale stato solo via *boot* (quindi se accade sarà avvenuto a causa di un errore);

T: temporarily stopped, il processo è stato temporaneamente stoppato dal proprietario o dall'amministratore tramite un segnale[25] (tipicamente un Ctrl + z);

X: Dead, non dovrebbe mai essere visto;

Z: Zombie, il processo è terminato, ma il suo *exit code*[26] non è stato catturato dal *parent*. Ciò significa che il processo non è realmente *morto* ma rimane sospeso nel sistema[27]. Se il sistema è infestato da *processi zombie*, si può assumere ci sia un problema nel processo che crea i processi in quanto non gestisce correttamente la terminazione dei *child*; in ogni caso essi spariranno quando il processo padre sarà terminato[28].

[20] `pts/number` solitamente si riferisce ad un *terminal window*

[21] è stato utilizzato il *BSD-style*

[22] per l'enumerazione completa v. `man ps`

[23] più precisamente è nella *run queue*

[24] più precisamente *interruptible sleep*

[25] v. **??** a pag. **??**

[26] si affronterà l'argomento in **??** a pag. **??**

[27] differentemente dalla vita più o meno reale, gli *zombie* non sono realmente un problema poiché non consumano più risorse (ad eccetto dello slot a loro assegnato nella *process table*)

[28] più precisamente diverranno *orfani*, quindi saranno adottati dal processo *init* che a sua volta li terminerà

Il comando `ps` accetta diversi argomenti, ad esempio si può inserire il numero di un processo:

```
1 $ ps $$
2     PID TTY      STAT   TIME COMMAND
3   37648 pts/2    S      0:00 -bash
```

Esempio 16.15: *`ps $$`*

in cui `$$` è il *PID* del processo corrente (la shell), oppure se si vogliono informazioni dettagliate su un particolare processo:

```
1 $ ps l 1
2 F   UID     PID    PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME
      COMMAND
3 4     0       1       0  20   0 170120 12580 -      Ss   ?          0:07 /
      sbin/init
```

Esempio 16.16: *`ps l 1`, in cui `l` mostra un formato lungo*

in cui le colonne nuove sono:

`F`
: riporta la somma dei *flags* associati al processo (4 per processi che utilizzano privilegi da *superuser*)

`UID`
: è l'`ID` numerico del proprietario del processo[29];

`PID`
: è il *process* `ID`;

`PPID`
: è il *process* `ID` del processo *parent*;

`PRI`
: è la priorità assegnata nella schedulazione[30]: valore intero compreso tra 0 e 139, più alto è il valore minore è la priorità;

`NI`
: *niceness*[31] associata al processo: valore intero compreso tra -20 (più favorevole) a +19 (meno favorebole) che modifica la priorità nella schedulazione;

`VSZ`
: acronimo di ***v**irtual **m**emory **s**ize* del processo misurato in KiB;

`RSS`
: acronimo di ***r**esident **s**et **s**ize* del processo (memoria utilizzata in RAM) misurata in KiB;

`WCHAN`
: nome della funzione del kernel nella quale il processo è in *sleeping*, il carattere – indica che il processo è *running*.

[29] v. 17.1 a pag. 213
[30] v. **??** a pag. **??**
[31] v. **??** a pag. **??**

Da notare che le colonne `VSZ` e `RSS` sono differenti, la prima misura la memoria totale associata al processo[32], la seconda esclude la quantità di memoria occupata sul disco fisso.

Un ulteriore esempio è il seguente:

```
1 $ ps axu
2 USER   PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
3 root     1  0.0  0.1 170120 12580 ?        Ss   gen04   0:07 /sbin/init
4 ...
5 ms   37648  0.0  0.0  11060  5268 pts/2    S    gen05   0:00 -bash
6 ...
```

Esempio 16.17: *`ps aux`, in cui `a` mostra tutti i processi, `x` elimina il vincolo di una `tty` associata, `u` l'user associato*

in cui le colonne nuove sono:

`USER`
: indica l'user associato al processo;

`%CPU`
: indica la percentuale di utilizzazione di CPU[33];

`%MEM`
: rapporto di *resident set size* diviso òa memoria del sistema espresso in percentuale;

`START`
: orario (o data se >= 24 ore) in cui il comando è stato eseguito.

Altri esempi si trovano nel manuale[34] del comando.

16.6 Memoria di sistema[35]

Informazioni relative sia alla memoria fisica libera e usata sia allo swap del sistema sono fornite dal comando `free`, informazioni recuperate dal parsing del file `/proc/meminfo`. Senza parametri il comando mostrerà quanto di seguito:

```
1 $ free
2               total        used        free      shared  buff/cache   available
3 Mem:        8034356     3973440      141296      748316     3919620     3007328
4 Swap:       6196216      765696     5430520
```

Esempio 16.18: *`free`*

in cui:

- lette per linea:

  `Mem:`
  : si riferisce alla memoria fisica del sistema (RAM);

[32] quindi somma dell'occupazione in RAM e in swap

[33] tempo di CPU utilizzato diviso il tempo di *running* del processo (cputime/realtime), il valore 100% sarebbe ottimale

[34] man ps

[35] questa sezione è solo un accenno, sarà argomentata in **??** a pag. **??**

`Swap:`
si riferisce alla memoria di swap su device;

- lette per colonna:

`total`
memoria totale installata (`MemTotal` e `SwapTotal`;

`used`
calcolata come `total - free - buffer - cache`;

`free`
memoria non utilizzata (`MemFree` e `SwapFree`;

`shared`
memoria utilizzata principalmente da *tmpfs* (`ShMem`);

`buffers`
memoria utilizzata dai buffer del kernel *tmpfs* (`Buffers`);

`cache`
memoria utilizzzata nell'attività di caching (`Cached` e `SReclaimable`);

`buff/cache`
è la somma dei precedenti `buffers` e `cache`;

`available`
è la stima di quanta memoria è disponibile per eseguire nuove applicazioni (escludendo lo swapping).

Il comando ha diverse opzioni, per esempiopuò mostrare un output *human readable*:

```
1 $ free --human
2                total        used        free      shared  buff/cache   available
3 Mem:           7,7Gi       2,8Gi       200Mi       726Mi       4,7Gi       3,9Gi
4 Swap:          5,9Gi       0,0Ki       5,9Gi
```

Esempio 16.19: *`free` con l'opzione `--human` o `-h`*

in cui i suffissi *Ki*, *Mi*, *Gi* sono rispettivamente i kibibyte, mebibyte e gibibyte in accordo con la ***Tabella 24.8*** a pag. 405.
Affinché l'output mostri gli omologhi multipli del *SI*, si può aggiugnere l'opzione `--si`:

```
1 $ free --human --si
2                total        used        free      shared  buff/cache   available
3 Mem:           7,8G        2,8G        190M        726M        4,8G        4,0G
4 Swap:          6,1G        0,0K        6,1G
```

Esempio 16.20: *`free` con l'opzione `--human` o `-h` aggiungendo `--si`*

in cui i suffissi *K*, *M*, *G* sono rispettivamente i kilobyte, megabyte e gigabyte mostrati nella medesima tabella.

16.7 Istantanee dei processi e della memoria[36]

Una combinazione delle informazioni mostrate dai comandi `ps` e `free` (di questo capitolo) e `uptime`[37] è fornita dal comando `top` come mostrato in ***Figura 16.1***. la cui schermata

[36] questa sezione è solo un accenno, sarà argomentata in **??** a pag. **??**
[37] già incontrato in 12.3.9 a pag. 140

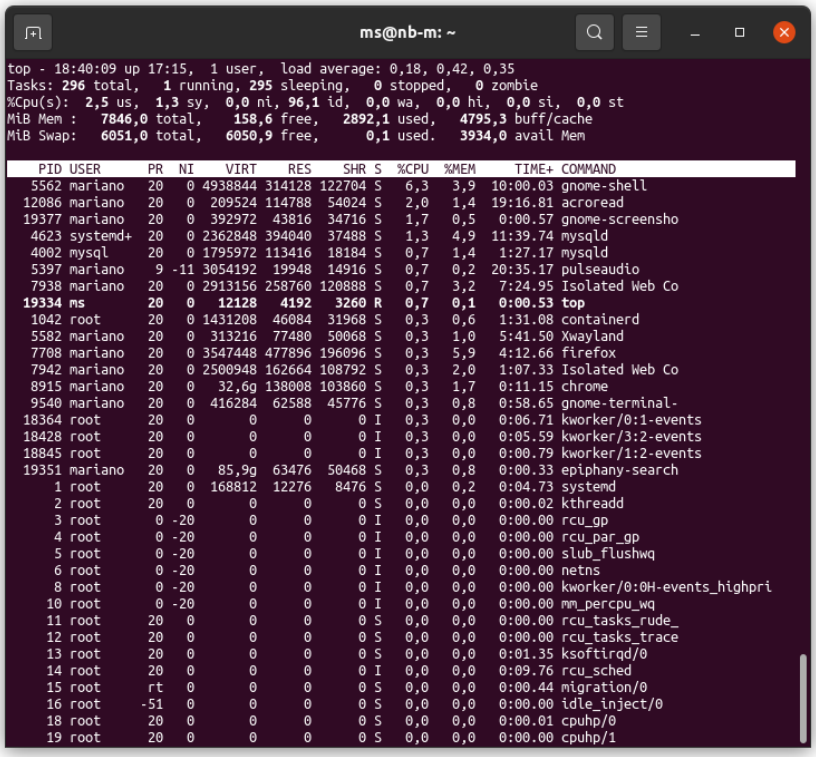


Figura 16.1: *Esempio di comando* `top`*, la console si aggiornerà ogni 3 secondi*

si aggiornerà ogni 3 secondi.
Una breve descrizione di quanto mostrato:

- la prima linea mostra l'orario di sistema e di seguito l'output quivalente ad `uptime` il quale riporta:
  - il tempo passato dall'avvio del sistema;
  - il numero degli utenti loggati;
  - una tripletta di numeri che indica il *system load average*[38] rispettivamente nell'ultimo minuto, negli ultimi 5 minuto e negli ultimi 15 minuti;
- la seconda linea mostra il numero dei processi[39] e la distribuzione degli stati;
- la terza linea mostra mostra le percentuali della distribuzione di utilizzazione della CPU[40] nell'intervallo tra i refresh:
  - `us` per **user** è l'esecuzione di codice utente;
  - `sy` per **system** è l'esecuzione del kernel;

[38] v. **??** a pag. **??**
[39] threads nel caso si nella modalità specifica
[40] per approfondimenti v. **??** a pag. **??**

  - `ni` per **nice** è l'esecuzione di processi *niced*[41];
  - `id` per **idle**, tempo inattivo;
  - `wa` per **IO-wait**, completamento di attività di I/O;
  - `hi` per **hardware interrupt**, processamento di tali segnali elettrici;
  - `si` per **software interrupt**, processamento delle chiamate mesesime;
  - `st`, per **stolen**, tempo di CPU utilizzato dall'*hypervisor* (tendenzialmente per servire altre *virtual machines* e non quella in oggetto);

- la quarta e quinta corrispondono all'output di `free`[42];
- le successive sono simili all'output mostrato da `ps u`, di default ordinato secondo il campo `%CPU` che indica la percentuale di di *CPU time*del processo rispetto al sistema (quindi la lista dei processi iniziali sono quelli che utilizzano la percentuale di CPU maggiore).

Durante il refresh dell'output è possibile digitare alcuni tasti che corrisponderanno a funzioni diverse, le principali:

`M`
: per passare ad un ordinamento per `%MEM`;

`P`
: per passare ad un ordinamento per `%CPU`;

`h`
: per visualizzare l'*help*;

`q`
: per uscrire dalla finestra o dal software.

16.8 Package Management Intro[43]

Una distribuzione normalmente[44] consiste di diverse migliaia di pacchetti, ciascuno contenente tutto il necessario per un determinato compito: programmi, script, librerie[45], documentazione, ...
Già in fase di installazione l'amministratore sceglierà alcuni dei tanti pacchetti da installare nel sistema, successivamente si potrà modificare tale lista ad esempio aggiungendo, aggiornando o rimuovendo pacchetti.
Ad esempio in questo momento su questa distribuzione sono installati 3765 pacchetti e quelli relativi all'editor *Vim* sono di seguito:

```
1 vim - Vi IMproved - enhanced vi editor
2 vim-common - Vi IMproved - Common files
3 vim-runtime - Vi IMproved - Runtime files
4 vim-tiny - Vi IMproved - enhanced vi editor - compact version
```

[41] per i quali la priorità è stata esplicitamente modificata, già accennato in 16.5, sarà argomentato in **??** a pag. **??**

[42] v. 16.6

[43] questa sezione è solo un accenno, sarà argomentata in 19.2.1 a pag. 257

[44] quasi tutte le distribuzioni hanno un sistema di pacchettizzazione, poche non l'hanno ovviamente per scelta, ma ovviamente non sono popolari

[45] v. 16.9.5 a pag. 211

Esempio 16.21: *Packege sul sistema relativi all'editor* Vim

Il package management si basa su un *database dei pacchetti* contenente le informazioni sui pacchetti della distribuzione e il loro stato[46].
Ciascun pacchetto è versionato, per cui il database tende rapidamente ad essere obsoleto, quindi andrà spesso confrontato con i *repository*[47], aggiornando le versioni relative ai pacchetti ed eventualmente scoprendone nuovi.

Il gestore dei pacchetti ovviamente permetterà le operazioni di installazione, aggiornamento e cancellazione, ma queste potrebbero essere operazioni non banali a causa della *gestione delle dipendenze*. Ad esempio, un caso complesso potrebbe essere il seguente: l'aggiornamento di un pacchetto potrebbe richiedere una versione di libreria più recente di quella installata, e quella più recente potrebbe essere incompatibile con un altro software installato che invece richiede proprio la vecchia versione di libreria[48]. Un sistema di package management *deve* rilevare tali situazioni e avvisare l'amministratore per permettere l'intervento.

I due *package management system* maggiori appartengono alle due maggiori famiglie di distribuzioni, e ciascuna ha il proprio formato di pacchetti:

dpkg: abbreviazione di ***D**ebian **p**ackage*, è il componente base del sistema di gestione della distribuzione *GNU/Linux Debian* i cui pacchetti hanno estensione **.deb**;

RPM package manager: precedentemente *Red Hat Package Manager*, abbreviato in *RPM*, è il sistema di gestione dei pacchetti utilizzato in numerose distribuzioni tra cui *Red Hat*, *Fedora*, *CentOS* e *openSUSE* i cui pacchetti hanno estensione **.rpm**.

Entrambe affrontano i medesimi problemi, ma sono differenti sia i dettagli implementativi dei *package* sia i comandi del *package manager*.
Ad esempio, per ottenere la lista dei package installati in un sistema:

Debian-based si digiterà:

```
1 $ dpkg --list
2 ...
```

Esempio 16.22: *Lista dei pacchetti installati in una* Debian-based

in cui l'opzione sipuò abbreviare con `-l`;

RPM-based si digiterà:

```
1 $ rpm --query --all
2 ...
```

Esempio 16.23: *Lista dei pacchetti installati in una* RPM-based

in cui le opzioni abbreviate sono `-qa`.

Il database dei pacchetti è localizzato in `/var/lib`:

[46] ad esempio se sono installati

[47] server contenenti pacchetti della distribuzione

[48] solitamente nel rilascio di una libreria si tende a rimanere *retrocompatibili* proprio per evitare tali problemi, ma può accadere che la retrocompatibilità non sia rispettata

Debian-based in `/var/lib/dpkg` e l'ultima copia dei repository in `/var/cache/apt` che conterrà anche pacchetti scaricati;

RPM-based in `/var/lib/rpm`.

In tempi relativamente recenti, i *package manager* citati sono spesso non utilizzati direttamente, ma si preferisce utilizzare software più astratti[49] che permettono:

- accedere in maniera diretta ai repository online (quindi anche cercare pacchetti);
- risoluzione automatica delle dipendenze tra pacchetti[50].

Quindi l'amministratore solitamente preferirà utilizzare strumenti più evoluti che a loro volta utilizzeranno il rispettivo tool di base:

Debian-based: `apt-cache`, `apt-get` e `apt`[51];

RPM-based: più recente `dnf`[52], meno recente `yum`[53].

16.9 Approfondimenti

16.9.1 Comparazione tra `su`, `sudo` e `doas`

I comandi `su`, `sudo` e `doas` permettono di eseguire attività amministrative con privilegi elevati. Di seguito un elenco utile per il confronto:

diffusione: legata all'installazione di default nella varie disribuzioni:

`su` è il sistema tradizionale presente su tutte le distribuzioni, ed è anche il metodo più utilizzato per lo switch all'utente `root`;

`sudo` è stato dapprima installato sulle *Ubuntu-based*, durante l'installazione all'account `root` non è associata nessuna password (per cui il superutente non è accessibile tramite `su`);

`doas` è molto meno popolare dei precedenti;

configurazione: `su` non ha configurazioni, l'utente dovrà conoscere semplicemente le credenziali associate all'utente target;

`sudo` permette il rispetto del principio di *privilegio minimo* associato all'utenza: la preconfigurazione effettuata in fase di installazione permetterà all'installatore di scalare ai privilegi del superutente[54], si potrà modificare il file di configurazione in maniera gradulare per compiti specifici;

`doas` permette anch'esso il rispetto del principio di *privilegio minimo* associato all'utenza, tramite una configurazione più semplice;

sicurezza: `su` prevede la conoscenza delle credenziali dell'utente target, nel caso di *root* la password associata, quindi una password condivisa non è sicuramente una buona pratica;

---

[49] v. 19 a pag. 253

[50] approfondimento in **??** a pag. **??**

[51] ***A**dvanced **P**ackage **T**ool*

[52] *Dandified YUM*

[53] ***Y**ellowdog **U**pdater, **M**odified*

[54] di base crea un gruppo a cui associa l'utente installatore, ma se ne possono aggiungere altri (v. **??** a pag. **??**)

`sudo` e `doas` permettono di accedere a privilegi maggiori per mezzo di configurazione, non tramite conoscenza di credenziali. Ad esempio, se `sudo` è installato in fase di installazione della distribuzione, di default l'installatore accederà a *root* tramite la propria credenziale, non tramite quelle associate al superutente (che di default non ha password).

16.9.2 Configurazione e utilizzo di `doas`

Per una esaustiva trattazione è consultabile il manuale[55], di seguito saranno mostrare le principali modalità di configurazione.

Configurazione Se il tool è installato ma non configurato, normalmente apparirà il messaggio:

```
1 $ doas ...
2 doas: doas is not enabled, /etc/doas.conf: No such file or directory
3 $
```

Esempio 16.24: *`doas` installato ma non configurato*

Quindi come suggerito sarà necessario creare il file `/etc/doas.conf`.
Una semplicissima linea di configurazione potrebbe essere la seguente:

```
1 permit ms as root
```

Esempio 16.25: *Semplice linea di configurazione per `doas`*

Brevemente, la linea abiliterà l'utente `ms` ad eseguire comandi come `root` dopo aver immesso la propria password.
Meno brevemente, la lineapuò essere suddivisa in tre componenti:

`permit`
: è l'azione. L'alternativa è `deny` che implica il diniego. Ad esempio, `deny ms as root` implicherebbe (se la regola non venisse modificata da una successiva, ad esempio fosse in fondo al file) che l'user `ms` non potrà mai agire da utente `root` indipendentemente da altre regole precedenti;

`ms`
: è l'identità: può essere una username, un gruppo[56][57], la *UID*[58];

`as root`
: è l'identità target. Ovviamente `root` può essere sostituito con un altro username, gruppo o *UID*. Il target è opzionale, se non è definito il target sarà *qualsiasi utente*. Ad esempio, se l'intera linea fosse `permit ms`, l'utente `ms` potrà eseguire qualsiasi comando con qualsiasi utente.

In generale, una linea di configurazione è:

```
1 permit|deny [options] identity [as target] [cmd command [args ...]]
```

Esempio 16.26: *Sintassi di una linea di configurazione per `doas`*

[55] v. `man doas`
[56] v. 17.2 a pag. 214
[57] per essere considerati tali, i gruppi devono essere preceduti dal carattere `:`
[58] ***user id***, v. 17.1 a pag. 213

in cui:

options

`nopass`
: disabilita la richiesta della password dell'utente quando utilizzerà il tool. Per esempio, una linea potrebbe essere:

```
1 permit nopass ms as root
```

Esempio 16.27: *Linea di configurazione per* `doas` *per non richiedere la password all'utente*

che permetterà all'utente `ms` di eseguire qualsiasi comando come `root` senza che fornisca la propria password;

`permit`
: è una scelta intermedia tra i due estremi `permit` e `permit nopass`: l'opzione `persist` la quale richiederà la password all'utente solo la prima volta, successivamente l'accesso sarà diretto per un lungo periodo;

`keepenv`
: manterrà le variabili di environment quando sarà creato l'environment del nuovo processo;

`setenv`
: preserverà o setterà le variavili passate al comando le quali dovranno essere:

- incluse tra parentesi graffe;
- ciascuna separata da uno spazio;
- immettendo un segno uguale per valorizzare la variabile;
- solo elencarla per preservarla;
- precedendo la variabile con un segno meno per eseguire l'*unset*;
- precedendo la variabile con il carattere `$` per riferirsi alla variabile d'ambiente;

`cmd`
: per vincolare l'uso dello strumento ad un comando in particolare, eventualmente con taluni argomenti. Ad esempio:

```
1 permit ms as root cmd apt args update
```

Esempio 16.28: *Linea di configurazione per* `doas` *con vincolo di comando e argomento*

permetterà all'utente di eseguire come `root` il comando `apt` con argomento `update`. Qualsiasi argomento differente produrrà un errore `Operation not permitted`. Se nella linea di configurazione `arg` non è specificato, l'utente potrà aggiungere argomenti arbitrariamente, se la linea terminerà con `args`, essi saranno disabilitati.

Una linea di configurazione completa è la seguente:

```
1 permit nopass setenv { -http_proxy APT_CONFIG=/etc/apt/apt.conf.d/50
      appstream } :updaters cmd apt args update
```

Esempio 16.29: *Linea di configurazione per* `doas`

la quale permetterà a tutti gli utenti del gruppo `updaters`:

- di eseguire `apt update`;
- senza necessità di inserire la password;
- con l'*unsettng* della variabile `http-proxy`;
- settando la variabile `APT_CONFIG` con il path indicato.

Utilizzo Di default, se l'utente target non sarà definito, il tool assume che sia `root`. Aggiungendo il flag `-u` si potrà esplicitare la user. Ad esempio:

```
1 $ doas -u ms2 whoami
2 ms2
```

Esempio 16.30: *Uso di `doas` con indicazione di utente target*

l'utente tenta di identificarsi con `ms2` e, se l'utente di destinazione esiste e se presente la linea di configurazione che lo permette, sarà mostrato l'output indicato.
Il flag `-s` è utilizztoper lanciare una shell dell'utente target, ad esempio:

```
1 $ doas -s
2 #
```

Esempio 16.31: *Uso di `doas` per lanciare una shell*

la quale apriràuna shell per l'utente `root`. Ovviamente si potrà aprire una shellper un altro utente terger:

```
1 $ doas -s -u ms2
2 $ whoami
3 ms2
```

Esempio 16.32: *Uso di `doas` per lanciare una shell con definizione di utente target*

in cui la nuova shell restituirà l'identificazione dell nuovo utente.

Ogni volta che si modifica il file `doas.conf` è consigliabile verificare che non siano stati inseriti errori sintattici. In questo caso è utile l'opzione `-C` la quale eseguirà un check del file di configurazione:

```
1 $ doas -C /etc/doas.conf
```

Esempio 16.33: *Check di `doas.conf`*

il quale non applicherà le regole, ma mostrerà eventuali errori sintattici.

Nel file si possono inserire i commenti aggiungendo il carattere iniziale #[59], infatti ne sono stati inseriti diversi nel file di esempio
`/usr/share/doc/opendoas/examples/doas.conf`[60] installato con il software, utilizzabile come configurazione di partenza o comunque valido per spunti.

Per qualsiasi dettaglio della sezione si consulti il manuale[61].

[59] carattere *hashmark*

[60] una copia è all'indirizzo https://github.com/slicer69/doas/blob/master/doas.conf.sample

[61] `man doas.conf`

16.9.3 system call e processi

In estrema sintesi Il kernel ha il compito della creazione del processo, assegnazione del *tempo di CPU*, spazi adeguati per la memoria, ed infine pulizia della stessa quando sarà terminato. Affinché il kernel possa procedere in tal senso, i processi dovranno chiamare le *system call*[62] per accedere al sistema, quindi alla memoria, al filesystem, alla rete e ai vari dispositivi, ma anche alle astrazioni fornite dallo stesso kernel. Tra queste i processi possono invocare due *system call* particolarmente interessanti in cui il processo chiamante:

fork(): crea un nuovo processo *quasi identico*[63] a se stesso;

exec(): esegue un nuovo processo non preservando se stesso.

L'inizio della catena di creazione è il primo processo con *PID 1* creato dal kernel durante la fase di boot. Tale processo è denominato *init process*, è localizzato in `/sbin/init`, ed è responsabile (direttamente o indirettamente) della creazione degli altri processi del sistema, e diviene padre di eventuali processi che restano *orfani*[64].

16.9.4 Rimuovere tutto, forse

Il *superutente* ha l'abilità di scrivere dappertutto nel sistema, quindi ha la via privilegiata per commettere danni importanti. In particolare un sistema GNU/Linux non chiederà all'amministratore se è sicuro di ciò che digita, semplicemente lo eseguirà in maniera efficiente senza porsi le domande sulle conseguenze[65].
Un'eccezione la si è creata per il comando `rm`, perché un errore di digitazione potrebbe essere fatale. Ad esempio, supponiamo che il superutente voglia cancellare ricorsivamente la directory `/tmp/da_cancellare`, quindi scriverà qualcosa come:

```
1 $ rm -rf /tmp/da_cancellare
2 $
```

Esempio 16.34: *Cancellazione tramite `rm -rf` di un path*

Ipotizziamo che per errore si sia digitato il path in maniera errata, ad esempio:

```
1 $ rm -rf / tmp/da_cancellare
2 $
```

Esempio 16.35: *Cancellazione tramite `rm -rf` con digitazione errata di path*

in cui si nota che il path immesso è `/ tmp/da_cancellare`, quindi si è inserito involontariamente uno spazio dopo il primo `/`. Facile notare il problema: il singolo path sono divenuti due path. Se da superutente il sistema eseguisse una rimozione ricorsiva della directory `/` ci sarebbero gravi conseguenze per l'intero filesystem.
Per tale motivo, la cancellazione ricorsiva della directory radice è per default inabilitata, in particolare il comando `rm -fR /`[66] non andrà a buon fine in quanto di default è abi-

[62] funzioni messe a disposizione dal SO

[63] la copia creata è quasi identica all'originale poiché è identica a livello di codice e memoria, ma il processo che ha generato la copia si chiama *parent*, la copia generata si chiama *child*

[64] argomento trattato in **??** a pag. **??**

[65] è una impostazione filosofica che condivido pienamente

[66] ho visto diverse volte questo comando stampato in forma di logo (magliette o vignette), ma mi viene da sorridere poiché non è corretta

litata l'opzione `--preserve-root`. Se però per una qualche ragione si vorrà comunque eseguire il comando di rimozione, dovrà essere aggiunta contestualmente l'opzione `--no-preserve-root`.

16.9.5 Pacchettizzazione delle librerie

Si è soliti distinguere i pacchetti delle librerie in due categorie:

run-time package: contengono software da installare (ad esempio *librerie caricabili dinamicamente* con estensione `.so`) che sarà utilizzato da altri eseguibili;

development package: contengono software di sviluppo (ad esempio sorgenti delle librerie, documentazione di sviluppo, ...) da installare solo se si è intenzionati alla compilazione di software proprio (o ricompilazionedi software terzo); un package tipico potrebbe contenere le instestazioni delle librerie scritte in *C* (quindi con estensione `.h`) da includere nella compilazione si software.

16.10 Concetti chiave

bit rot: genericamente identifica la decadenza nel tempo della qualità informativa (ad esempio per accumulo di informazioni non più necessarie o ridondanti o trascuratezza o difficoltà tipiche nella manutenzione centralizzata che conducono a degradazione)

escalation dei privilegi: anche *privilege escalation*, è la procedura attraverso la quale un utente ordinario acquisisce maggiori privilegi

JSON: sigla di ***J**ava**S**cript **O**bject **N**otation*, è un formato per lo scambio dati basato sul linguaggio di programmazione JavaScript, spesso utilizzato in programmazione come alternativa al formato *XML*

package management system: sistema di gestione dei pacchetti

process table: tabella dei processi del sistema con i relativi metadati associati

repository: server contenenti pacchetti della distribuzione

root: è l'*amministratore di sistema*, denominato anche *superutente*

`/root`: è la *home directory* dell'utente *root*

run queue: è la coda di sistema dei processi che sono in *running* (o possono esserlo), quindi è assegnato *tempo di CPU* (o pronto per l'assegnazione)

runnable: processo che può essere eseguito (è in *run queue*)

running: processo in esecuzione (è in *run queue*)

system call: meccanismo usato da un processo per richiedere un servizio a livello kernel del SO

tempo di CPU: indica il tempo cumulato di CPU assegnata

XML: sigla di ***ex**tensible **m**arkup **language*** è un *metalinguaggio* per la definizione di linguaggi di markup, in altri termini è un linguaggio basato su un meccanismo sintattico che consente di definire il significato degli elementi contenuti in un documento, spesso utilizzato in programmazione come alternativa al formato *JSON*

16.11 Comandi introdotti nel capitolo

`doas` ⇒ ***do as***, simile a `sudo` ma minimalista

`free` ⇒ restituisce informazioni relative sia alla RAM utilizzata sia allo swap

`ps` ⇒ mostra i processi correnti

`su` ⇒ acronimo di ***substitute user***, permette il passaggio ad altra identità

`sudo` ⇒ ***superuser do***, comando che permette l'esecuzione con privilegi associati ad un altro utente

`top` ⇒ combinazione delle informazioni mostrate da `ps`, `free` e `uptime` con tempo di refresh di default di 3 secondi

Capitolo 17

Amministrazione degli utenti

QUESTO capitolo avrà per oggetto l'amministrazione degli utenti locali[1].
Il sistema GNU/Linux distingue tra due categorie di utenza:

- il *superutente* che avrà il compito di amministrare il sistema;
- *utenti comuni*, denominati *normal users* i quali non avranno visibili molti degli aspetti di sistema, certamente non potranno effettuare modifiche allo stesso.

I normali utenti potranno sostanzialmente utilizzare gli applicativi installati nel sistema e disporre delle propria *home directory* entro i limiti fisici e tecnologici imposti dallo stesso (o eventuali vincoli impostati dal superutente).

17.1 Utenza

Nel sistema gli utenti sono distinti tramite l'*user account* assegnato. Durante l'installazione quasi tutte le distribuzioni creano due user account, di cui uno per i task amministrativi cui è assegnato il nominativo *root*, l'altro account sarà creato come *normal user*. Successivamente, secondo necessità, l'amministratore creerà ulteriori account.
Tra le altre informazioni assegnate, gli account sono distinti tramite due informazioni univoche:

UID: un ***user ID***, un numero identificativo[2];

username: una stringa testuale come `root` o `ms`.

All'utente `root` è associato l'*UID* 0, ai normali utenti l'*UID* iniziale associato è *1000* e sequenzialmente a crescere con l'utenza.

[1] solitamente in reti più o meno ampie la configurazione degli utenti non sarà memorizzata localmente, ma in qualche sistema prevedibilmente in un *database* (argomento affrontato in **??** a pag. **??** o in un *server LDAP* per ottenere un'autenticazione centralizzata dei client

[2] all'utente

17.2 Utenti e gruppi

Durante la sessione di lavoro, un utente lavorerà nella propria *home directory* in cui solo[3] l'utente proprietario avrà pieni diritti[4], sicuramente gli altri utenti non potranno scrivere. È però possibile voler accedere a risorse di sistema o a file in maniera condivisa da *gruppi* di utenti. Un sistema GNU/Linux identifica i membri di un gruppo in maniera simile a quella degli utenti, in particolare sono identificati internamente tramite un *ID numerico* denominato *GID*[5], ma non hanno una directory assegnata[6].

Ogni utente apparterrà ad un *gruppo principale*, ed eventualmente ad altri gruppi[7].
Per quanto concerne la politica di autorizzazioni, ogni gruppo è equivalente, quindi un utente avrà i medesimi privilegi[8] derivanti dall'appartenenza a qualsiasi gruppo. L'unica differenza tra il gruppo principale e quelli secondari è che i file creati da un utente di default sono assegnati anche al gruppo principale[9].

Per ottenere l'*UID* assegnato ad un user, il gruppo principale e gli altri gruppi, si può digitare:

```
1 $ id
2 uid=1002(ms) gid=1002(ms) gruppi=1002(ms)
3 $ id root
4 uid=0(root) gid=0(root) gruppi=0(root)
```

Esempio 17.1: UID *e* GID *associati all'utente*

in cui il primo output è per l'utente medesimo, il secondo è per l'user indicato come argomento.
L'elenco dei gruppi associati ad una user può essere ottenuto tramite il comando `groups`:

```
1 $ groups
2 ms
3 $ groups ms root
4 ms : ms
5 root : root
```

Esempio 17.2: `groups`

17.2.1 Pseudoutenti e pseudogruppi

Il concetto di *user* e i *group* è stato concepito per allocare diritti di accesso e gestione per funzionalità del sistema. Ciò implica che oltre agli account assegnati a persone, ci

[3] eccezione per il superutente che potrà leggere e scrivere
[4] l'argomento sarà trattato in 18 a pag. **??**
[5] ***g****roup* ***ID***
[6] a meno che l'amministratore non provveda a tale creazione assegnando per tali directory la destinazione d'uso appropriata assegnando i diritti al gruppo
[7] ad esempio, in sistemi aziendali molto ampi è normale introdurre gruppi per settori o progetti specifici, quindi assegnare utenti a tali gruppi secondo necessità, proprio per permette di condividere e gestire dati esclusivamente tra i membri del gruppo
[8] dal punto di vista sistemistico
[9] non c'è troppo da preoccuparsi per la condivisione, di default il gruppo principale è creato per ciasun utente omonimo alla username, quindi di default conterrà solo l'utente medesimo

sono altri account che corrispondono a *pseudoutenti* a cui sono assegnate funzioni amministrative. Questi account hanno *ruoli funzionali*, già nell'installazione[10] saranno installati pseudoutenti e pseudogruppi omonimi nei file `/etc/passwd` (ovviamente nel suo omologo `/etc/shadow` e in `/etc/group`.
Ulteriori pseudoutenti e/o relativi gruppi saranno installati con:

- software aggiuntivo, ad esempio `sshd` per l'installazione di un *ssh server*, `www-data` per un *server web*, `news` per le *news Usenet*, `postfix` per un *mail server Postfix*, `mysql` per un *MySQL* o *MariaDB server*, `memcache` per un server *Memcached*, `lxd` per un server *LXD* (*Linux container*;
- device, ad esempio stampanti.

Anche gli pseudoutenti e pseudogruppi hanno necessità di operare con i relativi privilegi, quindi anche per questi saranno associati *UID* e/o *GID*. Espandendo quanto già accennato in 17.1:

- all'utente `root` sono associati *UID* 0 e *GID* 0[11];
- ai normali utenti l'*UID* iniziale associato è *1000* e il *GID* è solitamente uguale;
- per gli altri account l'*UID* parte da 1 e solitamente sono sequenziale, per *GID* l'uguaglianza si perde.

17.3 Informazioni su utenti e gruppi

17.3.1 Il file `/etc/passwd`

Il file `/etc/passwd` è il database degli utenti e per ciascun utente del sistema corrisponde un'entry in cui saranno memorizzati gli attributi dello stesso. Di seguito il formato:

```
<username>:<password>:<UID>:<GID>:<GECOS>:<home
                directory>:<shell>
```

Il contenuto dei campi è il seguente:

`<username>`: è l'*username* associata all'utenza, dovrebbe essere formata da lettere in minuscolo e cifre (quest'ultime dal secondo carattere);

`<password>`: storicamente il campo conteneva la password criptata, le distribuzioni attuali utilizzando le *shadow password* in cui la memorizzazione della stessa aviene nel file `/etc/shadow`[12]. Attualmente la password è sostituita da una semplice `x` proprio per evidenziare tale questione.
Ciascun utente potrà cambiare la propria password[13];

`<UID>`: identificativo numerico compreso tra 0 e $2^{32} - 1$ assegnato univoco per ciascun utente. Per convenzione tipicamente gli *UID*:

- compresi tra 0 e 99 sono riservati per il sistema;
- compresi tra 100 e 999 sono per i software;

[10] appena installato saranno la maggior parte
[11] i privilegi di `root` sono legati solo all'*UID*, il *GID* è trascurabile
[12] può essere acceduto solo dal superutente e qualche software privilegiato
[13] rispettando eventuali restrizioni aggiunte, ad esempio *password troppo semplice*

- da 1000 procedendo in sequenza sono assegnati agli utenti umani.

Se l'univocità non è rispettata, poiché il sistema differenzia l'utenza non il base alla *user* ma all'*UID*, il sistema tratterà i due account in maniera completamente identica;

`<GID>`: identifica il *gruppo principale*, e tale campo sarà valutato successivamente al *login*. La politica di creazione varia con le distribuzioni, in particolare alcune assegnano uno stesso *GID* per tutti gli utenti umani, la maggior parte assegneranno un numero *GID* uguale a *UID*. Chiaramente nell'ultimo caso l'assegnazione dei diritti del gruppo principale è più granulare, quindi sarà più sofisticata la gestione dei diritti. Se l'utente avrà *gruppi secondari*, questi saranno visibili nel file `/etc/group`.

`<GECOS>`: è un campo di commento denominato *GECOS field*[14]. Il campo contiene informazioni sull'utente[15][16] il nome reale, dati telefonici, ...;

`<home directory>`: è l'area personale assegnata all'utente per memorizzare i propri file. Appena creata la *home directory* solitamente non sarà vuota ma conterrà dei file per un settaggio di base[17]. Successivamente alla fase di *login*, l'utente sarà proiettato al *path* indicato;

`<shell>`: indica il software eseguito se l'autenticazione del rispettivo utente sarà avvenuta con successo. L'utente potrà modificare la propria shell utilizzando il comando `chsh`[18]. L'elenco delle shell valide per il SO in uso è contenuto nel file `/etc/shells`. In questo campo non dev'esserci necessariamente una *shell*, ma può essere indicato un software con relativi argomenti. Se il campo è vuoto, sarà eseguito `/bin/sh`.

Un amministratore di sistema non dovrebbe editare il file `/etc/passwd` manualmente, sarebbe indicato utilizzare software progettati per tale modifica.

17.3.2 Il file `/etc/shadow`

Per questioni di sicurezza le distribuzioni solitamente utilizzano il sistema delle *shadow password* in cui le password criptate non sono memorizzate nel file `/etc/passwd` il cui accesso in lettura è permesso all'utenza, ma nel file `/etc/shadow` la cui lettura è inibita ai normali utenti. Solo l'utente *root* può leggere e modificare tale file, i membri del gruppo `shadow` solo leggere. Di seguito il formato:

```
<username>:<password>:<change>:<min>:<max>:
      <warn>:<grace>:<lock>:<reserved>
```

Il contenuto dei campi è il seguente:

`<username>`: deve corrispondere all'entry in `/etc/passwd` (è il campo per la *join*);

[14] *GECOS* è l'acronimo di ***G**eneral **E**lectric **C**omprehensive **O**perating **S**ystem* e tale campo era in tempi (molto) passati utilizzato per mantenere dati di compatibilità con il servizio *GECOS*

[15] per aggiornarle si può utilizzare il comando `chfn`

[16] ***ch**ange **full**name*

[17] ad esempio `.profile` e `.bashrc` i quali conterranno informazioni per la definizione dell'environment

[18] ***ch**ange **sh**ell*

<*password*>: la password criptata. Se il campo è vuoto non sarà necessaria nessuna password per autenticarsi al sistema. Il carattere ! [19] all'inizio della password corrisponde al *lock* della stessa. Per i dettagli dell'interpretazione della stringa, consulta `man 3 crypt`.

<*change*>: la data dall'ultimo cambio password, in giorni dal 1 gennaio 1970;

<*min*>: è il numero di giorni che l'utente dovrà attendere prima di cambiare nuovamente la password;

<*max*>: è il numero di giorni massimi passati i quali l'utente dovrà cambiare necessariamente la propria password. Se il campo è vuoto, non ci sarà un'età età massima per la password, periodo di avviso e nessun periodo di inattività (v. prossimi campi). Se l'età massima è minore della minima, l'utente non potrà mai cambiare la propria password;

<*warn*>: se valorizzato e il valore è diverso da 0, indica dopo quanti giorni l'utente sarà avvisato di cambiare password (l'avvio avveerà appena sarà avvenuto il login);

<*grace*>: numero di giorni di *inattività* durante i quali, nonostante la password sia scaduta, l'utente potrà ancora cambiare la propria password (sarà obbligato all'accesso). Se non avverrà, passati i giorni indicati, l'utente non potrà più accedere (per farlo dovrà contattare l'amministratore). Un campo vuoto significa che non sarà applicato periodo di inattività.

<*lock*>: data di scadenza della password. Questo non è un *account expiration*, ma una *password expiration* poiché l'utente potrà ancora loggarsi ma non gli sarà permesso utilizzando la propria password;

<*reserved*>: riservato per uso futuro.

17.3.3 Il file `/etc/group`

Il database dei gruppi è in `/etc/group`. Il file contiene un'entry per ogni gruppo di sistema il cui formato è di seguito:

```
<group name>:<password>:<GID>:<members>
```

Il contenuto dei campi è il seguente:

<*group name*>: il nome del gruppo;

<*password*>: è prevista una password opzionale. Gli utenti che non hanno come *GID* tale gruppo possono ottenerlo tramite il comando `newgrp`. Se non sono elencati tra i membri, gli sarà chiesta tale password. Se il contenuto è il carattere `*`, gli utenti che non sono membri del gruppo tramite `/etc/passwd` o `/etc/group` non potranno ottenere tale *GID*. Solitamente è presente il carattere `x` per puntare al relativo file di password `/etc/gshadow`;

<*GID*>: è l'identificativo numerico del gruppo;

<*members*>: una lista di user separati da una virgola. Questa lista contiene tutti gli utenti che hanno il gruppo definito come secondario (può anche essere listato il gruppo principale ma è non necessario).

[19] *exclamation mark*

Le password nei gruppi sono utilizzate con estrema difficoltà, per questioni di sicurezza molto più probabile l'inserimento del carattere `*` [20], oppure il carattere `x`[21].

17.3.4 Il file `/etc/gshadow`

Analogamente all'`/etc/shadow`, questo è lo *shadow password* per i gruppi. Senza questo file, la password assegnata ai gruppi dovrebbe essere contenuta direttamente nel file `/etc/group`, quindi sarebbe criptata ma leggibile per ogni utente, per cui si è preferito memorizzare le password nel file `/etc/gshadow`[22]. Nel terzo campo sono contenuti i nomi degli amministratori del gruppo che sono titolati per aggiungere o rimuovere membri dallo stesso gruppo. L'ultimo campo è la lista dei membri, lista allineata ad `/etc/group`.

17.4 Gestire gli account utente e le informazioni del gruppo

L'amministratore ha il compito di creare e gestire gli account per gli utenti. Ci sono diversi tool nel sistema per la gestione per cui saranno task semplici.

17.4.1 Creare account utente

La procedura per aggiungere un account è molto lineare e consiste dei seguenti step:

1. aggiungere un'entry nei file `/etc/passwd` e la corrispondente in `/etc/shadow`;
2. (se necessario) aggiungere un entry (o più) nel file `/etc/group`;
3. creare una *home directory*, copiare i file iniziali e assegnare i permessi[23] di proprietario e di gruppo all'utente appena creato.

Tutti i file coinvolti nell'aggiunta di un nuovo account sono *plain file* per cui è possibile eseguire ogni step manualmente attraverso un editor.
È però più immediato utilizzare il software `useradd` la cui sintassi generica è:

```
1 # useradd [options] username
```

Esempio 17.3: *Sintassi generica di* useradd

in cui le opzioni[24] principali sono:

`-c`
: inserisce il commento nel campo `GECOS`;

`-d`
: imposta la *home directory* (se omessa, si assume `/home/user`);

[20] comunque un carattere impedirà il match della password
[21] quindi il carattere `*` nell'entry del file `/etc/gshadow` corrispondente; v. prossima sezione
[22] il rapporto è equivalente tra il file `/etc/passwd` e `/etc/shadow`
[23] v. **??**
[24] i dettagli e i valori di default cambiano in maniera elastica con le distribuzioni, consigliabile consultare `man useradd`

`-e`
: data nella quale l'account sarà disattivato (formato `YYYY-MM-DD`);

`-g`
: gruppo primario di assegnazione (*GID* o nome) che dovrà già esistere;

`-G`
: gruppi secondari a cui sarà assegnato, anch'essi dovranno già esistere;

`-s`
: imposta la shell;

`-m`
: crea la *home directory* e di default saranno copiati i file contenuti in `/etc/skel` a meno che non sia utilizzato il flag `-k` (definisce un path per uno scheletro alternativo). Se tale flag non è aggiunto, di default non sarà creata la *home*.

Un esempio di creazione è il seguente:

```
1 # useradd -c "ms" -m -d /home/ms -g devel
```

Esempio 17.4: *Creazione utente tramite* `useradd`

in cui sarà creato l'utente `ms`, assegnata l'*home directory* `/home/ms` nella quale saranno copiati i file della directory `/etc/skel`, ed assegnato al gruppo `devel`. Da notare che se l'opzione `-m` non è aggiunta, la *home* non sarà creata.

Il tool `useradd` è un'utilità di basso livello per cui spesso è incorporato in uno script il quale aggiungerà l'utente secondo la policy aziendale.
Le distribuzioni *Debian-like* hanno il tool `adduser` che di default aggiungerà l'utente secondo le *Debian GNU/Linux rules*, si veda:

```
1  # adduser ms2
2  Aggiunta dell'utente «ms2» ...
3  Aggiunta del nuovo gruppo «ms2» (1003) ...
4  Aggiunta del nuovo utente «ms2» (1003) con gruppo «ms2» ...
5  Creazione della directory home «/home/ms2» ...
6  Copia dei file da «/etc/skel» ...
7  Nuova password:
8  Reinserire la nuova password:
9  passwd: password aggiornata correttamente
10 Modifica delle informazioni relative all'utente ms2
11 Inserire il nuovo valore o premere INVIO per quello predefinito
12   Nome completo []:
13   Stanza n° []:
14   Numero telefonico di lavoro []:
15   Numero telefonico di casa []:
16   Altro []:
17 Le informazioni sono corrette? [S/n] s
18 #
```

Esempio 17.5: *Creazione utente tramite* `adduser`

in cui ogni linea è stata conclusa con un ↵ ad eccezione di quelle richiedenti una nuova password (ovviamente per entrambe le richieste sarà la stessa).

17.4.2 Il comando `passwd`

Il comando `passwd` è utilizzato per cambiare la password degli utenti. Ad esempio:

```
1 # passwd ms
2 Nuova password:
3 Reinserire la nuova password:
4 passwd: password aggiornata correttamente
```

Esempio 17.6: *Cambio password tramite* `passwd`

modificherà la password per l'utente in argomento. Il comando è disponibile anche per i normali utenti che potranno cambiare esclusivamente la propria password:

```
1 $ passwd
2 Cambio password per ms.
3 Password attuale :
4 Reinserire la nuova password:
5 passwd: password aggiornata correttamente
```

Esempio 17.7: *Cambio della propria password tramite* `passwd`

Ciascun utente potrà verificare lo stato della propria password tramite l'opzione `-S`:

```
1 $ passwd -S
2 ms P 01/11/2024 0 99999 7 -1
```

Esempio 17.8: *Verifica dello stato della propria password tramite* `passwd -S`

in cui:

`ms`
: l'username;

`P` o `PS`
: se la password è impostata, `L` o `LK` se l'account è lockato, `NP` per un account senza password;

`01/11/2024`
: la data dell'ultimo cambio password;

`0` e `99999`
: rispettivamente il minimo e il massimo intervallo per cambiare la password;

`7`
: l'intervallo per il *warning*

`-1`
: il periodo di inattività prima che la password sia lockata.

Per i dettagli dell'elenco, consulta 17.3.2.
Ovviamente il superutente potrà verificarlo per tutti gli utenti aggiungendo come argomento l'user di qualsiasi utente:

```
1 $ passwd -S ms
2 ms P 01/11/2024 0 99999 7 -1
```

Esempio 17.9: *Verifica dello stato della propria password tramite* `passwd -S` *da* `root`

con ovviamente la medesima risposta.
Per eseguire un lock dell'account dell'utente `ms`:

```
1 # passwd -l ms
2 passwd: password expiry information changed.
3 # passwd -u ms
4 passwd: password expiry information changed.
```

Esempio 17.10: Lock *e* unlock *della password dell'account* `ms` *tramite* `passwd -l` *e* `passsswd -u` *da* `root`

in cui il primo comando ha effettuato il *lock* della password, il secondo l'*unlock*.
Il *lock* delle password agisce aggiungendo il carattere `!` ad inizio della password criptata, per cui sarà impossibile inserire una password che, tramite procedura di criptazione, produrrà la sequenza equivalente alla *nuova password criptata*. L'*unlock* della password agirà eliminando il carattere `!` precedentemente aggiunto. Si ricorda nuovamente che un *lock* della password dell'account non equivale ad effettuare il *lock* dell'account ma solo impedire l'accesso tramite password.

Per ulteriori opzioni si consulti il manuale[25].

17.4.3 Rimozione di un utente

Per rimuovere un utente sarà necessario rimuovere:

- le entry dal file `/etc/passwd` e `/etc/shadow`;
- cancellare i riferimenti dell'utente in `/etc/group`;
- rimuovere la *home directory* assegnata all'utente;
- rimuovere ulteriori tracce dell'utente (ad esempio in `/var/mail/user`).

Ovviamente anche qui ci sarà un comando per automatizzare questi step, in assonanza al comando `useradd` per l'aggiunta di un utente, sarà `userdel` la cui sintassi generica è:

```
1 # userdel [options] username
```

Esempio 17.11: *Sintassi generica di* userdel

in cui l'opzione principale è `-r` la quale cancellerà la *home directory* e la *mailbox* dell'utente in `/var/mail/user`. Altri file appartenenti all'utente si potranno rimuovere con il seguente comando:

```
1 # find / -uid UID -delete
```

Esempio 17.12: *Rimozione di file tramite* UID

in cui la stringa *UID* andrà sostituita con l'identificativo numerico dell'utente da rimuovere.

Anche qui il tool `userdel` è un'utilità di basso livello, per cui analogamente le distribuzioni *Debian-like* hanno aggiunto il comando `deluser` che tipicamente sarà eseguito come di seguito:

[25] `man passwd`

```
1 # deluser --remove-home ms
```

Esempio 17.13: *Rimozione di utente tramite* `deluser` *con rimozione della home directory associata*

in cui sarà cancellata la *home directory* dell'utente `ms`.
Per cancellare tutti i file del sistema appartenenti all'utente:

```
1 # deluser --remove-all-files ms
```

Esempio 17.14: *Rimozione completa dell'utente tramite* `deluser`

Per quest'ultimo comando `--remove-home` non ha effetto poiché la directory home e lo spool di posta sono già compresi dall'opzione `--remove-all-files`.

17.4.4 Modificare l'account utente e l'assegnazione dei gruppi

Solitamente l'account utente e i gruppi sono modificati editando i file `/etc/passwd` e `/etc/group`. È però più corretto utilizzare gli strumenti corretti, rispettivamente `usermod` e `groupmod`.

Il comando `usermod` accetta sostanzialmente le stesse opzioni di *useradd* ma cambierà le informazioni associate ad un account invece di crearlo. Per esempio, con:

```
1 # usermod -g group user
```

Esempio 17.15: *Cambiare* `GID` *tramite* `usermod`

si modificherà il *GID* associato ad un utente.
Per aggiungere l'utente `user` al gruppo `group`:

```
1 # usermod -a -G group user
```

Esempio 17.16: *Aggiungere un utente ad un gruppo tramite* `usermod`

in cui `-a` o `--append` aggiungerà la `user` al gruppo esistente.
Se il gruppo non esiste. v. 17.4.5.

È utile ricordare che un utente carica il proprio environment, quindi anche le informazioni relative ai gruppi di appartenenza, quando effettua la procedura di *login*, per cui un utente già loggato non disporrà di eventuali nuovi gruppi fino al successivo *login*. Potrà ovviare tramite il comando `newgrp` con queste alternative:

`newgrp` *nuovo_gruppo*
: in cui l'utente tenterà l'accesso al nuovo gruppo;

`newgrp -`
: (ri-)caricherà i gruppi dell'utente come se avvenisse tale fase della procedura di *login*.

17.4.5 Creazione, modifica e cancellazione di gruppi

Per creare gruppi si utilizzano due metodi alternativi:

- si aggiunge manualmente editando il file `/etc/group`, ma qualora sia utilizzata una password, si dovrà aggiungere anche una linea su `/etc/gshadow`;

- utilizzare i tool predisposti dal sistema.

Equivalentemente ai tool `useradd`, `usermod` e `userdel` esaminati per l'utente, esistono rispettivamente `groupadd`, `groupmod` e `groupdel` per la gestione dei gruppi.

Il comando `groupadd` si utilizza per creare gruppi con la seguente sintassi:

```
1 # groupadd [-g GID] groupname
```

Esempio 17.17: *Sintassi generica di* groupadd

in cui l'opzione `-g` è utilizzate per specificare un *GID*, se non è specificata sarà utilizzato un valore incrementale rispetto l'ultimo valore assegnato.
Analogamente al comando `adduser`, le distribuzioni *Debian-like* hanno il comando `addgroup`:

```
1 # addgroup group
```

Esempio 17.18: *Aggiunta di un gruppo tramite* `addgroup`

il quale aggiungerò il gruppo senza membri.

Il comando `groupmod` si utilizza per modificare gruppi con la seguente sintassi:

```
1 # groupmod [-g GID] [-n name] groupname
```

Esempio 17.19: *Sintassi generica di* groupmod

in cui:

`-g` è l'opzione utilizzate per specificare il *GID* se si vorrà modificarlo;

`-n` è l'opzione utilizzate per assegnare un nuovo nome al gruppo.

Il comando `groupdel` si utilizza per rimuovere gruppi con la seguente sintassi:

```
1 # groupdel groupname
```

Esempio 17.20: *Sintassi generica di* groupdel

in cui semplicemente si specificherà il numero del gruppo esistente da rimuovere. Anche in questo si vorrà verificare eventuali file orfani di gruppo e rimuoverli oppure modificare il gruppo assegnato tramite il comando `chgrp`[26].
Si utilizza il comando comando `gpasswd` si utilizza per modificare (o creare) la password assegnata ad un gruppo con la seguente sintassi:

```
1 # gpasswd [opzioni] groupname
```

Esempio 17.21: *Sintassi generica di* gpasswd

in cui le opzioni principali sono:

`-a` aggiungerà il membro al gruppo;

`-d` rimuoverà il membro dal gruppo;

`-A` sarà nominato un nuovo amministratore del gruppo.

[26] il comando sarà esaminato in 18.6 a pag. 240

17.5 Approfondimenti

17.5.1 Univocità di *UID* e *user*

Ovviamente per il sistema sarebbe sufficiente avere un unico campo univoco (quindi l'*UID*), ma è più difficile per gli utenti ricordare un numero e non un *username*. Sin dalla fase di *login* il sistema è conscio dei nostri limiti, quindi ad ogni interazione *uomo/sistema* ci concede l'inserimento di *user* associati (quindi campi testuali), ma poi il sistema lavorerà sempre con i corrispettivi *ID numerici*[27], e solo se ci sarà una interazione *sistema/uomo* in cui dovrà restituirci informazioni relativamente agli stessi *ID*, il sistema antitrasformerà da *ID numerico* al rispettivo *user* testuale. La riconversione *ID numerico* ⇒ *user* non sarà effettuata tipicamente nelle seguenti occasioni:

- se espressamente richiesto dall'utente la restituzione degli *ID numerici*; ad esempio si confronti l'output mostrato da:

```
1 $ ls -ld .*
2 drwxr-xr-x 20 ms   ms  4096 gen  8 21:47 .
3 drwxr-xr-x  5 root root  4096 ott 11 12:46 ..
4 -rw-------  1 ms   ms 27315 gen  2 14:33 .bash_history
5 -rw-r--r--  1 ms   ms   220 ott 11 12:46 .bash_logout
6 -rw-r--r--  1 ms   ms  3771 ott 11 12:46 .bashrc
7 -rw-------  1 ms   ms 19094 dic 27 15:59 .viminfo
8 -rw-------  1 ms   ms    89 ott 18 19:53 .vimrc
```

Esempio 17.22: *`ls -ld .*` della home*

  con il seguente:

```
1 $ ls -nd .*
2 drwxr-xr-x 20 1002 1002  4096 gen  8 21:47 .
3 drwxr-xr-x  5    0    0  4096 ott 11 12:46 ..
4 -rw-------  1 1002 1002 27315 gen  2 14:33 .bash_history
5 -rw-r--r--  1 1002 1002   220 ott 11 12:46 .bash_logout
6 -rw-r--r--  1 1002 1002  3771 ott 11 12:46 .bashrc
7 -rw-------  1 1002 1002 19094 dic 27 15:59 .viminfo
8 -rw-------  1 1002 1002    89 ott 18 19:53 .vimrc
```

Esempio 17.23: *`ls -nd .*` della home*

  in cui l'opzione `-n` o l'equivalente `--numeric-uid-gid` restituisce gli *UID* e *GID* associati, quindi non li converte nel corrispondente numerico;

- se il sistema non riuscirà ad effettuare la conversione da numero a testo, ad esempio se un account di un utente è stato cancellato ma non è stata cancellata l'*home* associata con i relativi file di cui ora non saranno più disponibili il testo[28].

17.5.2 Comando `id` in un sistema Desktop

In 17.24 a pag. 225 il comando `id` è stato digitato per un utente con pochi gruppi assegnati. Soltamente per un sistema desktop i gruppi saranno molti di più già in fase di installazione o con qualche applicativo installato:

[27] sia *UID* sia *GID*

[28] ancora peggio se ad esempio l'*UID* è stata riassegnata ad un'utenza, quindi ora i file con quell'*UID* saranno *eredidati* da un user differente

```
1 $ id
2 uid=1000(mariano) gid=1000(mariano) gruppi=1000(mariano),4(adm),24(cdrom)
      ,27(sudo),30(dip),46(plugdev),108(kvm),120(lpadmin),131(lxd),132(
      sambashare),138(libvirt),998(docker)
3 $ id -Gn
4 mariano adm cdrom sudo dip plugdev kvm lpadmin lxd sambashare libvirt
      docker
```

Esempio 17.24: UID *e* GID *tipicamente ad un utente desktop*

in cui l'opzione:

`-G`
: mostra solo i gruppi secondari;

`-n`
: mostra la conversione testuale al posto del *GID*.

17.6 Concetti chiave

GID: ***g****roup* ***ID***

UID: ***u****ser* ***ID***

shadow password: file contenente le password degli utenti di sistema

17.7 Comandi introdotti nel capitolo

`addgroup` ⇒ aggiunge un gruppo
`adduser` ⇒ crea un nuovo utente
`chfn` ⇒ cambia il nome dell'utente e altre informazioni nel *GECOS field*
`chsh` ⇒ cambia la shell di login
`chgrp` ⇒ v. 18.6 a pag. 240
`deluser` ⇒ crea un nuovo utente
`id` ⇒ mostra gli *user* e *group ID*
`groupadd` ⇒ crea un nuovo gruppo
`groupdel` ⇒ rimuove un gruppo
`groupmod` ⇒ modifica la definizione di un gruppo di sistema
`groups` ⇒ fornisce l'elenco dei gruppi degli *user* forniti
(se non è fornito nessun user, l'elenco si riferirà all'utente stesso)
`newgrp` ⇒ effettua l'accesso ad un nuovo gruppo
`passwd` ⇒ cambia la password utente
`useradd` ⇒ crea un nuovo utente ma è una *low level utility* (v. `adduser`)
`userdel` ⇒ rimuove un utente ma è una *low level utility* (v. `deluser`)
`usermod` ⇒ modifica l'account di un utente

Capitolo 18

Access control

POICHÉ l'accesso al medesimo sistema può essere effettuato da utenza diversa, è necessario che sia definito un sistema di **access control** per i processi, i file e le directory per assicurare che ciascuna risorsa sia privata e, allo stesso tempo, sia fornito un meccanismo affinché, qualora il proprietario lo desideri, la propria risorsa sia accedibile per altra utenza secondo modalità definite. Per questo scopo, i sistemi GNU/Linux implementano il *sistema di privilegi Unix* parzialmente accennato nel precedente capitolo.

18.1 Introduzione all'*access control system*

In accordo con il mondo Unix, ogni file (e directory) è assegnato ad un utente (definito *proprietario*) e ad un gruppo, e per ciascuno di essi sono assegnati separatamente privilegi per il proprietario, per i membri del gruppo[1] e *agli altri*[2].
Tecnicamente sono previsti privilegi per l'accesso di tipo **read**, **write** e **execute** individualmente per ciascuna risorsa e distintamente per le classi di utenza citata. L'**owner** della risorsa assegna l'**access mode**, cioè conferisce i privilegi per le altre classi e, se vorrà, li potrà modificare in ogni momento.
Il sistema così definito permette al proprietario di creare *mappe* per condividere l'accesso ad eventuali risorse o per eventuali collaborazioni con gruppi.

18.2 Mappaggio dei permessi

Come accennato, per ciascun file e directory il sistema permette di definire diritti di accesso separati per le tre classi di utenza (*owner*, *group* e *others*). I principali permessi sono *lettura*, *scrittura* ed *esecuzione*.
Per comprenderne il meccanismo di definizione, si esegua un `ls -l` in cui si otterrà un output ovviamente diverso ma simile nelle colonne:

```
1 drwxrwxr-x 2 ms ms 4096 mar  7 16:34 hello_worlds
2 -rwxrwxr-x 1 ms ms 39 dic 18 19:31 hello_world.sh
3 -rw-rw-r-- 1 ms ms 14 mar  7 16:33 hello_world.txt
```

[1] per brevità da qui in avanti *al gruppo*
[2] identifica dal resto dell'utenza che non sia identificata la precedente

Esempio 18.1: *Esecuzione di* `ls -l` *per l'interpretazione dei permessi associati alle risorse*

in cui fin qui sono state esaminate tutte le colonne ad esclusione della prima che di seguito è per chiarezza espositiva divisa in due blocchi:

```
1 d rwxrwxr-x
2 - rwxrwxr-x
3 - rw-rw-r--
```

Esempio 18.2: *Prima colonna precedente suddivisa in due blocchi*

in cui il primo indica la tipologia di file[3] e il secondo si riferisce ai permessi associati alla risorsa. Questi ultimi sono riportati nuovamente, anche qui per chiarezza suddivisi a loro volta nei blocchi costituenti:

```
1 rwx rwx r-x
2 rwx rwx r-x
3 rw- rw- r--
```

Esempio 18.3: *Blocco dei permessi suddiviso per la categoria di utenti a cui si riferiscono*

in cui i permessi del:

1^o blocco sono riferiti all'*owner*[4];

2^o blocco sono riferiti al *group*[5];

3^o blocco sono riferiti a *others*[6].

Ciascun blocco precedente è generalizzabile dalla codifica `rwx` in cui:

`r` indica **read permission** della risorsa associata;

`w` è il **write permission**;

`x` è l'**execute permission**;

`-` indica, relativamente alla posizione occupata, la mancanza del permesso associato.

Ad esempio, i blocchi di seguito indicheranno:

`rwx`
: i permessi di lettura, scrittura e esecuzione della risorsa;

`rw-`
: i permessi di lettura e scrittura, ma non esecuzione;

`r-x`
: i permessi di lettura ed esecuzione, ma non scrittura;

`-wx`
: i permessi di scrittura ed esecuzione, ma non di lettura;

[3] v. **??** a pag. **??**

[4] riportato nell'***Esempio 18.1***, `ms`

[5] riportato nell'***Esempio 18.1*** dopo l'*owner*, `ms`

[6] non riportato per ovvie ragioni poiché si riferisce a tutti ad esclusione dei precedenti blocchi *owner* e *group*

`r--`
: solo il permesso di lettura;

`-w-`
: solo il permesso di scrittura;

`--x`
: solo il permesso di esecuzione;

`---`
: nessun permesso;

sono comunque da riferire alla classe di utenza associata (*owner* o *group* o *others*).
Quindi, i blocchi di permessi di seguito (riportati anche qui con la fittizia suddivisione per chiarezza espositiva) indicano:

`rwx rwx r-x`

: l'*owner* ha permesso di lettura, scrittura ed esecuzione;
: il *group* ha permesso di lettura, scrittura ed esecuzione;
: gli *others* hanno esclusivamente i permessi di lettura ed esecuzione;

`rw- rw- r--`

: l'*owner* ha solo permesso di lettura e scrittura;
: il *group* ha solo permesso di lettura e scrittura;
: gli *others* hanno solo permesso di lettura;

`rwx r-- ---`

: l'*owner* ha permesso di lettura, scrittura ed esecuzione;
: il *group* ha solo permesso di lettura;
: gli *others* non hanno permessi associati;

`rwx --x ---`

: l'*owner* ha permesso di lettura, scrittura ed esecuzione;
: il *group* ha solo permesso di esecuzione;
: gli *others* non hanno permessi associati;

`rw- --- ---`

: l'*owner* ha permesso di lettura e scrittura;
: il *group* non ha permessi associati;
: gli *others* non hanno permessi associati.

I permessi finora sono stati definiti tramite lettere e tale rappresentazione è denominata *rappresentazione simbolica*, in cui anche l'`owner`, il `group` e gli `others` sono rispettivamente identificati dalle lettere `u`, `g` e `o`.

18.2.1 Rappresentazione numerica

In alternativa alla precedente si utilizza la *rappresentazione numerica* in base ottale[7] secondo la seguente corrispondenza:

`r`	$\longleftrightarrow$	4
`w`	$\longleftrightarrow$	2
`x`	$\longleftrightarrow$	1

quindi sommando i permessi associati ad una singola classe di utenza si ha:

`rwx`	$\longleftrightarrow$	4 + 2 + 1 = 7
`rw-`	$\longleftrightarrow$	4 + 2 + 0 = 6
`r-x`	$\longleftrightarrow$	4 + 0 + 1 = 5
`r--`	$\longleftrightarrow$	4 + 0 + 0 = 4
`-wx`	$\longleftrightarrow$	0 + 2 + 1 = 3
`-w-`	$\longleftrightarrow$	0 + 2 + 0 = 2
`--x`	$\longleftrightarrow$	0 + 0 + 1 = 1
`---`	$\longleftrightarrow$	0 + 0 + 0 = 0

e riprendendo gli esempi sella sezione precedente (ma qui senza separazione):

`rwxrwxr-x`	$\longleftrightarrow$	775
`rw-rw-r--`	$\longleftrightarrow$	664
`rwxr-----`	$\longleftrightarrow$	740
`rwx--x---`	$\longleftrightarrow$	710
`rw-------`	$\longleftrightarrow$	600

in cui:

la 1^a cifra è la risultante dei permessi associati all'*owner*;

la 2^a cifra è relativa ai permessi associati al *group*;

la 3^a cifra sono i permessi associati agli *others*.

18.3 Accesso a file e directory

Per quanto concerne i file, tali permessi sono banalmente intuibili con l'azione che suggeriscono. In particolare per un file:

read permission: necessario per l'*accesso al contenuto* del file, quindi ad esempio saranno utilizzabili i tool come `cat` o `less`;

write permission: necessario per la *modifica del contenuto* del file, quindi ad esempio si potrà aggiungere del contenuto con gli operatori di redirezione della shell (> e >>) o modifica tramite un editor;

[7] cosicché ogni cifra finale corrisponderà alla sintesi dei permessi complessivi assegnati ad una categoria di utenza

execute permission: per l'*esecuzione del file*, quindi lo stesso file sarà un programma

Occorre chiarire il permesso di esecuzione, poiché esso è sufficiente per l'esecuzione di un *binario*[8], ma per l'esecuzione di uno script (il quale dovrà essere letto dal relativo *interprete*) è necessario anche il permesso in lettura.
Per le directory, i diritti di accesso sono meno intuibili a meno che non si focalizzi che la directory non è altro che una tipologia di file contenente (tra gli altri dati) i *nomi dei file* e gli *inode* relativi. Per una directory:

read permission: necessario per l'*accesso al contenuto* della directory, quindi ad esempio poter ottenere la lista dei file contenuti nella stessa directory eseguendo il comando `ls`;

write permission: necessario per la *creazione*, *cancellazione* o *rinomina* dei file nella directory, quindi ad esempio poter utilizzare i comandi `touch`, `rm` e `mv`;

execute permission: per l'accesso ai *metadati*, essenziali per *utilizzare* la directory, nel senso di entrare alla stessa tramite il comando `cd`, oppure utilizzare il *pathname* di un file che contiene nel percorso la directory medesima.

La non immediata comprensione dei permessi precedenti applicati alle directory è esemplificata dai seguenti esempi non intuitivi:

no read permission: non sarà possibile ottenere la lista dei file, ma è possibile accedere ai file della directory conoscendo a priori il nome;

no write permission: non potendo modificare la tabella degli *inode*, non è permessa la rimozione di un file tramite il comando `rm`, ma è possibile la cancellazione del contenuto tramite il comando `echo '' > ...`[9];

no execute permission: è possibile ottenere la lista dei file della directory tramite il comando `ls`, ma non è possibile accedere alle metainformazioni associati al file, quindi non sarà possibile ottenere informazioni estese sui file tramite il comando `ls -l`. Inoltre non saranno possibili ulteriori operazioni, ad esempio l'aggiunta di file alla directory (nonostante l'eventuale *write permission*) perché l'accesso ai metadata della directory è essenziale. Per tale ragione le directory generalmente hanno sempre attivo il permesso di esecuzione (per l'*owner*, *group* e *others*) qualora siano associati altri permessi alla medesima classe di utenza.

18.3.1 Esempi su file e directory

È utile fornire un esempio per chiarire i permessi fin qui esaminati. Si abbia la casistica illustrata in ***Figura 18.1*** dal comando `tree -p`:
Poiché l'attività di questa sezione sarà eseguita dall'*owner*, è utile mentalmente isolare i permessi relativi allo stesso utente, quindi i primi tre caratteri simbolici relativi ai permessi mostrati (o equivalentemente la prima cifra).
Iniziamo dai file a livello base, inviati in output con i relativi permessi tramite il `find`:

```
1 $ find .  -maxdepth 1 -type f -printf "%f perm="%M/"perm="%m/\\n
2 nowrite.txt perm=-r-x------/perm=500/
3 noexecute.sh perm=-rw-------/perm=600/
4 noread.txt perm=--wx------/perm=300/
```

[8] un programma scritto in *linguaggio macchina*
[9] in cui al posto dei puntini ci saranno i file da azzerare

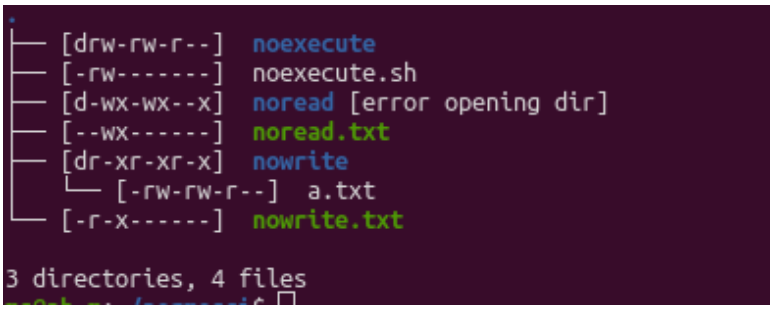


Figura 18.1: *Esempio di output del comando* `tree -p`

Esempio 18.4: *Lista dei file tramite* `find` *senza scendere di livello*

Per il file `noread.txt`:

```
1 $ cat noread.txt
2 cat: noread.txt: Permesso negato
3 $ echo 'Hello world' > noread.txt
4 $
```

Esempio 18.5: *Esempi con il file* `noread.txt`

non è stato possibile leggere da file, ma è stato possibile scrivere sullo stesso.
Per il file `nowrite.txt`:

```
1 $ cat nowrite.txt
2 Hello world
3 $ echo 'Hello' > nowrite.txt
4 -bash: nowrite.txt: Permesso negato
```

Esempio 18.6: *Esempi con il file* `nowrite.txt`

è stato possibile leggere da file, ma non è stato possibile scrivere sullo stesso.
Per il file `noexecute.sh`:

```
1 $ cat noexecute.sh
2 #!/usr/bin/bash
3
4 echo 'Hello world'
5 $ echo "echo 'Reply> Hello'" >> noexecute.sh
6 $ ./noexecute.sh
7 -bash: ./noexecute.sh: Permesso negato
8 $ bash noexecute.sh
9 Hello world
10 Reply> Hello
```

Esempio 18.7: *Esempi con il file* `noexecute.sh`

è stato possibile leggere da file, è stato possibile scrivere sullo stesso, ma non è possibile l'esecuzione diretta. Essendo uno script con permesso di lettura, si può ovviare eseguendo l'interprete indicando con argomento il file, cosicché l'interprete leggerà il file specificato in argomento.

Analogamente alla trattazione dei file, anche le directory mostrate in ***Figura 18.1*** sono inviate in output con i relativi permessi:

```
1 $ find .  -mindepth 1 -type d -printf "%f perm="%M/"perm="%m/\\n
      2>/dev/null
2 noread perm=d-wx-wx--x/perm=331/
3 noexecute perm=drw-rw-r--/perm=664/
4 nowrite perm=dr-xr-xr-x/perm=555/
```

Esempio 18.8: *Lista delle directory tramite* `find` *senza scendere di livello*

Anche se nella stessa immagine non è visibile[10], ciascuna directory contiene un file `a.txt` vuoto.
Per la directory `noread`:

```
1 $ ls noread
2 ls: impossibile aprire la directory 'noread': Permesso negato
3 $ ls -l noread/a.txt
4 -rw-rw-r-- 1 ms ms 0 mar  9 15:02 noread/a.txt
5 $ cat noread/a.txt
6 $ echo 'Hello world' > noread/a.txt
7 $ cd noread
8 noread$ touch b.txt
9 noread$ mv b.txt c.txt
10 noread$ rm c.txt
```

Esempio 18.9: *Esempi con la directory* `noread`

non sarà possibile ottenere la lista dei file tramite `ls`, ma conoscendo a priori il nome di file contenuti, sarà possibile accedere alle relative metainformazioni, al relativo contenuto e modificarlo. Inoltre è possibile accedere alla directory, creare file, rinominarli e cancellarli.
Per la directory `nowrite`:

```
1 $ ls -l nowrite
2 -rw-rw-r-- 1 ms ms 0 mar  6 18:12 a.txt
3 $ cat nowrite/a.txt
4 $ echo 'Hello world' > nowrite/a.txt
5 $ cd nowrite
6 nowrite$ touch b.txt
7 touch: impossibile fare touch di 'b.txt': Permesso negato
8 nowrite$ mv a.txt b.txt
9 mv: impossibile spostare 'a.txt' in 'b.txt': Permesso negato
10 nowrite$ rm a.txt
11 rm: impossibile rimuovere 'a.txt': Permesso negato
12 nowrite$ cat /dev/null > a.txt
```

Esempio 18.10: *Esempi con la directory* `nowrite`

è possibile ottenere la lista dei file e metainformazioni associate, è possibile accedere al contenuto dei file e modificarlo. È possibile accedere alla directory, ma non è stato possibile modificarne le metainformazioni (quindi creare file, rinominarli o cancellarli), ma è possibile la cancellazione del contenuto.
Per la directory `noexecute`:

```
1 $ ls noexecute
2 ls: impossibile accedere a 'noexecute/a.txt': Permesso negato
3 a.txt
```

[10] ovviamente a causa dei permessi che saranno esplicitati nel seguito

```
$ ls -l noexecute 2>/dev/null
totale 0
-????????? ? ? ? ?            ? a.txt
$ cat noexecute/a.txt
cat: noexecute/a.txt: Permesso negato
$ echo 'Hello world' > noexecute/a.txt
-bash: noexecute/a.txt: Permesso negato
$ cd noexecute
-bash: cd: noexecute: Permesso negato
$ touch noexecute/b.txt
touch: impossibile fare touch di 'noexecute/b.txt': Permesso negato
$ mv noexecute/a.txt noexecute/b.txt
mv: failed to access 'noexecute/b.txt': Permesso negato
$ rm noexecute/a.txt
rm: impossibile rimuovere 'noexecute/a.txt': Permesso negato
```

Esempio 18.11: *Esempi con la directory* `noexecute`

è possibile ottenere la lista dei file, ma non delle metainformazioni associate, non è possibile accedere al contenuto dei file o modificarlo. Non accedendo alle metainformazioni, non è possibile accedere alla directory, non è stato possibile creare file, rinominarli o cancellarli. In sintesi, senza permesso di esecuzione alla directory, è possibile solo averne la lista

18.4 Visualizzazione dei permessi

Nella sezione precedente si sono ottenute informazioni relativamente ai permessi su file e directory tramite il comando `ls -l` o utilizzando il `find` attivando il *flag* `-printf` con argomenti opportuni. Nel seguente esempio si confronteranno gli output di `ls -l` e del comando `stat`[11]:

```
$ ls -l noexecute*
d-wx-wx--x 2 ms ms 4096 mar 11 12:08 noread
--wx------ 1 ms ms   12 mar  9 15:22 noread.txt
$ stat noexecute*
  File: noexecute
  Dim.: 4096        Blocchi: 8          Blocco di IO: 4096   directory
Device: 806h/2054d  Inode: 13378068    Coll.: 2
Accesso: (0664/drw-rw-r--)  Uid: ( 1002/      ms)   Gid: ( 1002/      ms)
Accesso  : 2024-03-11 12:22:09.836309402 +0100
Modifica : 2024-03-06 18:07:09.878503894 +0100
Cambio   : 2024-03-09 14:51:44.310995161 +0100
Creazione: -
  File: noexecute.sh
  Dim.: 56          Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 806h/2054d  Inode: 13375485    Coll.: 1
Accesso: (0600/-rw-------)  Uid: ( 1002/      ms)   Gid: ( 1002/      ms)
Accesso  : 2024-03-11 11:50:57.951318315 +0100
Modifica : 2024-03-09 15:20:38.331072550 +0100
Cambio   : 2024-03-09 15:20:38.335072570 +0100
Creazione: -
```

Esempio 18.12: *Esecuzione di* `ls -ld noexecute*` *e* `stat noexecute*`

[11] sarà nuovamente utilizzato in **??** a pag **??**

eseguiti sia su file sia su directory.
Si nota che per l'output di `stat`, per entrambe le risorse alla $4^a riga$[12] sono specificati i permessi tramite *rappresentazione simbolica* e *numerica*, l'*user* e il *gruppo* con relativi *UID* e *GID*.
Si nota che i permessi numerici specificano entrambi uno 0 iniziale, ciò indica la mancanza di *permessi speciali*[13].

18.5 Cambio dei permessi

Sintetizzando quanto fin qui esaminato:

rappresentazione simbolica per una categoria di utenza è la tripletta è `rwx`, con `r` per la lettura, `w` per la scrittura e `x` per l'esecuzione, ciascuno sostituito con il simbolo `-` nella rispettiva posizione per indicare la mancanza del relativo permesso. Queste triplette sono riportare per le tre categorie di utenze rispettivamente *proprietario*, *gruppo* e *altri*.

rappresentazione numerica per ciascuna categoria di utenza il valore numerico da associare è costituito dalla somma del permesso di lettura (4), permesso di scrittura (2) e permesso di esecuzione (1), tale valore definito per ciascuna categoria di utenza. Il valore finale sarà compreso da una rappresentazione ottale che va da 000_8 a 777_8 i quali rispettivamente indicando *nessun privilegio* e *tutti i privilegi* associati alla risorsa.

Per cambiare i permessi associati alla risorsa il proprietario[14] utilizzerà il comando `chmod`[15] come di seguito:

```
1 $ chmod permesso risorsa
```

Esempio 18.13: *Sintassi generica del comando* `chmod`

in cui saranno da sostituire:

permesso
: con quello da assegnare (eventualmente in *forma differenziale* rispetto allo stato attuale), potendo utilizzare sia l'espressione simbolica sia numerica[16];

risorsa
: indica un file o directory (per quest'ultima è possibile operare ricorsivamente attivando il *flag* `-R`).

Per una trattazione completa del comando, si rimanda al manuale[17].

[12] le altre righe sono altre metainformazioni esaminate nel capitolo **??**
[13] i quali saranno argomentati in 18.8 a pag. 241
[14] ovviamente anche l'utente root
[15] ***ch****ange file* ***mo****de bits*
[16] si prosegua nelle prossime sezioni
[17] `man chmod`

18.5.1 Metodo simbolico

Data la triplice tripletta $\texttt{rwx}_u\texttt{rwx}_g\texttt{rwx}_o$, in cui i pedici sono stati aggiunti per esplicitare la categoria di utenza di riferimento, al comando `chmod` il *permesso* sarà esplicitato dall'espressione[18]:

```
[ugoa]*[-+=][perm]
```

in cui la precedente sequenza può essere suddivisa nelle sottosequenze di seguito:

`ugoa`
: indica rispettivamente `u` l'*user owner*, `g` il *group*, `o` gli *others* e `a` per all users (se la sottoespressione è omessa, di default sarà `a` a meno dei bit indicati tramite *umask* (v. **??** a pag. **??**));

`-+=`
: indica rispettivamente − per *eliminare* il *bit* indicante il permesso, + per *aggiungere* tale bit, = per aggiungere tale bit e eliminare quelli non menzionati[19];

`perm`
: formata da una sequenza che esprime i bit da attivare/disattivare, espressa dalle lettere `rwx` in cui saranno opzionalmente espressi `r` per `read`, `w` per `write`, `x` per *execute*.

Le principali forme utilizzate sono di seguito:

\+ per aggiungere il permesso, quindi sarà:

`+r` oppure `a+r`
: per aggiungere il permesso in *lettura* per *all users*, oppure per restringere l'utenza:

  `u+r`
  : in lettura solo per l'*user owner*;

  `g+r`
  : in lettura solo al *group*;

  `o+r`
  : in lettura solo agli *others*;

`+w` oppure `a+w`
: per aggiungere il permesso in *scrittura* per *all users*, oppure per restringere l'utenza:

  `u+w`
  : al solo *user owner*;

  `g+w`
  : al solo *group*;

  `o+w`
  : agli *others*;

`+x` oppure `a+x`
: per aggiungere il permesso di *esecuzione* per *all users*, oppure per restringere l'utenza:

[18] qui semplificata, il comando prevede un'espressività maggiore

[19] eccetto eventuali *permessi speciali* su directory (v. 18.8 a pag. 241

`u+x`
al solo *user owner*;

`g+x`
al solo *group*;

`o+x`
agli *others*;

– per eliminare il permesso, quindi sarà:

`-r` oppure `a-r`
per eliminare il permesso in *lettura* per *all users*, oppure per restringere l'utenza:

`u-r`
in lettura solo per l'*user owner*;

`g-r`
in lettura solo al *group*;

`o-r`
in lettura solo agli *others*;

`-w` oppure `a-w`
per eliminare il permesso in *scrittura* per *all users*, oppure per restringere l'utenza:

`u-w`
al solo *user owner*;

`g-w`
al solo *group*;

`o-w`
agli *others*;

`-x` oppure `a-x`
per eliminare il permesso di *esecuzione* per *all users*, oppure per restringere l'utenza:

`u-x`
al solo *user owner*;

`g-x`
al solo *group*;

`o-x`
agli *others*;

Le sequenze appena elencate sono minimali poiché possono essere concatenate per esprimere:

- più categorie di utenza con permesso omogeneo (ad esempio `go-x`);
- più permessi ad una stessa categoria (ad esempio `o+rwx`);
- più categorie con permessi omogenei[20] (ad esempio `og-rwx`);
- più sequenze dei precedenti punti separati dal carattere `,` (ad esempio `g+x,o-rw`).

Di seguito alcuni esempi:

[20] una composizione additiva delle due precedenti

`ug+x`
: per aggiungere i permessi di esecuzione sia per il proprietario sia per il gruppo;

`o-wx`
: per rimuovere i permessi di scrittura ed esecuzione per la categoria *altri*;

`ug+rwx`
: per aggiungere i permessi di lettura, scrittura ed esecuzione sia per il proprietario sia per il gruppo;

`g+rx,o-rx`
: per aggiungere i permessi di lettura ed esecuzione al gruppo ed eliminarli per la categoria *altri*;

`u+rwx,g+rx,o+x`
: per aggiungere i permessi di lettura, scrittura ed esecuzione al proprietario e lettura ed esecuzione al gruppo, solo esecuzione agli altri.

18.5.2 Metodo numerico

Ricordando quanto riportato in 18.2.1 a pag. 230, al comando `chmod` il *permesso* sarà esplicitato dall'espressione[21]:

```
[-+=][perm]
```

in cui la precedente sequenza può essere suddivisa nelle sottosequenze di seguito:

`-+=`
: indica rispettivamente − per *eliminare* il *bit* indicante il permesso, + per *aggiungere* tale bit, = per aggiungere tale bit e eliminare quelli non menzionati[22] (se non espresso, quest'ultimo simbolo sarà il default);

`perm`
: formata da una sequenza di cifre che esprime i bit da attivare/disattivare con *leading zeros*[23].

Senza ripetere quanto già scritto, alcuni esempi:

`700` (equivalentemente (`=700`))
: assegnerà diritti pieni al proprietario, nessun diritto al resto dell'utenza;

`664` (equivalentemente (`=644`))
: assegnerà diritti di lettura e scrittura sia al proprietario sia al gruppo, diritti di lettura al resto dell'utenza;

`755` (equivalentemente (`=755`))
: assegnerà diritti pieni al proprietario, diritti di lettura ed esecuzioni al resto dell'utenza;

`+100`
: aggiungerà il diritto di esecuzione al proprietario, il resto immutato;

[21] qui semplificata, il comando prevede un'espressività maggiore
[22] eccetto eventuali *permessi speciali* su directory (v. 18.8 a pag. 241
[23] le cifre non espresse saranno sostituite da sinistra con 0

`+60`
: aggiungerà i diritti di lettura e scrittura al gruppo;

`-17`
: eliminerà il diritto di esecuzione al gruppo, tutti i permessi agli *altri*;

`+11`
: aggiungerà i diritti di esecuzione al gruppo e agli *altri*;

`-1`
: eliminerà il diritto di esecuzione agli *altri*;

18.5.3 Esempio di cambio di permessi su file e directory

Creiamo ad esempio il file `script.sh` tramite *Here document*[24] e la directory `scripts`, poi visioniamo i permessi associati alle risorse appena create:

```
1 $ cat > script.sh << EOF
2 #!/bin/bash
3
4 echo 'Hello world'
5 EOF
6 $ mkdir scripts
7 $ ls -ld script*
8 drwxrwxr-x 2 ms ms 4096 mar 12 12:07 scripts
9 -rw-rw-r-- 1 ms ms 32 mar 12 12:03 script.sh
```

Esempio 18.14: *Creazione di `script.sh` e `scripts`*

in cui notiamo che i permessi associati dalla creazione[25] alla directory sono `rwxrwxr-x` e al file sono `rw-rw-r--`. Potremmo volere che:

1. solo il proprietario abbia la facoltà di aggiungere script alla directory `scripts` (al momento la `w` è associata anche al gruppo);
2. solo il proprietario possa modificare lo script, ma possano eseguirlo lui e il gruppo (attualmente manca il permesso `x` associato alle due utenze).

Affinché solo il proprietario possa aggiungere file in `scripts` si può esplicatemente *eliminare il permesso di scrittura al gruppo*:

```
1 $ chmod g-w scripts
2 -- oppure --
3 $ chmod 755 scripts
4 $ ls -ld scripts
5 drwxr-xr-x 2 ms ms 4096 mar 12 12:07 scripts
```

Esempio 18.15: *Eliminazione del permesso di scrittura al gruppo*

Affinché solo il proprietario possa modificare lo script, ma possano eseguirlo sia lui sia il gruppo:

[24] procedura in 8.2.3 a pag. 94

[25] quelli riportati non sono generalizzabili poiché potrebbero variare con la distribuzione, l'importante è la procedura che segue

```
1 $ chmod g-w,ug+x script.sh
2 -- oppure --
3 $ chmod 754 scripts.sh
4 $ ls -l script.sh
5 -rwxr-xr-- 1 ms ms 32 mar 12 12:03 script.sh
```

Esempio 18.16: *Eliminazione del permesso di scrittura al gruppo e aggiunta dell'esecuzione sia al proprietario sia al gruppo*

18.6 Cambio di owner e group

Il comando `chown`[26] permette di cambiare l'*owner* e il *group* di file o directory. Ovviamente per non incorrere in problemi nei cambi id owner e gruppo sarà necessario essere *superutente*.
La sintassi[27] è:

```
1 $ chown [OPTION] [OWNER][:[GROUP]] FILE
```

Esempio 18.17: *Sintassi generica del comando* `chown`

in cui alla risorsa indicata con `FILE` saranno assegnati i parametri:

`OWNER`
: indica l'eventuale *owner* da assegnare;

`GROUP` indica l'eventuale *group* da assegnare

Ad esempio sono espressioni valide:

```
1 $ chown syslog ...
2 $ chown :www-data ...
3 $ chown syslog:adm ...
```

Esempio 18.18: *Esempi di* `chown`

in cui il primo comando assegnerà il proprietario `syslog`, il secondo il gruppo `www-data` ed il terzo l'owner `syslog` e il gruppo `adm`.
Per tale comando l'opzione più frequentemente utilizzata è `-R`[28] eventualmente utilizzate per le directory, nel qual caso eseguirà ricorsivamente su file e directory.

Nell'***Esempio 18.17***, il secondo comando `chown` cambierà solo il gruppo lasciano l'utente immutato. Equivalentemente si sarebbe potuto utilizzare il comando il comando `chgrp`[29].
Anche per questo comando, l'opzione più frequentemente utilizzata è `-R`[30] eventualmente utilizzate per le directory, nel qual caso eseguirà ricorsivamente su file e directory.

[26] ***ch****ange file* ***own****er and group*
[27] ovviamente si consulti il manuale tramite `man chown` per ottenere la generica sintassi
[28] equivalente a `--recursive`
[29] ***ch****ange* ***group*** *ownership*
[30] equivalente a `--recursive`

18.7 L'ownership dei processi e nuovi permessi[31]

Nel sistema non soltanto le risorse su storage hanno l'owner, ma anche i processi[32]. Di seguito riporto un esempio:

```
1 $ ps -O user,group
2     PID USER     GROUP    S TTY          TIME COMMAND
3   25286 ms       ms       S pts/2    00:00:01 -bash
4  145891 ms       ms       R pts/2    00:00:00 ps -O user,group
```

Esempio 18.19: *Esempi di* `ps` *con output personalizzato*

in cui ci si è limitati alla visualizzazione dei processi del terminale.
Più in generale, durante l'attività molti processi risiedono in memoria e il sistema li proteggerà vicendevolmente. Ciascun processo, con i propri dati che risiederanno nel *virtual address space*, sono assegnati ad un proprietario e ad un gruppo. Di base il proprietario e il gruppo coincideranno con l'utente e il gruppo primario di colui che ha eseguito il programma, ma ci sono molti casi in cui ciò non è sufficiente. Si pensi all'attività di *cambio password*[33] in cui un utente ordinario potrà cambiare la propria password e, se l'operazione andrà a buon fine, dovrà essere modificato il file `/etc/shadow` per il quale lo stesso utente non avrà nessun permesso. Sinteticamente il comando `passwd` dovrà riassegnare l'ownership affinché possa modificare tale file.
Più in generale i processi creati per eseguire compiti di amministrazione dovranno riassegnare l'ownership. Tra questi alcuni meccanismi sarano presentati nella prossima sezione.

18.8 Permessi speciali

Finora si sono esaminati i permessi di lettura, scrittura ed esecuzione. Il panorama dei permessi è però più ampio e, per taluni file o directory, saranno utilizzati nuovi permessi esaminati nel seguito.

18.8.1 I permessi speciali per file

Per i file esistono due permessi speciali: **SUID bit** e **SGID bit**.

SUID bit

Si è accennato in 18.7 che alcuni processi hanno necessità di riassegnare l'ownership. Per esempio, si esamini l'utility `passwd` che può essere invocata da un utente generico, ma dovrà modificare un file per il quale l'utente ordinario non ha i permessi, quindi quando il tool sarà eseguito dovrà riassegnare l'ownership per poter procedere a tale modifica.
Verificando i permessi assegnati all'applicativo:

```
1 $ which passwd
2 -rwsr-xr-x 1 root root 68208 feb  6 13:49 /usr/bin/passwd
```

Esempio 18.20: *Verifica del tool* `passwd`

[31] il contenuto della sezione è semplificato per rendere la trattazione più agevole. Il procedimento sarà esaminato nuovamente però in maniera più rigorosa in 18.10.1 a pag. 248

[32] v. 16.5 a pag. 198

[33] v. 17.4.2 a pag. 220

in cui per il proprietario è apparso il carattere `s` in corrispondenza del permesso di esecuzione. Esso identifica il **set-UID bit**[34] il quale riassegnerà l'owner del processo creato al proprietario del file.
In questo caso la riassegnazione dell'owner corrisponderà all'utente `root`, il quale potrà effettivamente modificare il file `/etc/shadow` avendo permesso di scrittura sullo stesso.

Ovviamente tale riassegnazione apre una questione di sicurezza che dev'essere affrontata dal creatore, per l'esempio suddetto il tool è progettato per cambiare le password di ciascun utente se invocato dall'amministratore, quindi dovrà assicurarsi che l'utente ordinario cambi esclusivamente la propria password.
In generale, adottare il meccanismo della riassegnazione dell'owner implicherà la grande responsabilità dell'autore dell'applicativo nell'assicurarsi che l'utente ordinario non possa danneggiare altri utenti o addirittura il sistema.
Connesso a quest'ultimo passaggio, nei sistemi GNU/Linux il meccanismo di *set-UID* è attivabile sono per i binari e non per gli script[35].
Per attivare il *SUID bit* si veda l'esempio nella prossima sezione.

SGID bit

Analogamente al precedente, esiste il **set-GID bit**[36] il quale implicherà che il processo eseguito avrà assegnato il gruppo della risorsa e i privilegi che ne corrisponderanno (tipicamente accesso a file assegnati al gruppo).
Per attivare tali permessi si potrà utilizzare il comando `chmod` equivalentemente a quanto effettuato per i permessi ordinari[37], quindi:

```
1 chmod u+s ...
2 chmod g+s ...
```

Esempio 18.21: *`chmod` per* SUID *e* SGID *in modalità simbolica*

in cui:

la 1^a linea aggiungerà il *SUID bit*;

la 2^a linea aggiungerà l'*SGID bit*.

Eseguendo un `ls -l` del file, si otterrà:

```
1 -rwxrwxr-x ...
2 -rwsrwxr-x ...
3 -rwxrwsr-x ...
4 -rwsrwsr-x ...
```

Esempio 18.22: *`ls -l` per verificare* SUID *e* SGID *con tipico carattere `s`*

in cui la 1^a linea mostra un generico permesso alla risorsa iniziale, per differenza:

la 2^a linea mostra il carattere `s` al posto del carattere `x` per il proprietario poiché è stato attivato il *SUID bit* tramite `chmod u+s`;

[34] per contrazione **SUID bit**
[35] per approfondimenti [FAQ] e [SK]
[36] per contrazione **SGID bit**
[37] v. 18.5 a pag. 235

la 3^a mostra il carattere `s` al posto del carattere `x` per il group poiché è stato attivato l'*SGID bit* tramite `chmod g+s`;

la 4^a mostra i caratteri `s` al posto dei caratteri x per il proprietario e per il gruppo poiché sono attivi contemporaneamente sia il *SUID bit* sia il *SGID bit* tramite `chmod ug+s`.

Non è frequente ma il carattere `s` si potrebbe presentare con `S` come di seguito:

```
1 -rw-rw-r-- \dots
2 -rwSrw-r-- \dots
3 -rw-rwSr-- \dots
4 -rwSrwSr-- \dots
```

Esempio 18.23: *`ls -l` per verificare* SUID *e* SGID *con carattere `S`*

quasi equivalente all'***Esempio 18.22***, in cui però la risorsa iniziale[38] non aveva permessi di esecuzione né per il proprietario né per il gruppo. Quindi sono stati attivati alla risorsa alla prima linea rispettivamente *SUID bit*, *SGID bit* e entrambi, quindi il carattere `S` espliciterà il relativo *SUID bit* o *SGID bit* senza permesso di esecuzione per la relativa categoria di utenza.
È possibile esprimere il permesso espresso nell'argomento del comando `chmod` utilizzando la rappresentazione numerica con:

$$\texttt{SUID bit} \longleftrightarrow 4$$
$$\texttt{SGID bit} \longleftrightarrow 2$$

aggiungendo la cifra all'estrema sinistra come:

```
1 chmod 4750 ...
2 chmod 2750 ...
3 chmod 6750 ...
```

Esempio 18.24: *`chmod` per* SUID *e* SGID *in modalità numerica*

in cui tutti i comandi assegnano permessi di lettura ed esecuzione sia all'owner sia al gruppo e permesso di scrittura all'owner, in aggiunta:

la 1^a aggiunge il *SUID bit*;

la 2^a aggiunge l'*SGID bit*;

la 3^a aggiungerà sia il *SUID bit* sia l'*SGID bit*.

18.8.2 Permessi speciali per directory

Tralasciando il *SUID bit* che, se impostato per una directory, sarà semplicemente ignorato, anche per le directory esistono permessi speciali: **SGID bit** e lo **sticky bit**.

SGID bit

L'**SGID bit** può essere attivato su una directory e ogni file creato nella stessa non saràassegnato al gruppo principale del creatore del file, ma al gruppo definito dalla stessa directory. Tipicamente l'assegnazione dell'*SGID bit* ad una directory è eseguita a directory assegnate

[38] si noti la prima linea

al contenimento di progetti per cui, se un membro può creare file nel contenitore[39], questi saranno assegnati al gruppo definito dalla directory indipendetemente dal gruppo dell'owner.
Ad esempio, di seguito si è creato un gruppo `progetto_top` e assegnati due utenti al gruppo aggiungendo loro il gruppo medesimo come secondario:

```
1 # groupadd project_top
2 # usermod -a -G project_top user1
3 # usermod -a -G project_top user2
```

Esempio 18.25: *Creazione del gruppo* `progetto_top`

Poi si è creata la directory assegnandola al nuovo gruppo:

```
1 # mkdir /opt/project_top
2 # chgrp project_top /opt/project_top
3 # chmod u=rwx,g=srwx /opt/project_top
```

Esempio 18.26: *Creazione della directory assegnandola al gruppo* `project_top`

in cui l'ultima linea è per assegnare:

- i permessi di lettura, scrittura ed esecuzione all'owner e al gruppo;
- aggiungere l'*SGID bit*.

Da questo momento, qualunque utente creerà un file nella directory `/opt/project_top`, il file sarà assegnato al gruppo omonimo definito dal gruppo della directory.

C'è un punto da focalizzare: l'*SGID bit* modifica il comportamento del sistema quando un nuovo file è creato, quindi se il file è ad esempio creato con `touch` o copiato con `cp` all'interno della directory in oggetto.
Se il file è stato creato all'esterno della directory e poi spostato tramite `mv`, il file manterrà il gruppo della creazione.

Sticky bit

Lo **sticky bit**, se impostato per una directory[40], permetterà solo all'*owner* di una risorsa di rinominare o cancellare la stessa dalla directory in oggetto.
In particolare esistono due famose directory del sistema con tali caratteristiche:

```
1 # ls -ld /tmp /var/tmp
2 drwxrwxrwt 28 root root 20480 mar 14 17:46 /tmp/
3 drwxrwxrwt 11 root root  4096 mar 14 17:46 /var/tmp/
```

Esempio 18.27: *Directory di sistema con lo* sticky bit

in cui il carattere `x` per la categoria di utenza *others* è stato sostituito dal carattere `t`.
L'impostazione dello *sticky bit* è utile in contesto di directory *condivisa da tutti*, in cui il permesso `w` concesso a tutti permette a tutti di modificare il contenuto della directory, quindi permetterebbe a tutti di cancellare file di chiunque. Pertanto lo *sticky bit* esegue questo filtraggio necessario a rendere tale directory fruibile: solo il proprietario potrà cancellare

[39] ad esempio appartiene al gruppo del progetto in quanto è tra i suoi gruppi secondari
[40] equivalentemente al *SUID bit* per una directory, è ininfluente per un file

la propria risorsa, oltre ovviamente al proprietario della directory e al *superutente*.

Con il comando `chmod`:

```
1 chmod +t ...
2 chmod -t ...
```

Esempio 18.28: `chmod` *per* SUID *e* SGID *in modalità simbolica*

che rispettivamente:

aggiungerà lo *sticky bit*;

eliminerà lo *sticky bit*.

Analogamente al *SUID bit* e l'SGID bit che si presentano con il carattere `S` al posto di `s` qualora il corrispondente permesso di esecuzione sia mancante, il carattere `T` sostituirà il minuscolo `t`.

Per ultimo, anche per lo *sticky bit* esiste la rappresentazione numerica, e ricordando sia la rappresentazione degli altri permessi speciali sia la rappresentazione ottale, il valore numerico associato sarà immediato:

$$\begin{aligned} \texttt{SUID bit} &\longleftrightarrow 4 \\ \texttt{SGID bit} &\longleftrightarrow 2 \\ \texttt{sticky bit} &\longleftrightarrow 1 \end{aligned}$$

sempre aggiungendo la somma di tali bit come cifra all'estrema sinistra. Ad esempio:

```
1 chmod 1750 ...
2 chmod 2770 ...
3 chmod 3775 ...
4 chmod 4711 ...
5 chmod 5777 ...
6 chmod 6770 ...
7 chmod 7777 ...
```

Esempio 18.29: `chmod` *per* SUID *e* SGID *e* stickybit *in modalità numerica*

in cui per i permessi speciali:

la 1^a aggiunge lo *sticky bit*, simbolicamente sarà `rwxr-x--T`;

la 2^a aggiunge *SGID bit*, simbolicamente sarà `rwxrws--`;

la 3^a aggiunge *SGID bit* e lo *sticky bit*, simbolicamente sarà `rwxrwsr-t`;

la 4^a aggiunge *SUID bit*, simbolicamente sarà `rws-x-x`;

la 5^a aggiunge *SUID bit* e lo *sticky bit*, simbolicamente sarà `rwsrwxrwt`;

la 6^a aggiunge *SUID bit* e *SGID bit*, simbolicamente sarà `rwsrws--`;

la 7^a aggiunge *SUID bit*, *SGID bit* e *sticky bit*, simbolicamente sarà `rwsrwsrwt`.

È utile ricordare che dalla 4^a linea si è impostato il *SUID bit* che non ha alcun effetto sulle directory.

18.8.3 Note aggiuntive per modifica dei permessi speciali

Ovviamente per l'aggiunta dei permessi speciali, il comando `chmod` richiede ovviamente di essere l'utente proprietario per il *SUID bit* e membro del gruppo per il *GUID bit*, a meno che si sia utenti privilegiati.
Per quanto riguarda la directory, `chmod` preserva il *SUID bit* e *SGID bit* a meno che non si espliciti la volontà di modifica, ad esempio con `u+s` o `g-s`. Per la cancellazione di tali bit in modalità numerica sarà necessario aggiungere un ulteriore 0 oppure l'inserimento del simbolo = al posto dei permessi speciali:

```
1 chmod 00755 ...
2 -- oppure --
3 chmod =755 ...
```

Esempio 18.30: *Eliminazione dei permessi speciali per le directory in modalità numerica*

18.9 Maschera dei permessi

A differenza dl comando `chmod`, il quale modifica i permessi su file e directory esistenti, la *maschera dei permessi* determina il default dei permessi sui file e directory che saranno creati.
Il comando per visualizzare tale valore è `umask` eseguito senza argomenti:

```
1 $ umask
2 0002
```

Esempio 18.31: *Visualizzazione della maschera dei permessi correnti*

in cui sarà mostrata la maschera impostata per la sessione[41].
La maschera dei permessi è sostanzialmente la *negazione dei permessi da assegnare* ai file e directory che si creeranno. Ricordando che il sistema dei permessi è in ottale, ogni cifra visualizzata nell'esempio precedente può essere rapportata alla ***Tabella 18.1***:

VALORE$_8$ NEGATO	VALORE$_2$ NEGATO	PERMESSI NEGATI	PERMESSI CONSENTITI
0	000	---	rwx
1	001	--x	rw-
2	010	-w-	r-x
3	011	-wx	r--
4	100	r--	-wx
5	101	r-x	-w-
6	110	rw-	--x
7	111	rwx	---

Tabella 18.1: *Conversione decimale di una singola cifra della maschera al permesso da attribuire*

[41] terminata la sessione, qualora fosse stata modificata, la maschera tornerà ad avere i valori impostati dall'amministratore

La combinazione che sarà assegnata è facile da calcolare dalla stessa tabella, ma il calcolo sarà facilitato se allo stesso comando `umask` si aggiunge il flag `-S`:

```
1 $ umask -S
2 u=rwx,g=rwx,o=rx
```

Esempio 18.32: *Visualizzazione dei permessi consentiti*

È da effettuare una precisazione importante: per i file non saranno consentiti i permessi di esecuzione per le diverse utenze, quindi il valore massimo non sarà 0777 ma 0666[42].

I valori più comunemente impostati per la maschera sono:

002
: per impedire la scrittura agli utenti appartenenti agli *others*;

022
: per impedire la scrittura agli utenti diversi dall'*owner*;

044
: per impedire sia la lettura sia la scrittura agli utenti diversi dall'*owner*;

077
: per impedire tutti i permessi agli utenti diversi dall'*owner*.

Per modificare il valore della mascherà sarà sufficiente aggiungere al comando il nuovo valore numerico:

```
1 $ umask 7
2 -- oppure --
3 $ umask 0007
```

Esempio 18.33: *Modifica maschera dei permessi*

in cui i primi due argomenti sono alternativi[43], e nel caso in oggetto la maschera impedirà ai nuovi file e directory di essere letti, scritti o eseguiti dalla categoria *others*.

Esempi

In accordo con la mascherà 0002 riportata nell'***Esempio 18.31***, eseguendo la creazione di directory e file:

```
1 $ mkdir permessi
2 $ touch permessi.txt
3 $ ls -ld permessi*
4 drwxrwxr-x 2 ms ms 4096 mar 19 17:27 permessi
5 -rw-rw-r-- 1 ms ms    0 mar 19 17:27 permessi.txt
```

Esempio 18.34: *Creazione di directory e file e visualizzazione dei permessi associati*

si avranno i permessi in accordo con quanto riportato nell'***Esempio 18.32***[44].
Per essere numericamente precisi, il sistema determina il permesso da assegnare alla risorsa secondo la seguente espressione:

[42] ovviamente entrambi i valori mostrano uno 0 iniziale, ma potrebbe essere qualunque cifra poiché i permessi speciali potranno essere aggiunti alla risorsa creata solo esplicitamente tramite un `chmod`

[43] si ricorda che per i permessi è attivo il *leading zeros*

[44] si ricorda che i permessi di esecuzione sui file non sono assegnati al momento della creazione

directory: $0777 \;\&\; \sim 0002 = 0775$

file: $0666 \;\&\; \sim 0002 = 0664$

Seguendo l'***Esempio 18.33***, si imposterà la nuova maschera 0006:

```
1 $ umask 0007
2 $ mkdir permessi2
3 $ touch permessi2.txt
4 $ ls -ld permessi*
5 drwxrwxr-x 2 ms ms 4096 mar 19 17:27 permessi
6 drwxrwx--- 2 ms ms 4096 mar 20 15:30 permessi2
7 -rw-rw---- 1 ms ms    0 mar 20 15:30 permessi2.txt
8 -rw-rw-r-- 1 ms ms    0 mar 19 17:27 permessi.txt
```

Esempio 18.35: *Creazione di directory e file con maschera modificata*

in cui i comandi `mkdir` e `touch` creeranno rispettivamente la nuova directory e il nuovo file, le linee successive permettono di confrontare i permessi tra le risorse create con maschera differente.
Anche qui eseguiamo il calcolo numerico per verificare i permessi assegnati:

directory: $0777 \;\&\; \sim 0007 = 0770$

file: $0666 \;\&\; \sim 0007 = 0660$

18.10 Approfondimenti

18.10.1 id utente e gruppo nei processi^P

Ripercorriamo alcuni argomenti trattati:

- nel capitolo 16 si sono presentati alcuni strumenti per effettuare l'*escalation dei privilegi*, in particolare passare dall'utente ordinario al superutente;
- nel capitolo 17 si è focalizzato l'aspetto numerico corrispondente all'utenza introducendo i concetti di *UID* e *GID*;
- in questo capitolo invece l'argomento principale è stato l'*access control* per i processi, e si sono introdotti il *SUID bit* e l'*SGID bit* come eccezioni per l'esecuzione di processi ad utenti che altrimenti non avrebbero la possibilità di eseguire gli stessi.

Quando un utente esegue un processo, a quest'ultimo saranno associati:

RUID, ***real UID***, valorizzato con l'*UID* dell'esecutore;

EUID, ***effective UID***, valorizzato equivalentemente al precedente ad eccezione di esecuzione di binari con il *set-UID bit* attivo[45].

Analogamente saranno associati:

RGID, ***real GID***, valorizzato con il *GID* dell'esecutore;

EGID, ***effective GID***, valorizzato equivalentemente al precedente ad eccezione di esecuzione di binari con il *set-GID bit* attivo[46].

[45] come già argomentato in 18.8.1 a pag. 241, in quest'ultimo caso il valore sarà l'*UID* dell'*owner* della risorsa

[46] come già argomentato in 18.8.1 a pag. 242, in quest'ultimo caso il valore sarà lo *GID* del *group owner* della risorsa

Sintetizzando, quando un binario è invocato, in caso di *set-UID bit* e/o *set-GID bit* attivi, rispettivamente l'*EUID* e/o il *EGID* non saranno quelli dell'utente esecutore ma quelli dell'*owner* e *group owner* della risorsa.
Per quanto riguarda la necessità di avere più variabili con valore talvolta equivalente, principalmente:

- l'*EUID* e l'*EGID* saranno utilizzati per le verifiche di accesso e come *owner* e *group owner* dei file creati dal processo;
- l'*RUID* e lo *RGID* saranno utilizzati per verificare le autorizzazioni per l'invio di *segnali*[47].

Oltre queste variabili, esiste *SUID*[48] utilizzata quando un processo in esecuzione privilegiata deve ridurre *temporaneamente* i propri privilegi. Il processo modifica il proprio *EUID* in uno *non privilegiato*, ma precedentemente avrà copiato l'*EUID* privilegiato nel *SUID*. Successivamente il processo potrà riotterenere i privilegi elevati reimpostando l'*EUID* con il valore di *SUID*.

Per ottenere programmaticamente i valori delle precedenti variabili si utilizzano le funzioni[49] di seguito:

`getuid()` per *UID*;

`geteuid()` per *EUID*;

`getgid()` per *GID*;

`getegid()` per *EGID*;

e per modificarne il contenuto le funzioni[50] di seguito:

`setuid(uid_t uid)` per modificare *EUID* e eventualmente[51] *RUID*;

`setgid(uid_t uid)` per cambiare *EGID* e eventualmente[52] *RGID*;

`setreuid(uid_t ruid, uid_t euid)` per modificare *RUID* e *EUID*;

`setregid(gid_t rgid, gid_t egid)` per modificare *RGID* e *EGID*.

Queste ultime funzioni per la modifica di variabili potrebbero ritornare un errore, ad esempio:

- se alla funzione `setuid(new_uid)` è passato un valore *new_uid* non valido;
- se la funzione `setreuid(new_ruid, new_euid)` è chiamata da un processo non privilegiato e si tenta di cambiare *EUID* ad un valore differente da *RUID* oppure *EUID* oppure *SUID*.

Un esempio delle funzioni presentate è mostrato in 18.10.1 a pag. 248.

Un binario tipico già esaminato[53] è `passwd` utilizzato per il cambio della propria password. Ad esempio:

---

[47] v. capitolo **??** a pag. **??**
[48] ***saved UID***
[49] v. i relativi manuali tramite `man getuid`, ...
[50] v. i relativi manuali tramite `man setuid`, ...
[51] è previsto dalla funzione qualora il processo abbia privilegi da superutente
[52] è previsto dalla funzione qualora il processo abbia privilegi da superutente
[53] v. 18.7 a pag. 241

```
1 $ ls -l `which passwd`
2 -rwsr-xr-x 1 root root 68208 feb  6 13:49 /usr/bin/passwd
```

Esempio 18.36: *Verifica del binario* `passwd`

in cui si nota l'attivazione del *SUID bit*. Successivamente si esegua il binario digitando semplicemente `passwd` e si prema Enter senza andare avanti ulteriormente, quindi il processo di cambio password rimarrà sospeso.
Si apra un nuovo terminale e si digiti il comando `ps`[54] con le seguenti opzioni:

```
1 $ ps -a -o pid,ruid,euid,suid,rgid,egid,sgid,cmd
2     PID  RUID  EUID  SUID  RGID  EGID  SGID CMD
3 ...
4 1097219  1000     0     0  1000  1000  1000 passwd
5 ...
```

Esempio 18.37: *Verifica dei processi attivi con le colonne di interesse*

in cui si è evidenziato il processo di interesse, in particolare:

- il valore di *RUID* è allineato a quello di *UID*;
- i valori di *EUID* e *SUID* sono quelli associati al superutente in accordo con quanto ci si aspetta per l'attivazione del *SUID bit*.

Da sottolinare che queste funzionalità sono progettate non come meccanismo di sicurezza, ma per agevolare il flusso operativo, quindi solitamente per permettere *senza troppe complicazioni* di modificare il proprio *effettivo UID* o *GID* sulle quali i check di sicurezza effettueranno le principali verifiche.

[54] v. **??** a pag. 198 per alcuni esempi e si consulti il manuale per le opzioni utilizzate

18.11 Concetti chiave

maschera dei permessi: è la negazione del permesso da creare, numericamente corrisponde al complementare in base otto dei permessi da assegnare;

permessi: quelli di base sono identificati da una combinazione di autorizzazioni di lettura, scrittura ed esecuzione assegnati al proprietario, al gruppo e altri; oltre quelli base, sono previsti *SUID*, *SGID* e *sticky bit*;

SGID: se su file, il processo che eseguirà tale risorsa avrà assegnato il gruppo della risorsa; se su directory, le risorse create saranno assegnate al gruppo della directory;

set-GID bit: v. *SGID*

set-UID bit: v. *SUID*

sticky bit: su una directory, permette solo al proprietario (e superutente) di rinominare o cancellare la risorsa;

SUID: se su file, il processo che eseguirà tale risorsa avrà assegnato l'user dello script.

18.12 Comandi introdotti nel capitolo

`chmod` ⇒ visualizza/modifica i bit di accesso alla risorsa

`chown` ⇒ cambia l'owner e il gruppo della risorsa

`chgrp` ⇒ cambia il gruppo della risorsa

`stat` ⇒ mostra le metainformazioni relative alla una risorsa (file o filesystem)

`umask` ⇒ visualizza/modifica la maschera dei permessi

Capitolo 19

Installazione software

QUESTO capitolo illustrerà le due principali modalità di installazione di software nel sistema le cui fonti saranno *da sorgente* o da *pacchetto*.
L'installazione da sorgente prevede che ci siano il codice di un qualche software preparato tendenzialmente da qualcuno e che si vuole creare un eseguibile sul proprio sistema. L'alternativa è prendere un pacchetto che tendenzialmente conterrà, oltre l'eseguibile, le informazioni necessarie per l'integrazione con la propria distribuzione.

19.1 Da sorgente[1]

Le sezioni successive illustreranno in maniera sintetica i passaggi per l'installazione o pacchettizzazione da sorgente servendosi di tool di base. Qualora qualcuno sia mancante, essendo sicuramente[2] disponibili nella propria distribuzione, per completare il passaggio sarà necessaria l'installazione seguendo le indicazioni fornite in 19.2.1 a pag. 257.

19.1.1 Scaricamento sorgenti

Ovviamente tale fase dipenderà molto dalla locazione del sorgente, quindi non è possibile fornire una linea guida genereale. Fortunatamente molti progetti inseriscono una pagina di istruzioni*tipicamente un* `README` per scaricare il sorgente[3], e sarà utile anche per la successiva fase di compilazione.
Di seguito le più frequenti modalità:

```
1 $ wget source_url
2 -- oppure --
3 $ git clone source_project
```

Esempio 19.1: *Scaricare il sorgente*

in cui:

[1] questa sezione percorrerà in maniera sintetica le fasi di installazione da sorgente, ma saranno mostrati alcuni passaggi non illustrati in dettaglio per i quali si rimanda alla relativa nota

[2] avrei potuto scrivere *certamente* ma, seppur estremamente infrequente, non posso escludere che si sia optato per lavorare con una distribuzione che non fornisca tali strumenti di base

[3] se forniscono già il pacchetto per la propria distribuzione, si può optare per l'installazione diretta di tale pacchetto seguendo le indicazioni fornite in 19.2.3 a pag 264

`wget` *source_url*
: scaricherà il sorgente dall'url indicata;

`git clone` *project_url*
: tramite lo strumento `git`[4] sarà clonato il progetto dall'url indicata.

Nel primo caso solitamente si scaricherà un `tar` e si dovrà procedere, dopo il controllo del *checksum*[5], alla sua apertura:

```
1 $ tar -xvf file.tar.gz
2 -- oppure --
3 tar -jvf file.tar.bz2
```

Esempio 19.2: *Apertura dell'archivio*

in cui le alternative proposte sono quelle più frequenti per la decompressione dell'archivio.

19.1.2 Installazione o pacchettizzazione della sorgente

Avendo a disposizione in locale il sorgente sarà possibile procedere con l'installazione diretta oppure pacchettizzare il sorgente per poter poi procedere con l'installazione del pacchetto prodotto.
Lo svantaggio dell'installazione senza pacchettizzazione è non poter gestire quanto installato dal sistema di *package management* della distribuzione.
In ogni caso di seguito i due percorsi simili, entrambi dovrebbero iniziare con la lettura del file `README` che potrà fornire indicazioni utili, compreso le eventuali dipendenze[6] necessarie (anche se spesso saranno solo indicate).

Installazione diretta

Solitamente i passaggi da eseguire per la compilazione e installazione da sorgente sono i seguenti:

```
1 ./configure
2 make
3 make check
4 sudo make install
```

Esempio 19.3: *Comandi per la compilazione del sorgente*

in cui:

`./configure`
: è sicuramente il passaggio più delicato[7] poiché, tra le altre operazioni, verifica che il sistema abbia tutto il necessario per costruire l'eseguibile;

`make`
: esegue la compilazione[8] dei sorgenti;

[4] approfondito in **??** a pag. **??**
[5] anche *digest*, già incontrato nell'***Esempio 10.35***, sarà approfondito in **??**
[6] v. **??** a pag. **??**
[7] approfondito in 19.4.1 a pag. 267
[8] spesso le applicazioni che richiedono la compilazione sono scritte in C e C++. Ciò rende necessario rispettivamente i compilatori `gcc` e `g++` i quali nei sistemi *Debian-like* saranno entrambi installati con il metapacchetto `build-essential`

`make check`
: passaggio per verificare che la compilazione sia avvenuta correttamente. Tale passaggio è solitamente opzionale poiché, qualora nella fase precedente sia avvenuto un errore, la stessa terminerà bruscamente avvertendoci;

`make install`
: copia il binario prodotto dalla fase di compilazione nella directory determinata nella fase di configurazione.

Installazione tramite pacchettizzazione

Solitamente i passaggi da eseguire sono molto simili:

```
1 ./configure
2 make
3 make check
4 sudo checkinstall
```

Esempio 19.4: *Comandi per la compilazione del sorgente e l'installazione tramite pacchettizzazione*

in cui l'unica variazione rispetto la sezione precedente è nell'ultimo comando `checkinstall` che di default procederà ad installare il binario e, attraverso il monitoraggio dell'attività, procederà all'installazione del pacchetto e installazione dello stesso. Rispetto la sola installazione senza pacchettizzazione, il binario potrà eventualmente poi essere rimosso tramite la procedura definita dal sistema di package management[9]. Il tool `checkinstall` prevede diverse opzioni, alcune[10] sono:

```
1 checkinstall --install=no
2 -- oppure --
3 checkinstall -D
4 -- oppure --
5 checkinstall -R
6 -- oppure --
7 checkinstall -S
```

Esempio 19.5: *Alternative maggiori del comando* `checkinstall`

in cui:

`checkinstall --install=no`
: non effettua l'installazione del pacchetto;

`checkinstall -D`
: crea un pacchetto *Debian*;

`checkinstall -R`
: crea un pacchetto *RPM*;

`checkinstall -S`
: crea un pacchetto *Slackware*.

[9] v. 19.2.3 a pag. 265

[10] per la lista completa consultare il `man checkinstall`

19.2 Installazione da pacchetto

Sui sistemi GNU/Linux il software è solitamente installato tramite *pacchetti*, distribuito tramite *repository* e gestito nei server attraverso i *package manager*. Tipicamente ogni sistema avrà migliaia di pacchetti installati, molti dei quali saranno dipendenze di altri pacchetti.

19.2.1 Concetti essenziali

Di seguito saranno brevemente discussi i termini appena citati.

Pacchetti

Moltissimo software applicativo per i sistemi GNU/Linux e più in generale per i sistemi Unix sono distribuiti tramite pacchetti i quali sono solitamente archivi contenenti un mix di:

- software binario già precompilato;
- script di installazione;
- file di configurazione;
- requisiti di dipendenza;
- documentazione;
- altro.

Questi archivi eterogenei sono solitamente specifici per una particolare distribuzione e *pacchettizzati* per una particolare distribuzione in accordo a specifici vincoli del formato (ad esempio `.deb` per i Debian e Ubuntu, `.rpm` per RHEL/Fedora/CentOS, ...).
La pacchettizzazione semplifica estremamente la gestione agli utenti finali che installeranno, aggiorneranno o rimuoveranno il software, ma è uno strato di complessità che si aggiunge allo sviluppo del software, per cui si introduce la figura del *manutentore* il quale ha il compito di pacchettizzare (e ripacchettizzare) con eventuali upgrade di sicurezza e bug fix.

Repository

Per l'utente installatore è sufficiente ottenere il pacchetto in qualunque metodo[11], ma tipicamente si utilizzano i *repository*, locazioni in cui sono solitamente resi disponibili via web migliaia di pacchetti per varie distribuzioni ciascuna con le più recenti release.
Le varie distribuzioni manutengono i propri repository denominati *ufficiali* contenente tanti pacchetti, solitamente suddivisi in macro categorie. Esistono anche *repository non ufficiali* che solitamente sono manutenute da enti o community con lo scopo di fornire software che talvolta è in sovrapposizione con quanto contenuto nei repository ufficiali. I motivi per cui esistono tali repository non ufficiali possono essere diversi, spesso una combinazione delle motivazioni[12] seguenti:

[11] ad esempio scaricarlo direttamente dalle pagine di un progetto

[12] ovviamente l'elenco non si esaurisce con quanto riportato, ma quanto fornito sarà sicuramente integrato dalle intuizioni del lettore

- i repository non ufficiali mantengono software che per questioni di licenza non sono contenuti nei repository della distribuzione;
- i repository non ufficiali aggiornano frequentemente il repository per offrire pacchetti con versioni software maggiormente aggiornate;
- i repository ufficiali aggiornano meno frequentemente il pacchetto perché il manutentore, in relazione agli standard propri (o della community) preferisce un software maggiormente maturo.

Dipendenze

Per la maggior parte del software è richiesto che già ulteriore software sia in esecuzione o comunque disponibile. Nei pacchetti sono inclusi metadati per indicare software richiesto affinché quello contenuto nel pacchetto possa essere installato e eseguito: tali informazioni traducono il concetto di *dipendenze*[13]. La variazione del numero delle dipendenze richieste dai pacchetti è elevata, da poche a centinaia, ma solitamente le stesse dipendenze ne richiedono di proprie. Quindi quando si installeranno, aggiorneranno o rimuoveranno pacchetti, dovranno essere valutate e soddisfatte le dipendenze in maniera ricorsiva, quindi dovranno essere installati, aggiornati o anche rimossi pacchetti.

Package manager

Come già accennato in 16.8 a pag. 204, un'area di notevole differenziazione tra le varie distribuzioni è il sistema di *package management*. Sintetizzando quanto scritto nelle precedenti sezioni, essi sono stati introdotti per ridurre la complessità inerenti la gestione del processo di scaricamento, installazione, aggiornamento e rimozione di pacchetti e loro dipendenze.
Effettivamente ciò ha migliorato notevolmente la gestione del software dei sistemi e, tra le molte considerazioni, gli amministratori di sistema sicuramente valutano tale aspetto per le varie necessità o, più semplicemente, codeterminano le loro preferenze.
I benefici di un **package management system** si possono sintetizzare in:

facilità nella ricerca del pacchetto desiderato. La maggioranza delle distribuzioni mantiene i propri repository, per cui l'utilizzo di un gestore di pacchetti garantisce l'installazione dei pacchetti che sono stati precedentemente controllati, ritiene affidabili e sufficientemente stabili[14] per il compito che svolgeranno.

gestione automatica delle dipendenze. Anche se le dipendenze possono essere distribuite all'interno del pacchetto per garantire la loro presenza, per diverse ragioni (tra le quali un archivio più snello), i pacchetti solitamente definiscono le dipendenze attraverso metainformazioni, quindi sarà il gestore stesso a cercare pacchetti che soddisfino tali dipendenze.

rispetto delle convenzioni della distribuzione. Ciascuna distribuzione adotta proprie convenzioni[15] che evolveranno con il rilascio delle release, e attraverso la pacchettizzazione le varie distribuzioni potranno imporre il rispetto dei propri standard.

[13] argomento esaminato in **??**

[14] specialmente il criterio di *stabilità* è soggettivo poiché, anche se quantitativamente misurabile, la soglia per la quale si ritiene il pacchetto stabile e/o affidabile è fortemente legato agli standard della distribuzione e/o dell'approccio della community

[15] ad esempio, v. 12.3 a pag. 133

aggiornamento rapido. In particolare le patch di sicurezza, ma più in generale gli aggiornamenti software, i package manager forniscono comandi semplici per l'*upgrade* dei pacchetti installati che puntano ai repository indicati dal sistema. Ovviamente, la condizione necessaria è che gli stessi repository siano mantenuti aggiornati, specialmente per le patch di sicurezza[16].

19.2.2 Breve comparazione tra package manager

[17] Ci sono tanti gestori di pacchetti, ciascuno con le proprie caratteristiche. Di seguito una breve lista dei più comuni, indicando le distribuzioni che le forniscono, il formato dei pacchetti e una breve descrizione.

APT

distribuzioni: *Debian*, *Ubuntu*, *Kali Linux*, ...

comandi: `apt`, `apt-get`, `apt-cache`

package management di base : *Debian package* via `dpkg`

formato del pacchetto: `.deb`

L'***A**dvanced **P**ackage **T**ool*, comunemente conosciuto come **APT**, è un front-end di alto livello il quale utilizza il package manager a basso livello `dpkg` per la gestione dei pacchetti nella distribuzione. *APT* è formato da diversi tool a riga di comando, i maggiori sono `apt-get`, `apt-cache` e l'omonimo `apt`[18].

DNF

distribuzioni: *RHEL*, *Centos 8*, *Fedora*, ...

comandi: `dnf`

package management di base : *RPM Package Manager* via `rpm`

formato del pacchetto: `.rpm`

Il ***Dan**dified YUM*, comunemente conosciuto come **DNF**, è il successore di *YUM*[19]. Rispetto al predecessore ha diversi miglioramenti che includono migliori performance, maggior rapidità nella risoluzione delle dipendenze e una migliore documentazione. Si segnala che molte distribuzioni linkano il comando `yum` verso `dnf` poiché la retrocompatibilità è elevata[20].

---

[16] argomento trattato in **??** a pag. **??**

[17] probabilmente questa è la sezione che prevede maggior obsolescenza, sarà aggiornata quanto possibile, ma difficilmente con frequenza sufficiente

[18] quest'ultimo sostanzialmente fornisce all'utente le funzionalità dei due tool precedenti

[19] v. 19.2.2

[20] in altri termini, la maggior parte degli argomenti digitati via *cli* èil medesimo

YUM

distribuzioni: *RHEL*, *Centos 7*, *Fedora 21*, ...

comandi: `yum`

package management di base : *RPM Package Manager* via `rpm`

formato del pacchetto: `.rpm`

Lo ***Y**ellowdog **U**pdater, **N**odified*, comunemente conosciuto come **YUM**, è un package menager[21] per le distribuzioni *RHEL-based*, il quale fornisce un'interfaccia ad alto livello tramite `yum` per il package manager *RPM Package Manger* invocato tramite `rpm`.

Zypper

distribuzioni: *openSUSE*

comandi: `zypper`

package management di base : *ZYpp*[22] via `zypp`

formato del pacchetto: `.rpm`

Il package manager *Zypper* è lo strumento utilizzato nella distribuzione *openSUSE* ma, a differenza dei precedenti *DNF* e *YUM* che gestiscono pacchetti `.rpm`, la sua interfaccia a basso livello non è *RPM* ma *ZYpp*.
Dal punto di vista dell'utilizzatore, si segnala che lo stesso comando fornisce la possiblità di aggiungere direttamente i repository.

Pacman

distribuzioni: *Arch-based*

comandi: `pacman`

formato del pacchetto: `.tar.xz`[23] e altri formati `tar` compressi

Il package manager **Pacman** gestisce protopacchetti compressi in *tarball*, quindi tendenzialmente di dimensioni inferiori rispetto agli altri formati di pacchettizzazione. Si intuisce quindi che, a differenza degli altri gestori, *Pacman* non si limita alla gestione dei pacchetti ma incorpora un meccanismo di costruzione del pacchetto denominato **ABS**[24].

Portage

distribuzioni: *Gentoo*

comandi: `emerge`

formato del pacchetto: *ebuild* script o `.tbz2`[25]

Il package manager **Portage** non utilizza pacchetti con binari precompilati, ma solitamente è formato da script di shell denominati *ebuild*.

[21] sempre meno utilizzato a favore di **DFN** in 19.2.2
[22] anche *libzypp*
[23] compresso tramite il tool `xz`
[24] ***A**rch **B**uild **S**ystem*
[25] compresso tramite il tool `bzip`

Slackware Package Management

distribuzioni: *Slackware*

comandi: `slackpkg`, `pkgtool`, `installpkg`, `upgradepkg` e `removepkg`

formato del pacchetto: `.tgz`[26] e `.txz`[27]

Nella distribuzione *Slackware* convivono diversi package manager:

`slackpkg` permette l'installazione dei pacchetti disponibili via web;

`pkgtool` permette la gestione dei pacchetti attraverso un'interfaccia testuale;

`installpkg`, `upgradepkg` e `removepkg` sono i tool specializzati rispettivamente per installare, aggiornare e rimuovere pacchetti.

19.2.3 Esempi di package management

Come argomentato nella precedente sezione, genericamente un sistema di package management non dovrà solo prevedere l'installazione dei pacchetti della distribuzione, ma tra i compiti maggiori ci saranno l'aggiornamento di quelli installati e l'eventuale rimozione degli stessi. Inoltre, nelle installazioni di server spesso si ha necessità di aggiungere repository non ufficiali per poter accedere a pacchetti maggiormente aggiornati.
Di seguito si illustrano le attività più frequenti e si fornirà il relativo comando con i maggiori package manager.

Aggiornamento lista dei pacchetti I sistemi solitamente mantengono un database locale contenente la lista dei pacchetti disponibili dai repository. Una operazione preliminare da compiere è aggiornare tale lista cosicché si potrà procedere correttamente all'installazione o aggiornamento dei pacchetti. Come eccezione, il package manager *dnf* procede autonomamente ad effettuare tale aggiornamento prima di procedere ad installazioni/upgrade, ma si potrà verificare comunque eventuali aggiornamenti disponibili.

Debian / Ubuntu

```
1 apt-get update
2 -- oppure --
3 apt update
```

Esempio 19.6: *Comandi per l'aggiornamento della lista dei pacchetti* `.deb` *disponibili*

RHEL / Fedora

```
1 dnf check-update
```

Esempio 19.7: *Comandi per l'aggiornamento della lista dei pacchetti* `.rpm` *disponibili*

[26] compresso tramite il tool `gzip`
[27] compresso tramite il tool `xz`

Aggiornamento dei pacchetti installati Senza un package manager, per verificare che il software installato nella distribuzione sia aggiornato, si dovrebbe tener traccia di centinaia di fonti sparse sul web e, soprattutto, essere attenti ad alert di sicurezza[28], in sintesi un'attività enormemente complessa.
Fortunatamente, escludendo eventuali errori, con un package manager l'aggiornamento dei pacchetti installati o dell'intero sistema si riduce a pochi comandi.

Debian / Ubuntu

```
1 apt-get ugrade
2 -- oppure --
3 apt upgrade
```

Esempio 19.8: *Comandi per l'aggiornamento del sistema* `.deb`

RHEL / Fedora

```
1 dnf upgrade
```

Esempio 19.9: *Comandi per l'aggiornamento del sistema* `.rpm`

Ricerca di un pacchetto e dipendenze Per trovare pacchetti di interesse sarà utile cercare dal nome o descrizione dello stesso, ma anche entrare nei dettagli informativi relativi allo stesso prima dell'eventuale installazione.

Debian / Ubuntu

```
1 apt-cache search search_string
2 -- oppure --
3 apt search search_string
```

Esempio 19.10: *Ricerca di pacchetti* `.deb`

in cui *search_string* sarà sostituita dalla stringa da cercare. Dalla lista che uscirà sarà possibile entrare nei dettagli di un generico pacchetto digitando:

```
1 apt-cache showpkg pacchetto
2 -- oppure --
3 apt show pacchetto
```

Esempio 19.11: *Informazioni su pacchetto* `.deb`

il cui output conterrà metadati utili quali il numero di versione, una descrizione sommaria, la lista delle dipendenze, ...
Ad esempio, nella sessione seguente tramite `apt-cache`:

```
1 $ apt-cache search editor
2 ...
3 tmispell-voikko - interfaccia iso-Ispell per il correttore
      ortografico Voikko
4 vim - VI Migliorato - editor vi arricchito
```

[28] su questi ultimi è comunque bene non abbassare la guardia

```
5 vim-common - Vi IMproved - Common files
6 ...
7 $ apt-cache showpkg vim
8 Package: vim
9 Versions:
10 2:8.1.2269-1ubuntu5.22 (/var/lib/apt/lists/it.archive.ubuntu.
       com_ubuntu_dists_focal-updates_main_binary-amd64_Packages) (/var/
       lib/apt/lists/security.ubuntu.com_ubuntu_dists_focal-
       security_main_binary-amd64_Packages) (/var/lib/dpkg/status)
11  Description Language:
12                  File: /var/lib/apt/lists/it.archive.ubuntu.
       com_ubuntu_dists_focal_main_binary-amd64_Packages
13                   MD5: 59e8b8f7757db8b53566d5d119872de8
14  Description Language:
15                  File: /var/lib/apt/lists/it.archive.ubuntu.
       com_ubuntu_dists_focal_main_binary-i386_Packages
16                   MD5: 59e8b8f7757db8b53566d5d119872de8
17  Description Language: it
18 ...
```

Esempio 19.12: *Ricerca di un editor e check di pacchetto tramite* `apt-cache`

oppure analogamente tramite `apt`:

```
1 $ apt search editor
2 ...
3 vim/focal-updates,focal-security,now 2:8.1.2269-1ubuntu5.22 amd64 [
       installato]
4   VI Migliorato - editor vi arricchito
5 ...
6 $ apt show vim
7 Package: vim
8 Version: 2:8.1.2269-1ubuntu5.22
9 Priority: optional
10 Section: editors
11 Origin: Ubuntu
12 Maintainer: Ubuntu Developers <ubuntu-devel-discuss
       lists.ubuntu.com>Original-Maintainer:  Debian Vim Maintainers
       <pkg-vim-maintainerslists.alioth.debian.org>
13 Bugs: https://bugs.launchpad.net/ubuntu/+filebug
14 Installed-Size: 3.121 kB
15 Provides: editor
16 Depends: vim-common (= 2:8.1.2269-1ubuntu5.22), vim-runtime (=
       2:8.1.2269-1ubuntu5.22), libacl1 (>= 2.2.23), libc6 (>= 2.29),
       libcanberra0 (>= 0.2), libgpm2 (>= 1.20.7), libpython3.8 (>=
       3.8.2), libselinux1 (>= 1.32), libtinfo6 (>= 6)
17 Suggests: ctags, vim-doc, vim-scripts
18 Homepage: https://www.vim.org/
19 Task: server, cloud-image, lubuntu-desktop
20 Download-Size: 1.243 kB
21 APT-Manual-Installed: yes
22 APT-Sources: http://it.archive.ubuntu.com/ubuntu focal-updates/main
       amd64 Packages
23 Description: VI Migliorato - editor vi arricchito
24  Vim è una versione pressoché compatibile con l'editor UNIX Vi.
25  .
26  Sono state aggiunte molte nuove caratteristiche: annullamento
       operazioni
27  multilivello, evidenziazione della sintassi, cronologia della riga
       di
28  comando, aiuto in linea, completamento dei nomi dei file, operazioni
        a
```

```
29  blocchi, possibilità di nascondere il codice, gestione Unicode, ecc.
30  .
31  Questo pacchetto contiene una versione di vim compilata con un
       insieme
32 ...
```

Esempio 19.13: *Ricerca di un editor e check di pacchetto tramite* `apt`

In ultimo, per ottenere la lista delle dipendenze:

```
1 $ apt-cache depends vim
2 vim
3   Dipende: vim-common (= 2:8.1.2269-1ubuntu5.22)
4   Dipende: vim-runtime (= 2:8.1.2269-1ubuntu5.22)
5   Dipende: libacl1 (>= 2.2.23)
6   Dipende: libc6 (>= 2.29)
7 ...
8 -- oppure --
9 $ apt depends vim
10 ...
```

Esempio 19.14: *Lista delle dipendenze tramite* `apt-cache` *e* `apt`

in cui gli output sono equivalenti.

RHEL / Fedora

analogamente all'***Esempio 19.10*** sarà[29]:

```
1 dnf search search_string
```

Esempio 19.15: *Ricerca tramite* `dnf` *di pacchetti* `.rpm`

in cui *search_string* sarà sostituita dalla stringa da cercare. Dalla lista che uscirà, analogamente all'***Esempio 19.11***, sarà possibile entrare nei dettagli di un generico pacchetto digitando[30]:

```
1 dnf info pacchetto
```

Esempio 19.16: *Informazioni tramite* `dnf` *su pacchetto* `.rpm`

in cui per brevità non sarà riportata la sessione di esempio.
Non esiste propriamente un analogo dell'***Esempio 19.14*** il quale mostra le dipendenze, ma si può ovviare con:

```
1 dnf repoquery --requires pacchetto
```

Esempio 19.17: *Lista delle dipendenze tramite* `dnf`

Installazione di pacchetti

L'installazione dei pacchetti può avvenire con due modalità differenti:

da repository: in continuità con le sezioni precedenti, si procederà all'installazione dai repository;

da locale: se il pacchetto o i pacchetti sono già disponibili localmente.

[29] la linea estese è `dnf [options] search [--all] <keywords>`
[30] la linea estese è `dnf [options] info [<package-file-spec>...]`

Installazione da repository Trovati i pacchetti di interesse, si potrà procedere all'installazione degli stessi e delle dipendenze richieste con estrema semplicità.

Debian / Ubuntu

```
1 apt-get install pkg1 pkg2 ...
2 -- oppure --
3 apt install pkg1 pkg2 ...
```

Esempio 19.18: *Installazione pacchetti* `.deb` *da repository*

RHEL / Fedora

```
1 dnf install pkg1 pkg2 ...
```

Esempio 19.19: *Installazione pacchetti* `.rpm` *da repository*

Installazione da locale Talvolta alcuni progetti, soprattutto se sviluppati da vendor, sono già pacchettizzati per varie distribuzioni, ma non sono collocati nei repository ufficiali. Tralasciando le implicazioni di sicurezza, se si opta per questa[31], tipologia di installazione, i passaggi sono:

scaricamento: download[32] del pacchetto;

installazione: al termine dello scaricamento, si porcederà all'installazione secondo il proprio package manager.

Debian / Ubuntu
si utilizzerà direttamente lo strumento *dpkg* per l'installazione dei pacchetti:

```
1 dpkg -i pacchetto
```

Esempio 19.20: *Installazione pacchetti* `.deb` *da locale*

In caso il pacchetto avrà delle dipendenze fallite, queste ultime si risolveranno:

manualmente: tornando alla fase di scaricamento di altri pacchetti e poi provandoli ad installare tutti con `dpkg`;

automaticamente: ricorrendo al tool `gdebi`[33].

RHEL / Fedora
analogamente all'***Esempio 19.20***, sarà si può optare per l'installazione manuale

```
1 rpm -i pacchetto
```

Esempio 19.21: *Installazione pacchetti* `.rpm` *da locale a basso livello*

ma, nel caso di dipendenze fallite, si dovranno risolvere manualmente.
Alternativamente, in maniera più moderna e più rapida, si potrà utilizzare direttamente il package manager *dnf* poiché, oltre a saper risolvere le dipendenze, potrà procedere all'installazione diretta del file:

[31] è in alternativa a quanto illustrato in 19.1 a pag 253
[32] ad esempio con `wget` oppure `curl` o, in un sistema desktop, tramite browser
[33] v. 19.4.2 a pag.267

```
1 dnf install pacchetto.rpm
```

Esempio 19.22: *Installazione pacchetti* `.rpm` *da locale*

Rimozione di pacchetti

Se un software non è più necessario, talvolta si opterà per la rimozione.

Debian / Ubuntu

Per la semplice rimozione del pacchetto:

```
1 apt-get remove pacchetto
2 -- oppure --
3 apt remove pacchetto
```

Esempio 19.23: *Rimozione di un generico pacchetto* `.deb`

o, se si desidera anche la cancellazione[34] dei file di configurazione:

```
1 apt-get purge pacchetto
2 -- oppure --
3 apt purge pacchetto
```

Esempio 19.24: *Rimozione di un generico pacchetto* `.deb` *con rimozione dei relativi file di configurazione*

RHEL / Fedora

analogamente all'***Esempio 19.23***, si effettuerà la disinstallazione tramite:

```
1 dnf erase pacchetto
```

Esempio 19.25: *Rimozione di un generico pacchetto* `.rpm`

in cui non è prevista anche la rimozione dei file di configurazione.

Elenco pacchetti installati

Esistono diversi metodi per ottenere l'elenco dei pacchetti già installati nel sistema, probabilmente di seguito i più immediati.

Debian / Ubuntu

```
1 apt-cache pkgnames
2 -- oppure --
3 apt list --installed
```

Esempio 19.26: *Elenco dei pacchetti* `.deb` *installati*

RHEL / Fedora

```
1 dnf list installed
```

Esempio 19.27: *Elenco dei pacchetti* `.rpm` *installati*

[34] per approfondimenti v. 19.4.3 a pag. 269

19.3 Domare tutti i package manager

Come già accennato esistono tantissimi package manager per gestire l'installazione del software nelle diverse distribuzioni. Se l'utente installatore rimane in un'unica famiglia di distribuzione, tendenzialmente non avrà problemi nel ricordare gli argomenti, ma con l'aumentare dell'eterogeneità, le difficoltà cresceranno, con il caso estremo di non sapere quale sia il comando per invocare il package manager della distribuzione alla quale ci si approccia.
Per molti potrebbe essere un problema minore, però è nato il progetto **UPT**[35] il quale ha lo scopo di interfacciarsi direttamente con il package manager della distribuzione. Ovviamente non sarà un frontend universale, ma con il tempo sta ampliando i package manager supportati e la profondità nell'utilizzo degli stessi. Un breve elenco dei supportati:

- apk
- apt
- brew
- cards
- choco
- dnf
- emerge
- eopkg
- flatpak
- guix
- nix-env
- opkg
- pacman
- pkg
- pkg(termux)
- pkgman
- prt-get
- scoop
- slackpkg
- snap
- urpm
- winget
- xbps
- yum

[35] ***U**niversal **P**ackage-management **T**ool*, [sig]

- zypper

e, come riportato nella pagina di progetto di cui un estratto in ***Tabella 19.1***, per ciascuno esiste la corrispondente azione supportate per `install`, `uninstall`, `upgrade`, `search`, `info`, `update`, ...

19.4 Approfondimenti

19.4.1 ./configure

Preliminarmente andranno letti i file `README` e `INSTALL` i quali conterranno utili informazioni per l'installazione, come i requisiti software necessari. Questi ultimi sono spesso costituiti da librerie con nome non sempre identico ai pacheddi della distribuzione, per cui andranno spesso ricercati opportunamente[36].
Solitamente lo step successivo è procedere con lo script `configure` il quale ha come scopi principali:

- verifica che nel sistema siano già presenti i requisiti necessari per la costruzione dell'eseguibile;
- permette di cambiare il path in cui di default sarà installato il binario;
- permette di abilitare o disabilitare nel compilato varie opzioni o moduli.

Per verificare le opzioni disponibili dal proprio file di configurazione, sarà sufficiente digitare:

```
1 $ ./configure --help
```

Esempio 19.28: *Opzioni del config*

Una delle modifiche più frequenti è nel path, che di default è solitamente `/usr/local`. Per impostarlo[37] si modificherà come di seguito:

```
1 $ ./configure --PREFIX=/opt
```

Esempio 19.29: *Opzioni del config*

avendo scelto come path di destinazione `/opt`.

19.4.2 Tool Gdebi

Il software *gdebi* è uno strumento per l'installazione dei pacchetti *deb* il cui scopo è l'automatizzazione del processo di risoluzione delle dipendenze dello stesso pacchetto ricorrendo al package manager *APT*.
Senza tale strumento, è comunque possibile procedere alla risoluzione automatica delle dipendenze con questi passaggi:

```
1 # dpkg -i pacchetto.deb
2 ...
3 # apt-get -f install
4 ...
```

[36] ad esempio, nei *Debian like* tramite `apt-cache search` o semplicemente `apt search`

[37] ovviamente non sarà equivalente per ogni configurazione, ma questa procedura è corretta nella stragrande maggioranza dei casi. Per verificarla, si verifichi con l'*help*

TOOL	INSTALL	UNINSTALL	UPGRADE	SEARCH	INFO	UPDATE INDEX	UPGRADE ALL	LIST INSTALLED
upt	`upt install` *pkg*	`upt remove/ uninstall` *pkg*	`upt upgrade` *pkg*	`upt search` *pkg*	`upt info/ show` *pkg*	`upt update`	`upt upgrade`	`upt list`
apk	`apk add` *pkg*	`apk del` *pkg*	`apk upgrade` *pkg*	`apk search` *pkg*	`apk info` *pkg*	`apk update`	`apk upgrade`	`apk list -I/ --installed`
apt	`apt install` *pkg*	`apt remove` *pkg*	`apt install --only-upgrade` *pkg*	`apt search` *pkg*	`apt show` *pkg*	`apt update`	`apt upgrade`	`apt list -i/ --installed`
brew	`brew install` *pkg*	`brew uninstall` *pkg*	`brew upgrade` *pkg*	`brew search` *pkg*	`brew info` *pkg*	`brew update`	`brew upgrade`	`brew list`
dnf	`dnf install` *pkg*	`dnf remove` *pkg*	`dnf upgrade` *pkg*	`dnf search` *pkg*	`dnf info` *pkg*	`dnf check-update`	`dnf update`	`dnf list --installed`
flatpak	`flatpak install` *pkg*	`flatpak uninstall` *pkg*	`flatpak update` *pkg*	`flatpak search` *pkg*	`flatpak info` *pkg*	-	`flatpak update`	`flatpak list`
urpm	`urpmi` *pkg*	`urpme` *pkg*	`urpmi` *pkg*	`urpmq -y/ --fuzzy` *pkg*	`urpmq -i` *pkg*	`urpmi.update -a`	`urpmi --auto-update`	`rpm -q/ --query --all`
yum	`yum install` *pkg*	`yum remove` *pkg*	`yum upgrade` *pkg*	`yum search` *pkg*	`yum info` *pkg*	`yum check-update`	`yum update`	`yum list --installed`
zypper	`zypper install` *pkg*	`zypper remove` *pkg*	`zypper update` *pkg*	`zypper search` *pkg*	`zypper info` *pkg*	`zypper refresh`	`zypper update`	`zypper search -i/ --installed-only`

Tabella 19.1: *Equivalenza delle azioni in alcuni dei package manager supportati*

Esempio 19.30: *Installazione pacchetto locale con successiva risoluzione delle dipendenze*

La prima linea installerà il pacchetto, ma in caso di errore per dipendenze non soddisfatte, sarà necessaria la seconda utilizzando il flag `-f`[38] il quale cerca di correggere le dipendenze non soddisfatte.
Tale approccio non è propriamente corretto poiché l'ultima linea potrebbe fallire[39], in ogni caso il passaggio *non è pulito* in quanto il sistema avrà (nel migliore dei casi) uno stato di dipendenze non coerente.
Quest'ultima considerazione suggerisce di percorrere un *preferibile* approccio che conduce al tool *gdebi*, il quale, prima di alterare lo stato di pacchettizzazione del sistema, segnale eventuali errori e/o l'impossibilità di installazione.
Installando il pacchetto `gdebi`[40], si avrà la possibilità di utilizzare tale strumento. Quindi per installare *pacchetti deb* locali, ci si porrà nella stessa directory del pacchetto, quindi si digiterà:

```
1 # gdebi pacchetto.deb
```

Esempio 19.31: *Installazione pacchetto locale con risoluzione delle dipendenze*

il quale installera via *APT* le dipendenze del pacchetto, successivamente lo stesso sarà installato e registrato nel database del packet management. L'approccio consigliato è comunque verificare il contenuto del pacchetto tramite il tool dpkg con:

```
1 # dpkg --info pacchetto.deb
```

Esempio 19.32: *Installazione pacchetto locale con risoluzione delle dipendenze via* APT

e successivamente *simulare* le chiamate eseguite dallo stesso ad *APT*, verificando quindi le operazioni di quest'ultimo:

```
1 # gdebi --apt-line pacchetto.deb
```

Esempio 19.33: *Simulazione dell'installazione pacchetto locale con risoluzione delle dipendenze via* APT

Per gli altri utilizzi si rimanda al manuale[41].

19.4.3 Stato dei pacchetti

In 19.2.3 si è riscontrato che nei sistemi Debian-like un pacchetto rimosso non è necessariamente totalmente rimosso. In particolare si può consultare lo stato dei pacchetti e talvolta si incontrerà la sigla `rc` la quale indica che il pacchetto è stato rimosso dal sistema, ma non lo sono i relativi file di configurazione[42].

[38] equivalentemente `--fix-broken`

[39] potrebbe non riuscire a soddisfare le dipendenze, quindi gergalmente trovandosi con un pacchetto installato *rotto*

[40] ad esempio tramite `apt-get install gdebi`

[41] `man gdebi`

[42] questo comportamento è utile in caso si voglia reinstallare il package e trovare una configurazione probabilmente più desiderabile rispetto quella di default

L'elenco dei pacchetti del sistema compreso il relativo stato è contenuto nel file `/var/lib/dpkg/status`. Un elenco più compatto può essere ottenuto digitando `dpkg -l` in cui saranno mostrati tutti[43] i package del sistema ordinati alfabeticamente. Ciascuna riga mostrerà oltre gli ovvi nome del package e la relativa descrizione, ne mostrerà altri, ma di particolare interesse è la sequenza di tre flag[44] nelle prime tre colonne (l'ultima tendenzialmente non riporterà alcun flag), rispettivamente:

azione desiderata: indica l'azione da compiere al pacchetto, sarà uno tra i seguenti flag:

`u` per *Unknown*;
`i` per *Install*;
`h` per *Hold*;
`r` per *Remove*;
`p` per *Purge*;

stato del pacchetto: indica lo stato attuale del pacchetto, sarà uno tra i seguenti flag:

`n` per *Not-installed*;
`c` per *Config-files*;
`H` per *Half-installed*;
`U` per *Unpack*;
`F` per *Half-configured*;
`W` per *Triggers-awaiting*;
`t` per *Triggers-pending*;
`i` per *Installed*;

flag di errore: se presente fornisce un'indicazione di errore, in particolare i flag saranno uno tra i seguenti:

`' '`[45] se non ci sono errori;
`R` per *Reinst-required*.

In accordo con quanto indicato all'inizio della sezione, dall'elenco precedente è immediata l'interpretazione della sequenza di flag `rc`. Operativamente può essere ottenuta la lista dei package relativi come segue:

```
1 $ dpkg -l | grep ^rc
2 -- oppure --
3 $ apt list '~c'
```

Esempio 19.34: *Elencare i pacchetti `rc`*

e i pacchetti elencati saranno cancellati rispettivamente con i seguenti comandi:

```
1 # dpkg -l | grep ^rc | cut -d' ' -f 3 | xargs dpkg --purge
2 -- oppure --
3 # apt purge '~c'
```

Esempio 19.35: *Eliminare i pacchetti `rc`*

[43] nel caso specifico tutti, ma ovviamente si potrà aggiungere un ulteriore argomento che filterà il nome dei package; ad esempio, il comando `dpkg -l *libc6*` riporterà solo package nel cui nome sia contenuta la stringa `libc6`

[44] per approfondire ulteriormente è utile consultare il manuale `man dpkg-query`

[45] carattere spazio

19.5 Concetti chiave

dipendenze: requisiti software per procedere all'installazione del software in oggetto;

pacchetti: archivi contenenti solitamente software, metadati, script e documentazione che caratterizzano la distribuzione;

package manager: software per la gestione dei pacchetti e responsabile delle operazioni da effettuare per mantenere aggiornato il database del sistema di pacchettizzazione della distribuzione;

repository: alberi di directory accessibili via web contenenti pacchetti forniti dalla distribuzione o da community.

19.6 Comandi introdotti nel capitolo

`checkinstall` ⇒ genera e (di default) installa il pacchetto dal sorgente

`gdebi` ⇒ tool per installare file `.deb`

`git` ⇒ è un *revision control sytem* molto utilizzato per manutenere progetti, utilizzato in questo conteso solo per clonarlo in locale

`wget` ⇒ utility per scaricare dal web

Capitolo 20

Networking

In questo capitolo saranno esaminati i rudimenti del networking in GNU/Linux e gli strumenti da utilizzare per la presenza del sistema nel network.

20.1 Background

20.1.1 Protocolli

Oggigiorno i sistemi sono connessi in svariati modi ad internet, i principali sono:

- via *LAN*[1];
- via *router* di casa;
- via WiFi o telefonino.

Indipendentemente dalle modalità, l'utilizzo della rete (ad un livello sufficientemente astratto) funzionerà in maniera pressoché identica.
Senza addentrarci nei dettagli, Internet si basa sul funzionamento di una *famiglia di protocolli* il cui nome è **TCP/IP**, nome che deriva dai due più famosi protocolli contenuti nello stack.
Un *protocollo* è costituito da un insieme di convenzioni tecniche utilizzate[2] affinché la comunicazione possa avvenire e, soprattutto, possa avvenire efficacemente.
Gli aspetti che possono essere definiti nei dettagli tecnici del protocollo sono i più svariati ma, a livello intuitivo, avendo per oggetto la comunicazione, si indirizzeranno verso:

- i dispositivi, per definire dettagli sui segnali elettrici o ottici o radio da utilizzare;
- il software, per definire dettagli sul *linguaggio*[3] da utilizzare.

I protocolli possono essere distinti i tre categorie:

medium access protocol , per definire i dettagli funzionali per il controllo dell'accesso al mezzo fisico e (quindi al livello delle schede di rete e cavi) e trasmissione di dati sullo stesso;

[1] ***l****ocal* ***a****rea* ***n****etwork*
[2] tendenzialmente dai dispositivi o dai software
[3] essenzialmente lessico e sintassi

communication protocol , per regolare la comunicazione tra dispositivi geograficamente distanti[4], i più famosi sono i già citati *IP*[5] e *TCP*[6];

application protocol , per regolare la comunicazione al livello più astratto, ad esempio affinché:

- un client di posta elettronica possa leggere le email contenute in un server di posta (ad esempio la lista oppure il contenuto di una precisa email);
- un client di posta elettronica possa inviare email;
- un browser web possa ottenere le informazioni da renderizzare (testo, foto, ..., e certamente come l'autore della pagina avrà voluto disporre il tutto)
- una comunicazione (web, posta, . . .) possa avvenire in maniera cifrata.

Da notare che la categorizzazione di protocolli proposta è una distinzione a *macro livelli* in cui, affinché la comunicazione possa avvenire tra mittente e destinatario, le informazioni siano scambiate tra i diversi *macro livelli* con trasformazione della stessa per essere opportunamente gestita.
Tipicamente uno scambio di un messaggio tra mittente e destinatario prevederà in prima approssimazione questi step:

1. l'informazione trattata nel *protocolli applicativo* del mittente sarà trasformata in *pacchetti* per essere gestita dai *protocolli di comunicazione*;
2. i pacchetti dai *protocolli di comunicazione* saranno ulteriormente trasformati in *frame* per essere gestiti dai *protocolli di accesso al mezzo* per poter uscire dal dominio del mittente;
3. questo è tendenzialmente uno step ricorsivo poiché l'informazione sarà solitamente trasformata molte volte da frame a pacchetto e da pacchetto a frame; tale numero è correlata con il numero di passaggi e mezzi trasmissivi necessari per raggiungere il destinatario. Tale step sarà concluso quando si giungerà al dominio del destinatario;
4. l'informazione arriverà alla postazione del destinatario giungendo in frame ai *protocolli di accesso al mezzo* che la trasmuteranno in pacchetti per essere gestita dai *protocolli di comunicazione* del destinatario;
5. i pacchetti trattati dai *protocolli di comunicazione* saranno trasformati nell'informazione originale gestibile dal *protocollo applicativo* del destinatario per poter così essere utilizzata dall'utenza finale.

L'impostazione *multi livello* è in realtà maggiormente segmentata[7], ma in generale è interessante osservare che ogni *layer* comunicherà esclusivamente con quello immediatamente sopra o sotto, gestendo una piccola parte della complessità inerente la comunicazione, delegando quindi il resto della complessità ad altri layer.
Ad esempio, riprendendo l'elenco precedente, per una pagina web sommariamente accadrà quanto di seguito:

[4] intuitivamente, se ad esempio una persona esegue un download di un file, le informazioni organizzate in byte di dati che giacciono da qualche parte su un server, dovranno essere veicolate e copiate nello stesso ordine al sistema del soggetto che le ha richieste

[5] ***I**nternet **P**rotocol*, per approfondimenti [81b]

[6] ***T**ransmission **C**ontrol **P**rotocol*, per approfondimenti [Edd22]

[7] al momento quest'aspetto sarà trascurato

1. sarà gestita dal *protocollo applicativo* HTTP, il quale si occuperà solo del confezionamento della stessa per essere inviata, ma non della logica per convogliarla al destinatario;
2. la logica per consegnarla al destinatario sarà nei pacchetti gestiti dal *protocollo di comunicazione* il quale trascurerà i dettagli per la trasmissione sul mezzo fisico;
3. i dettagli trasmissivi saranno nel frame il quale è l'oggetto dei *protocollo di accesso al mezzo*, frame che sarà successivamente trasmutato, a seconda del dispositivo, in segnali elettromagnetici o ottici sul mezzo trasmissivo (quindi cavi o wireless o altro);
4. fino all'arrivo al destinatario, per tutti i nodi i frame saranno ritrasformati in pacchetti affinché il nodo stesso possa leggere i dettagli per l'invio al destinatario, quindi inoltrarlo[8] verso la via corretta, per cui dovrà nuovamente trasformarlo in frame in accordo al mezzo trasmissivo del successivo segmento;
5. i frame giunti al destinatario saranno trasformati in pacchetti e questa volta, non saranno inoltrati a nessuno ma passeranno allo step successivo;
6. poiché il destinatario è stato raggiunto, i pacchetti andranno a ricomporre l'informazione originale che formerà la pagina web per l'utente.

Quanto fin qui brevemente presentato è spesso presentato in letteratura come *modello ISO/OSI* il quale è costituito da 7 livelli, ma per lo più è rimasto teorico. La concretizzazione tecnica è con il più snello *Internet protocol suite*, più comunemente stack TCP/IP. Quest'ultimo si è evoluto nel tempo con tantissimi protocolli per gestire le più diverse necessità ed affrontare i tanti dettagli tecnici, spesso dettagli che sono cambiati in accordo con l'evoluzione tecnologica. Per citare i più famosi, sono:

protocolli applicativi: è il layer più elevato dell'*Internet protocol suite*:

HTTP: ***H**yper**T**ext **T**ransfer **P**rotocol*, già citato per il trasferimento di contenuti via web (pagine, foto, video, file, ...);

SMTP: ***S**imple **M**ail **T**ransfer **P**rotocol*, per l'invio di email a server di posta;

POP3/IMAP4: ***P**ost **O**ffice **P**rotocol* e ***I**nternet **M**essage **A**ccess **P**rotocol*[9], per la gestione (lettura, cancellazione, ...) delle email su server di posta;

DNS: ***D**omain **N**ame **S**ystem*, per la risoluzione di nomi di dominio ad ip e viceversa, ad esempio

$$example.com \rightarrow 192.0.2.1$$

oppure

$$192.0.2.1 \rightarrow example.com$$

FTP: ***F**ile **T**ransfer **P**rotocol*, per il trasferimento di file (tendenzialmente per uso più tecnico e spesso ad uso non pubblico);

[8] potrebbe accadere che taluni nodi vogliano ricostruire l'informazione dai pacchetti, proprio per logiche interne al dispositivo. In questo caso, i pacchetti non saranno subito trasformati in frame, ma si ricostruirà tendenzialmente parte dell'informazione originale e eventuali decisioni sulla sorte degli stessi

[9] anche noto come ***I**nteractive **M**ail **A**ccess **P**rotocol*

SSH: ***S**ecure **SH**ell*, per accesso interattivo su server remoti (uso più tecnico ad uso privato);

SIP: ***S**ession **I**nitiation **P**rotocol*, per instaurare, modificare o terminare sessioni multimediali tra partecipanti, includendo chiamate telefoniche via Internet (*Voip*[10]), videoconferenze, ...;

TLS/SSL: ***T**ransport **L**ayer **S**ecurity* e ***S**ecure **S**ockets **L**ayer*, protocolli crittografici progettati per fornire sicurezza[11] delle comunicazioni. Alcuni protocolli applicativi si appoggiano a questo per usufruire del livello di sicurezza, come *HTTPS*, *SMTPS*, *POP3S*, *IMAPS* e *FTPS*.

protocolli di comunicazione: senza entrare nei dettagli, è costituito da due layer:

Transport layer: i protocolli forniscono un canale logico alla comunicazione end-to-end per le applicazioni[12]. Le questioni fondamentali affrontate da questo layer sono:

segmentazione: trasforma il flusso di byte derivante dall'applicazione in blocchi denominati *segmenti*, i quali permetteranno di ricostruiranno l'informazione originale[13] quando saranno consegnati all'host di destinazione;

multiplexing: fornire più canali contemporanei per l'utilizzo dell'applicazione[14]. Tale obiettivo sarà ottenuto tramite l'assegnazione di un medesimo *port number* a tutti i segmenti relativi ad uno stesso flusso;

checksum: ad ogni segmento è inserito un checksum[15] cosicché il destinatario saprà se quello arrivato sarà corrotto (e deciderà di conseguenza).

I più famosi sono:

TCP: il già citato *Transmission Control Protocol*, si occupa di *controllo della trasmissione*, quindi rendere affidabile la comunicazione in rete tra mittente e destinatario. Le caratteristiche principali sono:

connection-oriented, inizializzando e chiudendo la trasmissione tra mittente e destinatario;

ordered data, l'host destinatario riassembla i segmenti secondo la sequenza numerica assegnata al segmento dal mittente;

reliable service, in cui l'affidabilità sarà garantita dalla ritrasmissione dei pacchetti persi o corrotti;

flow control, riduzione (e riaumento) nel *delivery rate* qualora si verifichino questioni di riempimento del buffer di ricezione client oppure problemi di congestione (tipicamente perdita di pacchetti).

UDP: *User Datagram Protocol*, molto più leggero del precedente:

connectionless, non è prevista l'esplicita inizializzazione o chiusura della connessione;

[10] ***V**oice **o**ver **IP***

[11] il *TLS* sostituisce il deprecato *SSL* citato per consuetudine

[12] i compiti principali cui i protocolli gestiscono sono *comunicazione orientata alla connessione*, *affidabilità*, *controllo del flusso*, *multiplexing*; nessuno di tali servizi è obbligatorio, per cui ciascuna applicazione sarà costruito sul protocollo più adatto

[13] o quasi, vedi la descrizione dei protocolli

[14] un server accetterà più connessioni contemporanee al servizio che offrirà, anche per lo stesso client

[15] alcuni dettagli relativi sono discussi nei protocolli

unordered data, non gestisce l'ordinamento dei blocchi in quanto non assegnati dal mittente;

unreliable service, considerato di minor affidabilità poiché, assieme al mancato ordinamento, i segmenti eventualmente persi non saranno ritrasmessi;

checksum, può essere usato[16], se errato l'integrità del segmento è compromessa per cui lo stesso sarà scartato.

Ovviamente le caratteristiche mancanti sono in negativo rispetto al TCP, ma è da considerare lo scopo per cui è stato progettato: l'*estrema rapidità*. Difatti non c'è latenza per riordino o ritrasmissione, per cui è efficiente per le applicazioni *time-sensitive*[17] quali le trasmissioni audio/video in realtime (VoIP o streaming).

Internet layer: anche denominato *network layer*, essenzialmente è utilizzato per trasportare i pacchetti dall'host di origine all'host di destinazione attraverso tutti i nodi del cammino. Questo è essenzialmente il problema del *routing*, cioè determinare il corretto instradamento dal mittente al destinatario, decisioni di routing che saranno prese per ogni nodo attraversato.
I principali protocolli sono:

IP: è il protocollo alla base del funzionamento della rete Internet. È stato progettato per interconnettere reti[18] tecnologicamente eterogenee.

ICMP: ***I**nternet **C**ontrol **M**essage **P**rotocol* non è generalmente dalle applicazioni utenti, ma è utilizzato dai device per inviare messaggi di errore o informazioni relativi a successo o fallimento, pertanto utilizzato da tecnici per eseguire attività di troubleshooting.

protocolli di accesso al mezzo: è costituito da un gruppo di metodi e protocolli di comunicazione confinati al collegamento a cui è fisicamente connesso un host, per cui la caratterizzazione generale è maggiormente sfumata poiché legata ai metodi del segmento di rete. Per esempio, taluni protocolli del segmento sono affidabili occupandosi della *rivelazione degli errori* di trasmissione, alcuni ricercano l'*ottimizzazione* dell'invio dei frame tramite politiche di sincronizzazione tra trasmettitore e ricevitore.
Dal punto divista dell'host, è il livello a cui si trova l'indirizzo fisico. Questo livello ha principalmente due funzioni:

incapsulamento dei dati: è l'incapsulamento per la trasmissione nel mezzo denominato *framing*;

accesso al mezzo: gestisce l'accesso al mezzo, comunicando direttamente con il livello fisico.

[16] la valorizzazione è opzionale nell'IPv4, se non usato sarà costituito da tutte cifre 0
[17] tipicamente flusso informativo in ritardo non ha utilità, eventualmente negativa
[18] non a caso è *inter-net*working *protocol*

I protocolli più famosi sono **ATM**[19], **ARP**[20], **MAC**[21], **Eternet**[22], **Frame Relay**, **PPP**[23], **X.25**, **WLAN**[24].

20.1.2 Indirizzamento e routing

Durante l'attività di navigazione sul web[25], si eseguiranno richieste a server remoti e le repliche torneranno all'host mittente. Affinché le repliche[26] possano tornare al richiedente, ciascun host connesso utilizzerà un *indirizzo logico* unico denominato *indirizzo IP*. Si argomenterà nel seguito, ma nell'*IPv4*[27], ma questa prevede l'adozione l'indirizzo è costituito da una sequenza di quattro *ottetti*[28], ad esempio 192.168.0.1.
Ovviamente l'indirizzo IP non può essere scelto arbitrariamente, ma dovrà essere assegnato seguendo dei vincoli[29]. Tali vincoli derivano dalla necessità insite nello scambio di dati tra differenti reti affinché i nodi possano adottare le politiche di instradamento definite **routing**.
Gli host GNU/Linux hanno solitamente più di un indirizzo IP:

loopback address: è disponibile su ogni host, corrisponde all'indirizzo 127.0.0.1, e si utilizza per definire se stessi. In altri termini, non è un indirizzo accessibile dall'esterno, ma si usa molto spesso per sviluppare e testare servizi di rete sulla stessa macchina di sviluppo;

network address: è l'indirizzo di rete del dispositivo (Ethernet o WiFi) il quale è connesso alla rete. Su questa interfaccia possono essere presenti ulteriori indirizzi come *alias* per test o configurazioni specifiche.

La questione posta precedentemente, è come un generico host su Internet possa comunicare con un server che non è sulla stessa rete, quindi come possono comunicare se non possono scambiare direttamente un flusso di dati?
La questione è già stata accennata: la soluzione è il *routing*. In prima approssimazione:

1. l'host conosce l'indirizzo del server consultando il servizio *DNS*;
2. l'indirizzo IP del server tipicamente non è sulla stessa **LAN**[31] dell'host;
3. l'host dovrà consultare l'esterno, cioè un nodo che saprà instradare il flusso verso la destinazione.

[19] ***A**synchronous **T**ransfer **M**ode*
[20] ***A**ddress **R**esolution **P**rotocol*
[21] ***m**edium **a**ccess **c**ontrol*, anche ***m**edia **a**ccess **c**ontrol*
[22] formalmente **IEEE 802.3**
[23] ***P**oint-to-**P**oint **P**rotocol*
[24] ***w**ireless **l**ocal **a**rea **n**etwork*, formalmente **IEEE 802.11**
[25] ma si può in generalizzare un host connesso via Internet
[26] l'argomento in realtà è valido anche per le richieste affinché possano raggiungere il server remoto
[27] è lo schema maggiormente utilizzato, in 20.1.4 a pag. 282 sarà argomentato l'*IPv6*
[28] nelle telecomunicazioni il termine corretto da utilizzare è ottetto, però nella maggior parte dei casi si utilizza byte e, anche se nella stragrande maggioranza dei casi sono quantitativamente equivalenti, non sono termini in assoluto interscambiabili
[29] per l'utente ordinario, nelle aziende gli *amministratori di rete* assegneranno l'indirizzamento, nelle connessioni casalinghe/da ufficio gli *ISP*[30] si preoccuperanno di tali assegnazioni. Si può notare che spesso nelle aziende agli utenti sarà assegnato sempre il medesimo ip, nelle connessioni da ufficio/casalinghe gli ISP assegneranno lo stesso indirizzo solo per un (relativamente breve) periodo di tempo
[31] ***l**ocal **a**rea **n**etwork*

Per l'ultimo passaggio si utilizzerà la *default route* (o eventuali *route* specifiche) che definiranno il path del nodo esterno che sarà *più vicino* al server.
Nel caso di *default route*, l'host ricevente è denominato **default gateway**.
Questo step sarà ripetuto per più passaggi finché, se tutto è configurato correttamente, un nodo potrà comunicare con il server di destinazione direttamente.
Analogamente accardà per le repliche dal server all'host, esse dovranno seguire la stessa logica, anche se i cammini di andata e di ritorno non saranno necessariamente identici[32] tra loro.
Da notare che l'host mittente e il server potranno comunicare senza conoscere i nodi intermedi per i quali i pacchetti sono passati. Più in generale, i nodi non dovranno conoscere null'altro che i vicini, tipicamente gli host connessi alla propria rete e il (o i) gateway a cui inviare i flussi di datagrammi qualora non conoscano direttamente il destinatario.

In sintesi, l'host dovrà conoscere il proprio IP, gli indirizzi degli host con cui potrà dialogare direttamente, e il default gateway a cui invierà tutte le altre comunicazioni.
L'indirizzamento con cui un host potrà dialogare direttamente è definito dalla *rete locale* individuata dal proprio IP e dalla **subnet mask**.
Ipotizziamo che il proprio host abbia l'indirizzo $192.168.1.31$ e la propria LAN abbia indirizzamento $192.168.1.0/24$.
Il numero 24 è la subnet mask la quale indica che i primi 24 bit dell'indirizzamento di rete sono equivalenti ai 24 bit del proprio indirizzo. In maniera complementrare, possiamo affermare che nel caso suddetto la rete dell'host è formata da indirizzi in cui varieranno solo gli ultimi 8[33] bit, quindi proprio l'ultimo ottetto. Dal punto di vista della comunicazione, se l'indirizzo dell'altro host sarà compreso tra $192.168.1.0$ a $192.168.1.255$[34] tenterà la comunicazione diretta, se varierà almeno un qualunque altro bit tra i primi 24, l'host sarà considerato esterno, quindi si procederà con l'invio della comunicazione al gateway relativo[35].
Sintetizzando, i tre parametri discussi (indirizzo IP, subnet mask e gateway) saranno i parametri da configurare, in aggiunta alla definizione dell'indirizzo del *DNS server* se l'host vorrà comunicare conl'esterno tramite i nomi di dominio. Solitamente tali parametri saranno configurati dal proprio *DHCP client* che consulterà il *DHCP*[36] *server* che, se esisterà, fornirà tali parametri ai richiedenti.
Dal punto di vista operativo, per connettere un computer ad una LAN spesso sarà sufficiente collegarlo ad una rete (tramite cavo Eternet o WiFi), e l'host riceverà i parametri necessari dal DHCP server.

Saturazione dell'indirizzamento

Con l'aumento del numero di dispositivi connessi, lo spazio di indirizzamento della versione 4 del protocollo IP si sta esaurendo[37].

[32] anche se solitamente non accade, in linea di principio le politiche di routing possono variare anche tra pacchetti di una stessa comunicazione, quindi instradarli in maniera differentemente e, escludendo problemi, tutti raggiugerebbero la destinazione, quindi caratterizzando il protocollo IP di un'elevatissima flessibilità

[33] 32 bit complessivi meno i 24 dell'indirizzamento di rete

[34] sarà argomentato tra breve, ma sono da escludere gli estremi

[35] tendenzialmente il default

[36] ***D****ynamic* ***H****ost* ***C****onfiguration* ***P****rotocol*

[37] questo problema è indicato dai media anglosassoni con *IPcalypse* o *ARPAgeddon*

Escludendo la riprogettazione del protocollo[38], il quale prevede l'adozione di un nuovo protocollo, sono state introdotte diverse contromisure per combattere tale saturazione. Una già esaminata è avvenuta con l'introduzione della *subnet mask esplicita*[39]. Prima di tale introduzione, l'indirizzo definiva implicitamente la subnet mask[40] la quale era fissa di 1 o 2 o 3 ottetti[41]. Con l'introduzione della subnet mask esplicita, la lunghezza della rete è divenuta di lunghezza variabile al bit, in particolare essa è definita esplicitamente tramite *notazione decimale*[42] o notazione **CIDR**[43].

Una seconda importante evoluzione del protocollo è stata l'introduzione degli *indirizzi ip privati*[44].
Fino a quel momento ogni host aveva un indirizzo IP che per definizione era pubblico, cioè visibile a tutti gli host connessi. Successivamente chiunque ha potuto utilizzare indirizzi IP privati per la propria rete locale, e spesso un unico indirizzo ip pubblico da assegnare ai router.
Adottando quest'ultimo schema, per comunicare con l'esterno la rete locale dovrà inviare i pacchetti al router, il quale li instraderà come se fossero propri, modificando l'indirizzo mittente con il proprio indirizzo pubblico. Quando tornerà la risposta, sarà effettuata l'operazione inversa, sostituendo al proprio indirizzo quello dell'host originariamente mittente[45].
Nella ***Tabella 20.1*** sono riassunte le classi di indirizzamento ad uso privato.

LUNGHEZZA	INDIRIZZO		BLOCCO CIDR	N° INDIRIZZI
	INIZIALE	FINALE	PIÙ GRANDE	DISPONIBILI
24 bit	10.0.0.0	10.255.255.255	10.0.0.0/8	16.777.216
20 bit	172.16.0.0	172.31.255.255	172.16.0.0/12	1.048.576
16 bit	192.168.0.0	192.168.255.255	192.168.0.0/16	65.536

Tabella 20.1: *Classi di indirizzi privati su IPv4*

Indirizzi speciali

Per completezza si citano alcuni indirizzi IPv4 destinati ad usi diversi:

`127.0.0.0/8`
: utilizzati per il loopback verso l'host locale;

`169.254.0.0/16`
: utilizzati per la comunicazione tra 2 host quando senza utilizzare uno specifico indirizzo IP di rete[46];

[38] avvenuta con l'*IPv6*, argomento affrontata in 20.1.4 a pag. 282
[39] v. [RL93] e [Ful+93]
[40] tramite il **classful addressing**, in cui i primi 3 bit dell'indirizzo IP determina la classe, e di conseguenza, la maschera di sottorete
[41] quindi di lunghezza fissa rispettivamente di 8, 16 o 24 bit
[42] ad esempio `255.255.255.0`
[43] ***C**lassless **I**nter-**D**omain **R**outing*, come `/24`
[44] v. [Mos+96]
[45] questa procedura va sotto il nome di **NAT** (***n**etwork **a**ddress **t**ranslation*)
[46] ad esempio è utilizzato per l'acquisizione di un indirizzo da un server DHCP

`224.0.0.0/4`
: utilizzati per il multicast;

`255.255.255.255/32`
: usati per il brodcast.

20.1.3 Nomi e DNS

Si intuisce navigando dal web, ma generalizzando è più facile che si utilizzino nome di dominio rispetto gli indirizzi IP corrispondenti. Ad esempio, sarà difficile ricordare 159.65.31.14, meno il suo dominio corrispondente `one2zero.it`.
Come già accennato nella sezione 20.1.1 a pag. 273, il servizio che trasforma i nomi in indirizzi e viceversa è il *DNS*, in cui fornirà rispettivamente la *risoluzione diretta* e la *risoluzione inversa*.
Esso è un database distribuito a livello globale di nomi, indirizzi e altre informazioni mantenute su record dai *DNS server* e consultabili da *DNS client*[47].
Solitamente ogni ISP gestirà un proprio server DNS[48], spesso molte grandi aziende gestiranno il proprio, e anche il lettore potrebbe gestirne uno su sistema GNU/Linux[49].
Il protocollo DNS[50] è gestito a livello gerarchico[51] come segue:

TLD[52], anche *dominio di primo livello*, è la parte all'estrema destra dall'ultimo punto della stringa del dominio, quindi `.com`, `.org`, `.eu`, `.it`, ...;

dominio di secondo livello: formato da parole indicativamente arbitrarie ed accostate al primo livello. Ad esempio, `one2zero.it`

dominio di livello *n*: figlio del dominio di livello *n-1*, ogni livello è una suddivisione del precedente; assegnato il dominio di secondo livello, l'assegnatario è autorizzato a registrare e gestire i nomi di dominio relativi ai successivi livelli come preferisce. Ad esempio un terzo livello sarà `www.one2zero.it`.

Nel processo di ricerca dei nomi di dominio, i *TLDs* giocano un ruolo molto importante: per tutte le richieste non in cache, quando si richiede il dominio di secondo livello, il *DNS resolver*[53] richiederà[54] il dominio al server TLD relativo. Rimanendo nell'esempio *one2zero.it*, la richiesta generica sarà inoltrata al DNS server di riferimento, il *DNS resolver* di quest'ultimo, qualora non abbia già l'informazione in cache, inoltrerà la richiesta al relativo TLD (quindi `.it`), il quale consulterà il proprio registro e replicherà con l'indirizzo dei *DNS autoritativo* per il dominio richiesto, quest'ultimo replicherà con la risoluzione.
Qualora si richiedesse un dominio di terzo livello o con profondità maggiori, il processo sarebbe analogo, con l'aggiunta di eventuali passaggi qualora il DNS autoritativo delegherà ad altri per profondità maggiori.

---

[47] un qualunque software che utilizzi il *protocollo DNS* per effettuare *query DNS*
[48] in realtà, almeno per ridondanza, più di uno
[49] la gestione sarà approfondita nel capitolo **??** a pag. **??**
[50] per approfondimenti [87a] e [87b]
[51] in realtà manca il livello radice dei *DNS root zone*, ma al momento è preferibile non appesantire la trattazione
[53] componente software sui *DNS server* che si propone di risolvere l'indirizzo
[54] in realtà questa procedura è un po' più complessa poiché coinvolge i *root nameserver* (i quali gestiscono la *root zone*) che sono gli unici server a possedere l'elenco dei server autoritativi di tutti i domini di primo livello, e lo forniscono in risposta a ciascuna richiesta

È utile notare che il disaccoppiamento nomi/indirizzi porta ad una flessibilità interessante in cui possono essere distinti due casi estremi:

1. nomi diversi possono essere associati ad un unico indirizzo ip;
2. più indirizzi ip possono essere associati ad un unico nome.

Ad esempio, il primo caso è associato a più servizi erogati dallo stesso server, o nel caso del *virtual hosting*, una tecnica per cui un web server con un unico IP può ospitare più *siti web* ai quali correttamente reindirizzerà ricevendo la richiesta tramite nome del sito[55].
Il secondo caso si può utilizzare per bilanciare il carico dei client che richiedono quel dominio poiché la richiesta sarà distribuita sui vari IP associati ai relativi server.
Meno frequente, ma è chiaramente possibile una combinazione delle due alternative dell'elenco: più nomi possono essere associati ad un pool di indirizzi.

20.1.4 IPv6

Finora si è trattato dell'indirizzo logico riferendosi allo schema tradizionale che è l'*IPv4*[56].
La maggiore specifica riguarda l'indirizzamento usato, sequenza di di 32 bit per l'indirizzo mittente, 32 bit perl'indirizzo destinatario.
Nel tempo, aumentando il numero di dispositivi connessi, questo spazio è iniziato ad essere relativamente stretto, per cui si è affrontato il problema iniziando ad adottare l'*IPv6*[57] il quale, tra le altre modifiche, prevede un indirizzamento di 128 bit per il mittente, 128 bit per il destinatario.
Per intuire la differenza dello spazio di indirizzamento tra le due versioni, si usa confrontare le due grandezze:

$$2^{32} << 2^{128}$$

in cui a sinistra è indicata l'espressività numerica permessa da 32 bit, a destra con 128; esplicitando le grandezze, la disuguaglianza equivale a:

$$4.294.967.296 << 340.282.366.920.938.463.463.374.607.431.768.211.456$$

ma il confronto così mostrato ha poco senso in quanto, per entrambi i protocolli, non tutto lo spazio è utilizzabile per l'indirizzamento[58].
Al momento non solo GNU/Linux, ma la quasi totalità dei sistemi gestisce entrambi i protocolli

Notazione

Per la rappresentazione di un indirizzo IPv6 si utilizza la rappresentazione esadecimale suddividendo la sequenza in 8 gruppi e utilizzando il carattere `:` come separatore ogni due byte (4 cifre esadecimali).
Ad esempio, un indirizzo potrebbe essere:

```
2001:0db8:0000:0001a:0000:0000:2b3c:4d5e
```

[55] più precisamente, utilizzerà il nome ricevuto nella richiesta HTTP specificato nell'header
[56] ***I**nternet **P**rotocol **v**ersion **4***
[57] ***I**nternet **P**rotocol **v**ersion **6***
[58] in ogni caso si ammetterà l'efficacia del confronto numerico

. Se i gruppi sono composto da cifre 0 a sinistra, queste possono essere omesse, quindi la precedente può essere scritta più sinteticamente in:

```
2001:0db8:0:1a:0:0:2b3c:4d5e
```

. Inoltre una sequenza di uno o più gruppi contigui di sequenze `0` può essere sostituita con la sequenza `::`. La precedente può essere scritta:

```
2001:0db8:0:1a:0::2b3c:4d5e
```

o più sinteticamente

```
2001:0db8:0:1a:::2b3c:4d5e.
```

Chiaramente la regola precedente può essere utilizzata una solo volta per non creare ambiguità[59].

Prefisso

Nell'IPv6 i primi 64 bit indentificano il *routing prefix* e sono utilizzati per definire a quale rete un indirizzo appartiene, quindi sono utilizzati dai router per la logica di instradamento. Nell'IPv6 le subnet mask sono contenute in tale prefisso, ad esempio gli ISP potrebbero[60] utilizzare una subnet `/48` per i propri clienti, lasciando i bit complementari `/16`[61] per il subnetting interno.
La principale[62] schematizzazione è proposta di seguito:

global unicast address, in sigla **GUA**[63], è l'analogo degli indirizzi pubblici per l'IPv4, quindi sono indirizzi globalmente instradabili, per cui sono assegati ai dispositivi su Internet per ottenere la raggiungibilità;

unique local address, in sigla **ULA**, avrà prefisso `fd00::/8`[64], indirizzi simili ai privati per l'IPv4 poiché non saranno instradati a livello globale, ma scambiati solo nel dominio della propria rete privata. La differenza sostanziale con gli indirizzi IPv4 privati è la loro utilizzazione *libera*, senza registrazione centralizzata all'interno dell'organizzazione;

link local address, sono indirizzi validi solo localmente che non saranno mai instradati, quindi sono utilizzati per scambiare pacchetti IPv6 nella rete locale. Quando si abilita un'interfaccia su IPv6, essa creerà automaticamente un indirizzo di tipo *link local* nel range di indirizzamento `FE80::/10`[65], sostanzialmente equivalenti al blocco `169.254.0.0/16` per il *link local* su IPv4.

---

[59] ad esempio, l'ip precedente non potrebbe essere scritto come `2001:0db8::1a::2b3c:4d5e` poiché non sarebbe correttamente interpretabile

[60] ma ovviamente c'è arbitrarietà

[61] `/64` del prefisso meno i `/48` dell'esempio sarà una `/16`

[62] di seguito sono trascurati gli indirizzi *multicast* e *anycast*

[63] talvolta *global unique address*

[64] formalmente `fc00::/7` (in binario `1111 110`), quindi il range che che ne consegue sarà suddiviso in due blocchi equivalenti con l'ottavo bit `0` oppure `1`, rispettivamente `fc00::/8` (in binario `1111 1100`) e `fd00::/8` (in binario `1111 1101`). Il primo blocco è stato proposto per l'allocazione ma l'IETF non ha accolto l'utilizzo

[65] quindi i primi 10 bit saranno `1111 1110 10`

Retrocompatbilità con IPv4

Nell'IPv6 è stata creata una classe per favorire la transizione da IPv4 a IPv6 per host che supportano solo IPv4 e, per tale indirizzamento, gli ultimi 32 bit possono essere equivalentemente espressi nella notazione *dotted decimal*. Tale sintassi IPv6 è retrocompatibile con quella IPv4 ed è denominata *IPv4-mapped address*.
Il funzionamento è molto semplice: i primi 80 bit sono posti a `0`, i successivi 16 bit sono posti a `1` (in esedacimale `ffff`) e gli ultimi 32 bit rappresentano l'indirizzo IPv4.
Per esempio, l'indirizzo `0:0:0:0:0:ffff:192.168.30.2` può essere scritto equivalentemente `::ffff:192.168.30.2` che corrisponde all'indirizzo esadecimale `::ffff:c0a8:1e02`.
Analogamente `::ffff:192.168.89.9` è equivalente a `::ffff:c0a8:5909`.

Indirizzi speciali

Per completezza si citano alcuni indirizzi speciali IPv6 destinati ad usi diversi che rappresentano meno del 2% dell'intero spazio di indirizzamento:

`::1/128`
: utilizzati per il loopback verso l'host locale;

`::ffff:0:0/96`
: quindi dall'indirizzo `::ffff:0.0.0.0` all'indirizzo `::ffff:255.255.255.255`, è IPv4-mapped esaminato nella sezione precedente;

`2001:db8::/32`
: quindi dall'indirizzo `2001:db8::` all'indirizzo `2001:db8:ffff:ffff:ffff:ffff:ffff:ffff` per essere utilizzzati nella documentazione e nel codice di esempio;

`fe80::/10`
: è il prefisso per *link-local*, quindi l'indirizzo è valido esclusivamente sullo specifico link fisico;

`ff00::/8`
: è il prefisso per il multicast.

20.2 GNU/Linux come network client

20.2.1 Indirizzamento

Come già esaminato precedentemente, è possibile utilizzare sia il protocollo storico IPv4 sia il protocollo IPv6. La modalità di configurazione sarà analoga e, ovviamente, sarà possibile utilizzare entrambi i protocolli contemporaneamente.

IPv4

Come già accennato nella precedente sezione, connettendo il proprio host via cavo LAN o via wireless, qualora sia già presente un server DHCP la configurazione si attiverà automaticamente.
Se il DHCP non è presente o altre ragioni, è possibile procedere manualmente.
Per procedere è da tener conto le i dettagli dipenderanno dalla propria distribuzione, ad esempio:

Debian GNU/Linux: il file di configurazione è in `/etc/network/interfaces` con esempi commentati in `/usr/share/doc/ifupdown/examples/network-interfaces`;

SUSE: il file di configurazione è in `/etc/sysconfig/network`;

RedHat: i file di configurazione sono nella directory `/etc/sysconfig/network-scripts`.

In ogni caso i file sono facilmente comprensibili e la distribuzione prevede molti esempi. Senza scrivere su file ma solo per effettuare esperimenti, si può utilizzare il comando `ip link` oppure `ifconfig`[66]. Ad esempio, di seguito:

```
1 # ifconfig eth0 192.168.1.10 netmask 255.255.255.0
2 # ifconfig eth0
3 ...
```

Esempio 20.1: *Esempio di utilizzo di* `ifconfig`

in cui:

`ifconfig eth0 192.168.1.10 netmask 255.255.255.0`
: assegnerà l'indirizzo `192.168.18.10` e subnet mask `255.255.255.0` all'interfaccia di rete `eth0`[67];

`ifconfig eth0`
: mostrerà l'output delle interfacce, compreso quanto appena configurato.

Il default gateway può essere aggiunto con il comando `route`:

```
1 # route add -net default gw 192.168.1.1
2 # route -n
3 ...
```

Esempio 20.2: *Esempio di utilizzo di* `route` *per aggiungere il* default gateway

in cui:

`route add -net default gw 192.168.1.1`
: aggiungerà l'indirizzo `192.168.1.1` come *default gateway*;

`route -n`
: mostrerà l'output di della *route table* e si potrà verificare quanto appena impostato.

IPv6

Come l'indirizzamento IPv4, l'indirizzo IPv6 può essere assegnato manualmente o dinamicamente. Per l'assegnazione dinamica, l'IPv6 ha due tipologie di ottenimento dei parametri di rete:

DHCPv6: è la versione aggiornata del DHCP per IPv4, protocollo per ottenere tutte le informazioni per la configurazione;

[66] in aggiunta sisegnala `iwconfig` specifico per le interfacce wirless

[67] per ottenere rapidamente la lista delle schede di rete sul proprio host, si può consultare 20.2.3 a pag. **??**, per vari esempi argomentati v. 21.10 a pag. 311

SLAAC: sigla per *Stateless Address Autoconfiguration*, è un metodo semplificato per ottenere le necessarie[68] informazioni per la comunicazione su IPv6 tramite il *Neighbor Discovery Protocol*;

Senza ripetere quanto già scritto nella precedente sezione, in maniera analoga si configurerà l'interfaccia su protocollo IPv6. Questa volta utilizzeremo il comando `ip`[69]:

```
1 # ip addr add 2001:db8:0:abcd:0:0:0:1358/64 dev eth0
2 # ip a show
3 ...
```

Esempio 20.3: *Esempio di utilizzo di* `ip`

in cui, analogamente all'***Esempio 20.1***:

`ip addr add 2001:db8:0:abcd:0:0:0:1358/64 dev eth0`
: assegnerà l'indirizzo `2001:db8:0:abcd:0:0:0:1358` e subnet mask `/64` all'interfaccia di rete `eth0`;

`ip a show` o semplicemente `ip a`
: mostrerà l'output delle interfacce, compreso quanto appena configurato.

Analogamente all'***Esempio 20.2***, il default gateway può essere impostato sempre con il comando `route` aggiungendo come argomento `-A inet6`[70], oppure utilizzando sempre lo strumento `ip` con la linea generica seguente `ip -6 route add default via <IPV6_ADDRESS> dev <INTERFACE>`[71], ad esempio:

```
1 # ip route add default via 2001:db8:0:abcd:0:0:0:1234 dev eth0
2 # ip -6 route show
3 ...
4 default via 2001:db8:0:abcd:0:0:0:1234 dev eth0
```

Esempio 20.4: *Esempio di utilizzo di* `ip -6 route` *per aggiungere il* default gateway

in cui:

`ip route add default via 2001:db8:0:abcd:0:0:0:1234 dev eth0`
: aggiungerà l'indirizzo `2001:db8:0:abcd:0:0:0:1234` come *default gateway*;

`ip -6 route show` oppure `ip -6 route` oppure `ip -6 r` mostrerà l'output della *route table* limitatamente al protocollo IPv6.

20.2.2 DNS

Per configurare gli indirizzi del[72] DNS server, si modificherà il file `/etc/resolv.conf`:

```
1 search example.com
2 nameserver 192.168.1.1
3 nameserver 192.168.1.2
```

Esempio 20.5: *Contenuto del file* `/etc/resolv.conf`

[68] principalmente l'indirizzamento, gateway e DNS
[69] semplicemente per avere un esempio alternativo rispetto `ifconfig`
[70] equivalentemente `route -A -6`
[71] spesso utilizzata la forma contratta `ip r add ::/0 via ...`, v. `man ip-route`
[72] preferibilmente *dei*

in cui:

`search example.com`
: in fase di ricerca, se la stringa immessa non termina con un punto, il DNS client effettuerà la ricerca di quanto immesso e, in aggiunta, una seconda preponendo la stringa al dominio indicato. Nell'esempio, se l'utente scriverà `www`, il DNS client effettuerà la ricerca di `www.example.com` considerando quanto diditato un sottodominio di `example.com`;

`nameserver` *`n.n.n.n`*
: consulterà il DNS server indicati, passando al successivo qualora si verifichi un timeout nella replica.

Modificando il file, la modifica sarà persistente[73] al riavvio.

20.2.3 Troubleshooting

Questa sezione potrebbe essere saltata se non si verificano problemi, ma è utile precorrere eventuali necessità e continuare di seguito.

`ifconfig`

Il comando è già stato utilizzato nella precedente sezione per configurare l'indirizzo e la subnet mask dell'interfaccia. Di seguito esaminiamo l'output di una interfaccia:

```
1 # ifconfig wlp3s0
2 wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
3         inet 192.168.1.20  netmask 255.255.255.0  broadcast 192.168.1.255
4         inet6 fe80::cdb5:ff49:3845:21e8  prefixlen 64  scopeid 0x20<link>
5         ether 80:a5:89:d3:fd:a7  txqueuelen 1000  (Ethernet)
6         RX packets 13386781  bytes 10431233764 (10.4 GB)
7         RX errors 0  dropped 0  overruns 0  frame 0
8         TX packets 11306901  bytes 4098202189 (4.0 GB)
9         TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Esempio 20.6: *Esempio di utilizzo di* `ifconfig interfaccia`

in cui si è esaminata l'interfaccia `wlp3s0`. Le linee più importanti sono:

`flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500`
: mostra alcuni flag con:

  `UP`
  : in opposizione a `DOWN`, indica che l'interfaccia è abile alla trasmissione e ricezione di datagrammi in quanto il sistema ha allocato le risorse all'interfaccia;

  `BROADCAST` e `MULTICAST`
  : indicano che l'interfaccia è abile nel gestire *broadcast* e *multicast*;

  `RUNNING`
  : indica che i datagrammi sono inviati e ricevuti. Tale flag scompare se, ad esempio, l'interfaccia ethernet ha il cavo ethernet staccato;

[73] talvolta non lo sarà poiché tipicamente il file sarà sovrascritto dal DHCP. Ad esempio, spesso modificando il file `/etc/dhcp/dhclient.conf` la modifica sarà realmente persistente. Oppure si può utilizzare il tool `resolvectl`. In ogni caso, consultare la documentazione della distribuzione

`inet 192.168.1.20 netmask 255.255.255.0 broadcast 192.168.1.255`
riporta l'indirizzo IPv4 e la subnet mask assegnati all'interfaccia;

`inet6 fe80::cdb5:ff49:3845:21e8 prefixlen 64 scopeid 0x20<link>`
mostra l'indirizzo IPv6 che l'host ha assegnato a se stesso. Può apparire un secondo indirizzo IPv6 indicato con `Scope Global` (con gli ultimi sei caratteri presi dal MAC address) se il DHCP avesse supportato tale indirizzamento);

`ether 80:a5:89:d3:fd:a7 txqueuelen 1000 (Ethernet)`
riporta il *MAC address* dell'interfaccia, indirizzo hardware assegnato dal produttore;

Digitando `ifconfig` senza argomenti avremo la lista dell'interfacce attive con il sommario per ciascuna, con `ifconfig -a` la lista sarà per tutte le interfacce.

ping

Il comando `ping`[74] è spesso utilizzato per la verifica di connettività. In sintesi è utilizzato per misurare intervalli di tempo impiegati dai pacchetti *ICMP*[75] a raggiungere i dispositivi di rete target e tornare indietro[76]. In altri termini, è utilizzato per verificare la presenza di tali dispositivi e host, e misurare le latenze nella trasmissione in rete. Ad esempio:

```
1 $ ping 192.168.1.1
2 64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=2.61 ms
3 64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=1.84 ms
4 64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=1.74 ms
5 64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=2.84 ms
6 64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=2.74 ms
7  Ctrl + C
8 --- 192.168.1.1 ping statistics ---
9 5 packets transmitted, 5 received, 0% packet loss, time 4006ms
10 rtt min/avg/max/mdev = 1.739/2.354/2.835/0.466 ms
```

Esempio 20.7: *Esempio di utilizzo di `ping` su rete locale*

in cui l'host mittente ha inviato 5 pacchetti al destinatario, ciascuno a distanza di un secondo dal precedente e, premendo dopo il quinto pacchetto la sequenza @ Ctrl + C @, il tool ha interrotto l'invio e riportato il sommario statistico della trasmissione:

5 pacchetti trasmessi, 5 ricevuti, 0% di pacchetti persi, tempo complessivo 4.006 millisecondi;

`rtt`[77] `min/avg/max/mdev = 1.739/2.354/2.835/0.466 ms` indicano rispettivamente l'intervallo minimo, medio, massimo e deviazione standard.

Ovviamente non è necessario rimanere nella rete locale, ma si può inviare un ping ad un host remoto:

```
1 $ ping -c4 one2zero.it
2 PING one2zero.it (159.65.31.14) 56(84) bytes of data.
3 64 bytes from 159.65.31.14 (159.65.31.14): icmp_seq=1 ttl=53 time=54.7 ms
4 64 bytes from 159.65.31.14 (159.65.31.14): icmp_seq=2 ttl=53 time=50.8 ms
```

[74] ***P**acket **in**ternet **g**roper*

[75] ***Internet Control Message Protocol***, incapsulato in pacchetti IP, progettato per trasmette informazioni di controllo o malfunzionamenti tra i vari dispositivi di rete – v. [81a]

[76] tecnicamente l'invio è una `ECHO_REQUEST`, di ritorno si avrà una `ECHO_REPLY`

[77] ***R**ound **T**rip **T**ime*, è l'intervallo di tempo che intercorre tra l'invio e la conferma dell'arrivo

```
5 64 bytes from 159.65.31.14 (159.65.31.14): icmp_seq=3 ttl=53 time=51.0 ms
6 64 bytes from 159.65.31.14 (159.65.31.14): icmp_seq=4 ttl=53 time=50.9 ms
7
8 --- one2zero.it ping statistics ---
9 4 packets transmitted, 4 received, 0% packet loss, time 3002ms
10 rtt min/avg/max/mdev = 50.820/51.855/54.709/1.648 ms
```

Esempio 20.8: *Esempio di utilizzo di* `ping` *remoto*

in cui con il flag `-c` si è indicato il numero dei pacchetti da trasmettere. Si nota facilmente che i tempi indicati sono notevolmente aumentati poiché il target è un host remoto.
Ci sono diverse flag interessanti da utilizzare[78], i più interessanti:

`-c` *count*
: già utilizzato, per terminare l'invio di pacchetti `ECHO_REQUEST` dopo averne inviati *count*;

`-f`
: per effettuare un *flood*[79] all'host di destinazione: ad ogni `ECHO_REQUEST` (pacchetto inviato) invierà in output il carattere `.`[80], per ogni `ECHO_REPLY` (pacchetto ricevuto) il tool manderà in output un *backspace*[81];

`-i` *interval*
: per modificare l'intervallo di attesa tra invii successivi; Di default l'attesa è 1 secondo tra un invio e il successivo (nessun attesa per il *flood*)

`-s` *packetsize*
: per specificare i data byte da immettere[82] nel pacchetto.

Per esempio, di seguito si setta un intervallo di 0.2 secondi per definire il distanziamento temporale tra gli invii:

```
1 $ ping -i 0,2 192.168.1.1
2 PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
3 64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.36 ms
4 64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=2.83 ms
5 64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=1.86 ms
6 64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=1.77 ms
7 64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=2.85 ms
8 64 bytes from 192.168.1.1: icmp_seq=6 ttl=64 time=1.85 ms
9 64 bytes from 192.168.1.1: icmp_seq=7 ttl=64 time=1.79 ms
10  Ctrl + C
11 --- 192.168.1.1 ping statistics ---
12 7 packets transmitted, 7 received, 0% packet loss, time 1205ms
13 rtt min/avg/max/mdev = 1.364/2.043/2.847/0.526 ms
```

Esempio 20.9: *Esempio di utilizzo di* `ping` *con invio intervallato da 0.2 secondi*

in cui si è utilizzato *0,2* per l'intervallo in accordo con il `locale`.
Per un flood un esempio di seguito:

[78] ovviamente si rimanda al manuale tramite `man ping`
[79] invio massiccio
[80] carattere *punto*
[81] equivale a premere ⟵, carattere di controllo *08* in ASCII code – v. ***Tabella 24.6*** a pag. 401
[82] il default è 56 byte per i dati per complessivi 64 byte sommando l'*ICMP header*

```
1 # sudo ping -f 192.168.1.1
2 PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
3 . Ctrl + C
4 --- 192.168.1.1 ping statistics ---
5 12768 packets transmitted, 12767 received, 0,00783208% packet loss, time
      26453ms
6 rtt min/avg/max/mdev = 0.658/1.987/20.300/0.604 ms, pipe 2, ipg/ewma
      2.071/1.966 ms
```

Esempio 20.10: *Esempio di utilizzo di* `ping` *con flood*

in cui sono stati inviati 12768 pacchetti in poco più di 26 secondi, con la perdita di un pacchetto[83].

Equivalentemente al comando `ping` c'è il comando `ping6`[84] il quale utilizza il protocollo *ICMPv6*[85] protocollo incapsulato nel pacchetto IPv6. Un esempio verso la propria *loopback*:

```
1 $ ping6 ::1
2 PING ::1(::1) 56 data bytes
3 64 bytes from ::1: icmp_seq=1 ttl=64 time=0.046 ms
4 64 bytes from ::1: icmp_seq=2 ttl=64 time=0.062 ms
5 64 bytes from ::1: icmp_seq=3 ttl=64 time=0.068 ms
6 64 bytes from ::1: icmp_seq=4 ttl=64 time=0.049 ms
7 64 bytes from ::1: icmp_seq=5 ttl=64 time=0.049 ms
8  Ctrl + C
9 --- ::1 ping statistics ---
10 5 packets transmitted, 5 received, 0% packet loss, time 4093ms
11 rtt min/avg/max/mdev = 0.046/0.054/0.068/0.008 ms
```

Esempio 20.11: *Esempio di utilizzo di* `ping6` *su loopback*

in cui si notano intervalli di *rtt* piccoli poiché, essendo *loopback*, sia per l'`ECHO_REQUEST` sia per l'`ECHO_REPLY` non c'è attraversamento di rete ma tutto rimane confinato nel proprio host.

Ricordando che i tool per i ping sono per verifica di connettività, qualora si accadano problemi tra l'host di destinazione e il proprio, un approccio alla questione potrebbe essere seguire questi semplici step:

1. `ping` della *loopback*[86];
2. `ping` con target l'ip della propria interfaccia di rete (ethernet, wlan, ...)[87];
3. `ping` del proprio *default gateway*[88]
4. `ping` con target di rete remota.

Ovviamente se il problema è:

- nei primi due punti, la questione è relativa al proprio sistema;

[83] o comunque è stata interrotta l'esecuzione prima della stessa ricezione
[84] equivalentemente `ping -6`
[85] ***Internet Control Message Protocol Version 6****, v. [DC98]*
[86] 127.0.0.1 per IPv4, :: 1 per IPv6
[87] v. precedente sezione su `ifconfig`
[88] per conoscerne l'ip, si consulti l'***Esempio 20.2***

- nel terzo punto la questione è di rete locale (ad esempio può essere utile staccare e riattaccare il cavo di rete);
- nell'ultimo punto, provare da un altro host e, dalla differenza di comportamento, si può intuire se il problema è relativo al nostro host[89] oppure all'host remoto[90].

`dig`

Il tool `dig`[91] è utilizzato per la risoluzione di nomi di dominio attraverso la consultazione del servizio DNS.
Un semplice utilizzo è richiedere l'indirizzo ip di un dominio come di seguito:

```
$ dig one2zero.it
; <<>> DiG 9.16.48-Ubuntu <<>> one2zero.it
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61847
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;one2zero.it.     IN  A

;; ANSWER SECTION:
one2zero.it.    300 IN  A 159.65.31.14

;; Query time: 40 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Mon Apr 29 16:23:18 CEST 2024
;; MSG SIZE  rcvd: 56
```

Esempio 20.12: *Esempio di utilizzo di* `dig` *su dominio*

il cui l'output mostrato è molto *verboso* e in prima approssimazione si può suddividere in diversi blocchi:

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61847
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
```

Esempio 20.13: `HEADER` *relativa all'***Esempio 20.12**

indica che l'operazione è una `QUERY`, con replica;

```
; QUESTION SECTION:
;one2zero.it.                   IN      A
```

Esempio 20.14: `QUESTION SECTION` *relativa all'***Esempio 20.12**

il dominio richiesto

[89] ad esempio è necessario impostare il *proxy aziendale* per navigare

[90] probabilmente dovremo effettuare ulteriori verifiche per diagnosticare il problema o sarà necessario contattare il system/network administrator che talvolta potrà solo confermare la presenza del problema poiché dipendente esternamente al dominio di propria competenza

[91] ***d****omain* ***i****nformation* ***g****roper*

```
;; ANSWER SECTION:
one2zero.it.            300     IN      A       159.65.31.14
```

Esempio 20.15: *`ANSWER SECTION` relativa all'*Esempio 20.12

la replica con record di tipo `A`;

```
;; Query time: 40 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Mon Apr 29 16:23:18 CEST 2024
;; MSG SIZE  rcvd: 56
```

Esempio 20.16: *Sommario relativo all'*Esempio 20.12

il tempo per la riposta, l'indirizzo del server che ha replicato, la data della query e i byte della risposta.

Una domanda più complessa che prevede una replica più corposa è la seguente:

```
$ dig any example.com
; <<>> DiG 9.16.48-Ubuntu <<>> ANY example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25942
;; flags: qr rd ra; QUERY: 1, ANSWER: 7, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;example.com.     IN  ANY

;; ANSWER SECTION:
example.com.    49137 IN  RRSIG NS 13 2 86400 20240506130404
     20240415002326 45023 example.com. L4Thjn22MvFXi63cn9ZhssLiUmH6UhW7Op+/
     lefVF1QsdQjd2M6V35gh 6ji7au5BoQcd25S9lfNsts+amRc1Jg==
example.com.    49137 IN  NS  a.iana-servers.net.
example.com.    49137 IN  NS  b.iana-servers.net.
example.com.    3532  IN  RRSIG AAAA 13 2 3600 20240509090933
     20240418174115 19794 example.com. C70CaN4/mH1rcdrtC6zalL+HNIceeFns/
     IzPRuckmzBXAblIKw+Il9w9 e7foGh+2RM1kNKgb/2IX+6Ia/f12Ng==
example.com.    3532  IN  AAAA  2606:2800:21f:cb07:6820:80da:af6b:8b2c
example.com.    2740  IN  RRSIG A 13 2 3600 20240509112947 20240418174115
     19794 example.com. nCD4pZOK5mI+2ld5OBz/3vKqdUG2IGk1h5iRatZn8eWwDg/SiR+
     S6WZA U/gaeV/doYzpKk+e8d5JQs3xY+xQOw==
example.com.    2740  IN  A 93.184.215.14

;; Query time: 12 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Mon Apr 29 16:53:33 CEST 2024
;; MSG SIZE  rcvd: 453
```

Esempio 20.17: *Esempio di utilizzo di `dig` su dominio con type `any`*

in cui si è richiesto i record di tipo `any`[92] la replica ha prodotto 7 risposte in 12 msec.
È possibile effettuare una query DNS ad uno specifico *name server*:

[92] quindi tutti i record associati, ma è possibile indicare una tipologia quale `A` per gli indirizzi ip (il default per le richieste dirette), `AAAA` per gli indirizzi ipv6, `MX` per i server di posta, `NS` per i *name server*, ... La lista completa dei record utilizzati https://en.wikipedia.org/wiki/List_of_DNS_record_types

```
1 dig @192.168.1.1 example.com
2 dig @a.iana-servers.net example.com
```

Esempio 20.18: *Query* `dig` *diretta ad un name server*

in cui nel primo caso si è utilizzato l'ip del default gateway, nel secondo caso di un name server indicato da una precedente riposta.
Si può richiedere una replica sintetica tramite l'opzione `+short`:

```
1 $ dig +short wikipedia.org
2 185.15.58.224
3 $ dig +short -x 185.15.58.224
4 text-lb.drmrs.wikimedia.org.
```

Esempio 20.19: *Query diretta e inversa con* `+short`

in cui nella prima si richiede l'indirizzo associato al dominio, nella seconda la risoluzione inversa.

20.3 Concetti chiave

datagramma: genericamente frame o pacchetto o altra entità che riporta una partizione dell'informazione a cui è aggiunta un'intestazione contenente informazioni utili affinché possa essere trasmesso al destinatario;

DNS: ***D****omain Name* ***S****ystem*, per la risoluzione di nomi di dominio ad ip e viceversa;

frame: oggetto dei *medium access protocols*, è un contenitore di pacchetto (o partizioni di pacchetto), ciascuno confezionato per poter viaggiare dal mittente al destinatario, solitamente attraverso diversi nodi internet, via segmenti di rete fisici eterogenei (ad esempio collegamenti wireless e/o via cavo);

indirizzo fisico: anche *indirizzo Ethernet* o *indirizzo MAC*, è un codice di 48 bit associato a ciascun dispositivo di rete che implementi lo standard Ethernet. Esso attribuisce un'identità univoca a ciascun host appartenente allo stesso segmento di rete, permettendo così la comunicazione locale diretta tra mittente e destinatario;

indirizzo logico: è un indirizzo IP che identifica univocamente un host, utilizzato nell'header dei datagrammi IP affinché ciascun nodo della rete possa interpretare l'indirizzamento del destinatario e, quindi, procedere correttamente al routing;

LAN: ***l****ocal* ***a****rea* ***n****etwork*, si intende la rete locale, quindi la stessa rete dell'host in oggetto;

pacchetto: oggetto dei*communication protocols*, l'informazione completa del mittente è partizionata in pacchetti, ciascuno dei quali contiene informazioni aggiuntive per poter essere trasferita non solo dal mittente, ma da un nodo internet di transito verso la destinazione corretta;

protocollo: insieme di convenzioni tecniche utilizzate affinché la comunicazione tra nodi possa avvenire;

routing: è il processo di selezione del percorso per il traffico dati in una rete o tra reti;

subnet mask: partendo dal proprio IP, essa permette di conoscere gli indirizzi della LAN, quindi quelli con cui un host dialogherà direttamente senza passare per il gateway;

TCP/IP: sigla che indentifica uno stack di protocolli in cui i protocolli *TCP* e l'*IP* sono i più famosi.

20.4 Comandi introdotti nel capitolo

`dig` ⇒ tool per effettuare query su DNS

`ifconfig` ⇒ configurazione dell'interfaccia di rete

`iwconfig` ⇒ configurazione dell'interfaccia wireless

`ping` ⇒ tool utilizzato per diagnostica di dispositivi di rete

`ping6` ⇒ equivalente al tool `ping` ma per *IPv6*

`route` ⇒ mostra e permette di modificare la tabella di routing

Capitolo 21

Hardware

In questo capitolo saranno esaminati i principali tool per ottenere informazioni relative all'hardware disponibile[1] nel proprio sistema.
Molti di questi strumenti saranno in parte sovrapponibile e alcuni riporteranno anche informazioni software, quindi l'utente valuterà il più congeniale rispetto alla propria esigenza o semplicemente preferenza.
.
Un'avvertenza: questi tool generano veramente tante informazioni e difficilmente si sapranno tutte interpretare. Alcune saranno approfondite nel seguito, ma solo la curiosità dell'utente saprà valutarle al meglio.

21.1 `inxi`

L'invocazione di default del comando `inxi` è di seguito:

```
1 $ inxi
2 CPU: dual core Intel Core i3-4005U (-MT MCP-) speed/min/max: 932/800/1600 MHz
3 Kernel: 5.15.0-105-generic x86_64 Up: 1d 2h 25m Mem: 63.8/7846.0 MiB (0.8%)
4 Storage: 465.76 GiB (45.7\% used) Procs: 16 Shell: Bash inxi: 3.3.13
```

Esempio 21.1: *Esecuzione di* `inxi`

in cui sono riportati:

`CPU: dual core Intel Core i3-4005U (-MT MCP-) speed/min/max: 932/800/1600 MHz`
: le CPU disponibili con le caratteristiche (core, tipologia, clock speed (e min/max se disponibili));

`Kernel: 5.15.0-105-generic x86i_64 Up: 1d 2h 25m Mem: 63.8/7846.0 MiB (0.8%)`
: la versione del kernel, l'architettura e statistiche quali l'uptime, e la memoria;

`Storage: 465.76 GiB (45.7% used) Procs: 16 Shell: Bash inxi: 3.3.13`
: statistiche sullo storage, numero dei processi, shell utilizzata e versione del tool.

Molto sintetico ma essenziale, può essere arricchito attivando i tantissimi flag e opzioni a disposizione. Ad esempio `inxi -B` o `inxi --battery` per visualizzare lo stato delle batterie:

[1] come riportato nel capitolo 12, tali informazioni sono ottenute prevalentemente consultando le directory `/dev`, `proc` e `sys`

```
$ inxi -B
Battery:
  ID-1: BAT0 charge: 17.6 Wh (85.4%) condition: 20.6/37.3 Wh (55.2%)
    volts: 7.6 min: 7.6
```

Esempio 21.2: *Esecuzione di* `inxi -B`

oppure lo stato dettagliato della cpu con `inxi -C` o `inxi --cpu`:

```
$ inxi -C
CPU:
  Topology: Dual Core model: Intel Core i3-4005U bits: 64 type: MT MCP
  L2 cache: 3072 KiB
  Speed: 799 MHz min/max: 800/1600 MHz Core speeds (MHz): 1: 846 2: 836 3: 933
  4: 890
```

Esempio 21.3: *Esecuzione di* `inxi -C`

oppure possono essere combinati (se non conflittuali) per avere informazioni ad esempio sia sui dischi (`-D` oppure `--disk`) sia sulla memoria (`-m` oppure `--memory`):

```
$ inxi --disk --memory
Memory:
  RAM: total: 7.66 GiB used: 4.74 GiB (61.8%)
  RAM Report: permissions: Unable to run dmidecode. Root privileges required.
Drives:
  Local Storage: total: 465.76 GiB used: 385.50 GiB (82.8%)
  ID-1: /dev/sda vendor: Crucial model: CT500MX500SSD1 size: 465.76 GiB
```

Esempio 21.4: *Esecuzione di* `inxi --disk --memory`

in cui si sarà notata la nota `RAM Report: permissions: Unable to run dmidecode. Root privileges required.`[2], per cui ovviamente si avrà un report completo con i privilegi del superutente.
Simpatica l'opzione per ottenere le informazioni sul tempo atmosferico con `-w` o `--weather`:

```
$ inxi --weather
Weather:
  Temperature: 19.94 C (68 F) Conditions: broken clouds
  Current Time: ven 10 mag 2024, 10:51:39 (Europe/Rome) Source: OpenWeatherMap.org
```

Esempio 21.5: *Esecuzione di* `inxi --weather`

in cui si possono cambiare le unità di misura, sorgente dei dati e ovviamente la città specificando la località tramite `-W` *`location_string`* oppure `--weather-location` *`location_string`*:

```
$ inxi -W pescara,italy
Weather:
  Temperature: 19.14 C (66 F) Conditions: overcast clouds
  Current Time: ven 10 mag 2024, 10:58:51 Source: OpenWeatherMap.org
```

Esempio 21.6: *Esecuzione di* `inxi -W pescara,italy`

Per completezza, tra le tante opzioni, probabilmente la più utile è `-v` *`NUMBER`* in cui si indica la verbosità che il tool dovrà esprimere, con valori di verbosità crescente da `1` a `8`[3], più precisamente:

[2] il tool `dmidecode` sarà esaminato in 21.5 a pag. **??**

[3] ovviamente le indicazioni seguenti potrebbero non corrispondere alla propria versione del tool, pertanto si invita a consultare il manuale `man inxi`

`-v 0` è il livello di default, verbosità minimale;

`-v 1` verbosità basic: informazioni di sistema (hostname, kernel, distribuzione, ..., attivate con `-S`), basilari su CPU, sulla grafica (device, display, ..., attivate con `-G`), basilari sui dischi e sul sistema in esecuzione (processi, uptime, memoria, shell, versione del tool attivate con `-I`);

`-v 2` aggiunge su networking card (`-N`), dati sulla macchina (`-M`), batteria (`-B`) se disponibili. Equivale a `inxi --basic`);

`-v 3` aggiunge informazioni avanzate sulle CPU (`-C`) e dati di rete (`-n`);

`-v 4` aggiunge informazioni sulle partizioni (`-P`) e mostra informazioni complete sui dischi (`-D`);

`-v 5` aggiunge informazioni sul device audio (`-A`), memoria (`-m`), sensori (`-s`), label delle partizioni (`-l`), UUID (`-u`), a basilari sui drive ottici;

`-v 6` aggiunge informazioni complete sulle partizioni (`-p`), partizioni non montate (`-o`), complete sui drive ottici (`-d`), USB (`--usb`);

`-v 7` aggiunge informazioni di rete (`-i`);

`-v 8` riporta tutti i dati disponibili, probabilmente utile solo per testing: aggiunge i repository (-r), PCI slot (`--slots`), processi (`-tcp`), admin (`--admin`).

In ultimo è utile utilizzare il flag `-z` (o equivalentemente `--filter`) per attivare i filtri di sicurezza e non rendere visibili alcune informazioni quali l'IP,il MAC, la localizzazione, ..., ad esempio:

```
1 $ inxi -iz
2 Network:
3   Device-1: Realtek RTL8111/8168/8411 PCI Express Gigabit Ethernet driver: r8169
4   Device-2: Qualcomm Atheros QCA9565 / AR9565 Wireless Network Adapter driver: ath9k
5   IF-ID-1: eth0 state: up speed: 10000 Mbps duplex: full mac: <filter>
6   WAN IP: <filter>
```

Esempio 21.7: *Esecuzione di `inxi -iz`*

in cui con il flag `-i` (o equivalentemente `--ip`) in output saranno mostrati le informazioni sui device, IP lato WAN, MAC, ..., ma poiché si è anche attivato il flag di filtraggio, talune informazioni saranno sostituite dalla stringa `<filter>`.

21.2 hwinfo

Questo tool è utilizzato per effettuare verifiche hardware sul sistema e, al termine delle stesse, stende un report.
Se chiamato senza argomenti, saranno attivati quelli di default `--all --log=-` e si intuisce che il report è completo (`--all`) ed il log è inviato allo standard output (`--log=-`), ma di seguito non sarà così invocato per evitare un report estremamente lungo.
È pertanto utile consultare il manuale[4], di seguito invochiamo un report minimale:

```
1 $ hwinfo --short
2 cpu:
3                       Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz, 1250 MHz
4                       Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz, 1200 MHz
5                       Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz, 1256 MHz
6                       Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz, 1300 MHz
```

[4] `man hwinfo`, in cui è presente anche una sezione di esempi

```
keyboard:
  /dev/input/event3    AT Translated Set 2 keyboard
mouse:
  /dev/input/mice      SiGma Micro Usb Mouse
  /dev/input/mice      FocalTechPS/2 FocalTech Touchpad
monitor:
                       LG Display LCD Monitor
graphics card:
                       nVidia GK208BM [GeForce 920M]
                       Intel Haswell-ULT Integrated Graphics Controller
sound:
                       nVidia GK208 HDMI/DP Audio Controller
                       Intel 8 Series HD Audio Controller
                       Intel Haswell-ULT HD Audio Controller
storage:
                       Intel 8 Series SATA Controller 1 [AHCI mode]
network:
                       Qualcomm Atheros QCA9565 / AR9565 Wireless Network Adapter
                       Realtek RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller
network interface:
  eth0                 Ethernet network interface
  lo                   Loopback network interface
disk:
  /dev/sda             CT500MX500SSD1
partition:
  /dev/sda1            Partition
  /dev/sda2            Partition
  /dev/sda3            Partition
  /dev/sda4            Partition
  /dev/sda6            Partition
  /dev/sda7            Partition
cdrom:
  /dev/sr0             HL-DT-ST DVDRAM GUE1N
usb controller:
                       Intel 8 Series USB xHCI HC
bios:
                       BIOS
bridge:
                       Intel 8 Series PCI Express Root Port 1
                       Intel 8 Series LPC Controller
                       Intel 8 Series PCI Express Root Port 4
                       Intel Haswell-ULT DRAM Controller
                       Intel 8 Series PCI Express Root Port 5
                       Intel 8 Series PCI Express Root Port 3
hub:
                       Linux Foundation 2.0 root hub
                       Linux Foundation 3.0 root hub
memory:
                       Main Memory
bluetooth:
                       IMC Networks Bluetooth Device
unknown:
                       FPU
                       DMA controller
                       PIC
                       Keyboard controller
                       Intel Haswell-ULT Thermal Subsystem
                       Intel 8 Series HECI #0
                       Intel 8 Series SMBus Controller
                       Intel 8 Series Thermal
                       Chicony Electronics USB2.0 VGA UVC WebCam
                       Realtek RTS5129 Card Reader Controller
```

***Esempio 21.8:** Esecuzione di `hwinfo --short`*

il quale non sarà realmente minimale riportando sezioni relative all'hardware della macchina quali `cpu`, `keyboard`, `mouse`, `monitor`, `graphics card`, `sound`, `storage`, `network` (device), `network interface`, `disk`, `partition`, `cdrom`, `usb controller`, `bios`, `bridge` (essenzialmente controller PCI), `hub` (per porte usb), `memory`, `bluetooth` e altri che non ha categorizzato.
La categorizzazione appena riportata può essere direttamente indicata nella query di chiamata per ottenere un output relativo a quanto di interesse, ad esempio:

```
$ hwinfo --memory
01: None 00.0: 10102 Main Memory
  [Created at memory.74]
  Unique ID: rdCR.CxwsZFjVASF
```

```
5   Hardware Class: memory
6   Model: "Main Memory"
7   Memory Range: 0x00000000-0x1ea609fff (rw)
8   Memory Size: 7 GB + 512 MB
9   Config Status: cfg=new, avail=yes, need=no, active=unknown
```

Esempio 21.9: *Esecuzione di* `hwinfo --memory`

in cui è stata correttamente riportata solo la sezione `--memory` richiesta.

21.3 lshw

È molto simile al precedente, anch'esso estrae informazioni dettagliate sull'hardware configurato sulla macchina, e anche questo tool, se chiamato senza argomenti, produce un report eccessivo per la maggior parte degli scopi. Pertanto sarà utile valutare le opzioni[5], di seguito un esempio *minimale sanificato*[6]:

```
1  $ sudo lshw -short -sanitize
2  H/W path          Device   Class          Description
3  ====================================================
4                             system         Computer
5  /0                         bus            Motherboard
6  /0/0                       memory         8GiB System memory
7  ...
8  /1                mmc0     bus            MMC Host
9  /2                input0   input          Lid Switch
10 /3                input1   input          Sleep Button
11 /4                input11  input          FocalTechPS/2 FocalTech Touchpad
12 /5                input12  input          Video Bus
13 /6                input13  input          Video Bus
14 /7                input14  input          Asus Wireless Radio Control
15 /8                input15  input          Asus WMI hotkeys
16 /9                input18  input          USB2.0 VGA UVC WebCam: USB2.0 V
17 /a                input2   input          Power Button
18 /b                input3   input          AT Translated Set 2 keyboard
19 /c                input92  input          SIGMACHIP Usb Mouse
```

Esempio 21.10: *Esecuzione di* `lshw -short -sanitize`

in cui il report[7] è suddiviso come di seguito:

`H/W path`
: una stringa che individua il path hardware[8]

`Device`
: mostra i device quali hard drive, network interface, usb device, usb host controller, ...;

`Class`
: riporta la categoria[9]. Le classi principali sono:

[5] v. man lshw

[6] tramite il flag `-sanitize` sono rimosse informazioni potenzialmente sensibili (IP address, serial number, ...)

[7] per un'esaustiva visione v. [Vin]

[8] in `man lshw` sono individuati i file che riportano i path hardware (bus, device, ...)

[9] la quale può essere direttamente specificata nella query; v. prossimo esempio

system: utilizzata per riferirsi all'intera macchina;

bridge: relativo ai *bus converter* interni, quindi *PCI-to-PCI brige*, *PCMCIA controller*, *host bridge*, ...;

memory: individua banchi di memoria;

processor: processori e eventuali RAID controller;

storage: controller dello storage (controller SCSI o IDE);

disk: *random access storage device* (dischi, storage ottici, ...);

tape: *sequential access storage device* (DAT, DDS, ...);

bus: bus per la connessione di device (USB, SCSI, ...);

network: network interface (Ethernet WiFI, ...);

display: schede e adattatori video;

input: device per input dell'utente (tastiera, mouse, ...);

multimedia: device multimediali (device e controller audio o video);

communication: dispositivi di comunicazione (porte serial, modem, ...);

volume: volumi dei dischi (filesystem, swap, ...);

generic: dispositivi generici, utilizzati per device non categorizzabili nelle categorie predefinite.

`Description`
descrizione della relativa entry.

Le classi su individuate possono essere direttamente specificate nella query di chiamata tramite `-C` o `-class`, ad esempio:

```
1  $ sudo lshw -class volume
2  ...
3    *-volume:5
4         description: Linux swap volume
5         vendor: Linux
6         physical id: 7
7         bus info: scsi@0:0.0.0,7
8         logical name: /dev/sda7
9         version: 1
10        serial: 793aed9f-b3d6-4523-b2fe-8069d603c8ae
11        size: 4002MiB
12        capacity: 4002MiB
13        capabilities: nofs swap initialized
14        configuration: filesystem=swap pagesize=4095
```

Esempio 21.11: *Esecuzione di* `lshw -class volume`

in cui è stato troncato l'output riportando solo il volume di swap.
Interessanti i flag `-html`, `-xml` e `-json` i quali formattano l'output utilizzando rispettivamente *html*, *xml* e *json*.

21.4 `lscpu`

È possibile conoscere i dettagli della CPU utilizzando il comando `lscpu`[10] come di seguito:

[10] analogamente tramite `hwinfo --cpu` oppure `lshw -class cpu` oppure un semplice `cat /proc/cpuinfo`

```
$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  4
  On-line CPU(s) list:   0-3
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz
    CPU family:          6
    Model:               69
    Thread(s) per core:  2
    Core(s) per socket:  2
    Socket(s):           1
    Stepping:            1
    CPU max MHz:         1600,0000
    CPU min MHz:         800,0000
    BogoMIPS:            3394.96
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic ...
Virtualization features:
  Virtualization:        VT-x
Caches (sum of all):
  L1d:                   64 KiB (2 instances)
  L1i:                   64 KiB (2 instances)
  L2:                    512 KiB (2 instances)
  L3:                    3 MiB (1 instance)
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0-3
Vulnerabilities:
  Gather data sampling:  Not affected
...
```

Esempio 21.12: *Esecuzione di* `lscpu`

in cui una delle linee più interessanti è `Flags:` la quale riporta l'elenco delle *CPU capabilities*, quindi le caratteristiche supportate dal processore[11].

21.5 dmidecode

Il tool è uno strumento utile per ottenere informazioni sull'hardware decodificando le tabelle *DMI*[12] le quali contengono principalmente una descrizione dei componenti hardware.
L'invocazione senza parametri è particolarmente lunga, di seguito si preferisce ottenere

[11] ad esempio:

`fpu` indica *x87 FPU*, un insieme di istruzioni per il processamento di *floating point* number che originariamente era un'estensione delle istruzioni per 8086 processate da un chip esterno denominato *coprocessiore matematico* poiché lavorava in tandem con la corrispondente CPU (ad esempio il set di istruzioni x87 prevede istruzioni per il calcolo delle funzioni *seno* e *coseno*)

`acpi` è l'acronimo di ***A**dvanced **P**rogrammable **I**nterrupt **C**ontroller*, che identifica un circuito integratocon il compito di gestire IRQ multipli anche simultanei provenienti da differenti sorgenti (utilizzati nei sistemi di elaborazione per multiprocessori simmetrici).

Per un elenco esaustivo v. [comc]

[12] standard **SMBIOS/DMI** (***S**ystem **M**anagement **BIOS**/**D**esktop **M**anagement **I**nterface*, standard legati tra loro e sviluppati dalla **DMTF** (***D**esktop **M**anagement **T**ask **F**orce*)

un output relativo al *type*[13] che può essere una *keyword* tra le seguenti: `bios`, `system`, `baseboard`, `chassis`, `processor`, `memory`, `cache`, `connector` e `slot`.
In analogia con l'***Esempio 21.12***:

```
$ dmidecode -type processor
# dmidecode 3.2
Getting SMBIOS data from sysfs.
SMBIOS 2.8 present.

Handle 0x0011, DMI type 4, 42 bytes
Processor Information
  Socket Designation: SOCKET 0
  Type: Central Processor
  Family: Core i3
  Manufacturer: Intel(R) Corporation
  ID: 51 06 04 00 FF FB EB BF
  Signature: Type 0, Family 6, Model 69, Stepping 1
  Flags:
    FPU (Floating-point unit on-chip)
    VME (Virtual mode extension)
    DE (Debugging extension)
    PSE (Page size extension)
    TSC (Time stamp counter)
    MSR (Model specific registers)
    PAE (Physical address extension)
    MCE (Machine check exception)
    CX8 (CMPXCHG8 instruction supported)
    APIC (On-chip APIC hardware supported)
    SEP (Fast system call)
    MTRR (Memory type range registers)
    PGE (Page global enable)
    MCA (Machine check architecture)
    CMOV (Conditional move instruction supported)
    PAT (Page attribute table)
    PSE-36 (36-bit page size extension)
    CLFSH (CLFLUSH instruction supported)
    DS (Debug store)
    ACPI (ACPI supported)
    MMX (MMX technology supported)
    FXSR (FXSAVE and FXSTOR instructions supported)
    SSE (Streaming SIMD extensions)
    SSE2 (Streaming SIMD extensions 2)
    SS (Self-snoop)
    HTT (Multi-threading)
    TM (Thermal monitor supported)
    PBE (Pending break enabled)
  Version: Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz
  Voltage: 0.7 V
  External Clock: 100 MHz
  Max Speed: 1700 MHz
  Current Speed: 1700 MHz
  Status: Populated, Enabled
  Upgrade: Socket BGA1168
  L1 Cache Handle: 0x000E
  L2 Cache Handle: 0x000F
  L3 Cache Handle: 0x0010
  Serial Number: NULL
  Asset Tag: To Be Filled By O.E.M
  Part Number: To Be Filled By O.E.M
  Core Count: 2
  Core Enabled: 2
  Thread Count: 4
  Characteristics:
    64-bit capable
    Multi-Core
    Hardware Thread
    Execute Protection
    Enhanced Virtualization
    Power/Performance Control
```

Esempio 21.13: *Esecuzione di* `lscpu`

[13] consultando il manuale `man dmidecode` i *type* sono indicati da numeri da 0 a 42 che corrispondono ciascuno ad una lista di componenti (`0 per BIOS, 1 per System, ...`), ma, come nell'esempio di seguito, possono essere utilizzate delle *keyword* con matching *case insensitive* le quali a loro volta corrisponderanno ad uno o più dei precedenti valori numerici

Sempre consultando il manuale, si nota che è possibile stampare solo valori particolarmente interessanti tramite `-s keyword` oppure equivalentemente `--string keyword` quali ad esempio la frequenza del processore:

```
$ sudo dmidecode --string processor-frequency
1700 MHz
```

Esempio 21.14: *Esecuzione di* `dmideconde --string processor-frequency`

che può essere verificata nella seconda parte dell'output relativo all'***Esempio 21.13***.

21.6 lspci

Il tool `lspci` mostra informazioni relative ai bus **PCI**[14] del sistema e ai relativi device connessi.
Di default mostra una lista breve di device, come di seguito:

```
$ lspci
00:00.0 Host bridge: Intel Corporation Haswell-ULT DRAM Controller (rev 0b)
00:02.0 VGA compatible controller: Intel Corporation Haswell-ULT Integrated Graphics Controller (rev 0b)
00:03.0 Audio device: Intel Corporation Haswell-ULT HD Audio Controller (rev 0b)
00:04.0 Signal processing controller: Intel Corporation Haswell-ULT Thermal Subsystem (rev 0b)
00:14.0 USB controller: Intel Corporation 8 Series USB xHCI HC (rev 04)
00:16.0 Communication controller: Intel Corporation 8 Series HECI #0 (rev 04)
00:1b.0 Audio device: Intel Corporation 8 Series HD Audio Controller (rev 04)
00:1c.0 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 1 (rev e4)
00:1c.2 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 3 (rev e4)
00:1c.3 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 4 (rev e4)
00:1c.4 PCI bridge: Intel Corporation 8 Series PCI Express Root Port 5 (rev e4)
00:1f.0 ISA bridge: Intel Corporation 8 Series LPC Controller (rev 04)
00:1f.2 SATA controller: Intel Corporation 8 Series SATA Controller 1 [AHCI mode] (rev 04)
00:1f.3 SMBus: Intel Corporation 8 Series SMBus Controller (rev 04)
00:1f.6 Signal processing controller: Intel Corporation 8 Series Thermal (rev 04)
02:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 10)
03:00.0 Network controller: Qualcomm Atheros QCA9565 / AR9565 Wireless Network Adapter (rev 01)
04:00.0 3D controller: NVIDIA Corporation GK208BM [GeForce 920M] (rev a1)
04:00.1 Audio device: NVIDIA Corporation GK208 HDMI/DP Audio Controller (rev a1)
```

Esempio 21.15: *Esecuzione di* `lspci`

Per ottenere maggiore verbosità si utilizzeranno flag aggiuntivi, i principali[15] in ***Tabella 21.1***. Ad esempio:

FLAG	DESCRIZIONE
`-v`	output più verboso, mostra maggiori dettagli per ogni device
`-nn`	mostra i codici e i nomi sia del fornitore sia del device
`-k`	mostra i driver del kernel deputati alla gestione del device
`-t`	mostra i dispositivi in uno pseudo diagramma ad albero

Tabella 21.1: *Flag principali per* `lspcilspci`

```
$ lspci -v
00:00.0 Host bridge: Intel Corporation Haswell-ULT DRAM Controller (rev 0b)
  Subsystem: ASUSTeK Computer Inc. Haswell-ULT DRAM Controller
```

[14] ***P**eripheral **C**omponent **I**nterconnect*

[15] si consulti il manuale tramite `man lspci` per ottenere la lista esaustiva

```
4   Flags: bus master, fast devsel, latency 0
5   Capabilities: <access denied>
6   Kernel driver in use: hsw_uncore
7
8  00:02.0 VGA compatible controller: Intel Corporation Haswell-ULT Integrated Graphics Controller (
        rev 0b) (prog-if 00 [VGA controller])
9   Subsystem: ASUSTeK Computer Inc. Haswell-ULT Integrated Graphics Controller
10  Flags: bus master, fast devsel, latency 0, IRQ 48
11  Memory at 99400000 (64-bit, non-prefetchable) [size=4M]
12  Memory at a0000000 (64-bit, prefetchable) [size=256M]
13  I/O ports at 5000 [size=64]
14  Expansion ROM at 000c0000 [virtual] [disabled] [size=128K]
15  Capabilities: <access denied>
16  Kernel driver in use: i915
17  Kernel modules: i915
18 ...
```

Esempio 21.16: *Esecuzione di* `lspci -v`

si noterà che, rispetto il precedente:

- attivando il flag `-v` l'output generato è *molto* più esplicativo;
- saranno mostrati i moduli del kernel associati a ciascun device;
- appaiono diverse stringhe `<access denied>` poiché l'invocazione non è avvenuta da superutente.

Un'ultima nota suggerita proprio dal manuale: qualora si voglia riportare un bug per i *PCI device driver*, è buona norma includere nel report l'output del comando `lspci -vvx` di cui è fornito uno spezzone:

```
1  # lspci -vvx
2  00:00.0 Host bridge: Intel Corporation Haswell-ULT DRAM Controller (rev 0b)
3   Subsystem: ASUSTeK Computer Inc. Haswell-ULT DRAM Controller
4   Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B-
        DisINTx-
5   Status: Cap+ 66MHz- UDF- FastB2B+ ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort+ >SERR- <PERR-
        INTx-
6   Latency: 0
7   Capabilities: [e0] Vendor Specific Information: Len=0c <?>
8   Kernel driver in use: hsw_uncore
9  00: 86 80 04 0a 06 00 90 20 0b 00 00 06 00 00 00 00
10 10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
11 20: 00 00 00 00 00 00 00 00 00 00 00 00 43 10 6d 1a
12 30: 00 00 00 00 e0 00 00 00 00 00 00 00 00 00 00 00
13
14 00:02.0 VGA compatible controller: Intel Corporation Haswell-ULT Integrated Graphics Controller (
        rev 0b) (prog-if 00 [VGA controller])
15  Subsystem: ASUSTeK Computer Inc. Haswell-ULT Integrated Graphics Controller
16  Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B-
        DisINTx+
17  Status: Cap+ 66MHz- UDF- FastB2B+ ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR-
        INTx-
18  Latency: 0
19  Interrupt: pin A routed to IRQ 48
20  Region 0: Memory at 99400000 (64-bit, non-prefetchable) [size=4M]
21  Region 2: Memory at a0000000 (64-bit, prefetchable) [size=256M]
22  Region 4: I/O ports at 5000 [size=64]
23  Expansion ROM at 000c0000 [virtual] [disabled] [size=128K]
24  Capabilities: [90] MSI: Enable+ Count=1/1 Maskable- 64bit-
25    Address: fee01004  Data: 0026
26  Capabilities: [d0] Power Management version 2
27    Flags: PMEClk- DSI+ D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
28    Status: D0 NoSoftRst- PME-Enable- DSel=0 DScale=0 PME-
29  Capabilities: [a4] PCI Advanced Features
30    AFCap: TP+ FLR+
31    AFCtrl: FLR-
32    AFStatus: TP-
33  Kernel driver in use: i915
34  Kernel modules: i915
35 00: 86 80 16 0a 07 04 90 00 0b 00 00 03 00 00 00 00
36 10: 04 00 40 99 00 00 00 00 0c 00 00 a0 00 00 00 00
37 20: 01 50 00 00 00 00 00 00 00 00 00 00 43 10 6d 1a
38 30: 00 00 00 00 90 00 00 00 00 00 00 00 ff 01 00 00
39
40 00:03.0 Audio device: Intel Corporation Haswell-ULT HD Audio Controller (rev 0b)
41  Subsystem: ASUSTeK Computer Inc. Haswell-ULT HD Audio Controller
```

```
42 ...
```

Esempio 21.17: *Esecuzione di* `lspci -vvx`

in cui i flag:

`-vv`
: l'output sarà molto più dettagliato mostrando qualsiasi informazione[16] possa essere utile;

`-x`
: mostra un dump esadecimale dello spazio di configurazione associato al device.

21.7 hdparm

Il nome del tool è la contrazione di ***h**ard **d**isk **param**eter* ed è uno strumento molto utile per ottenere statistiche relative agli hard disk, ma anche per settare taluni parametri relativi alla **DMA**[17], alla *cache*, al *power management*, *sleep mode*, ...

In linea di principio la modifica di valori predefiniti subottimali potrebbe migliorare notevolmente le prestazioni, ma urge un avvertimento: non esistendo metodi affidabili per determinare impostazioni ottimali, si sconsiglia di avventurarsi in tale attività poiché taluni parametri settati impropriamente potrebbero mandare in crash il sistema e/o rendere inaccessibili i dati su disco, o semplicemente *corrompere il filesystem*.

Dopo l'avvertimento che mi pare necessario, di seguito si mostrano alcuni test per ottenere informazioni o verificare le performance dell'hardware.
Ad esempio, per ottenere informazioni dettagliate sull hardisk individuato nel sistema da `/dev/sda`:

```
1  # hdparm -I /dev/sda
2  /dev/sda:
3
4  ATA device, with non-removable media
5    Model Number:       CT500MX500SSD1
6    Serial Number:      2150E5F10363
7    Firmware Revision:  M3CR043
8    Transport:          Serial, ATA8-AST, SATA 1.0a, SATA II Extensions, SATA Rev 2.5, SATA Rev 2.6,
         SATA Rev 3.0
9  Standards:
10   Used: unknown (minor revision code 0x006d)
11   Supported: 10 9 8 7 6 5
12   Likely used: 10
13 Configuration:
14   Logical   max current
15   cylinders 16383 0
16   heads   16  0
17   sectors/track 63  0
18   --
19   LBA    user addressable sectors:   268435455
20   LBA48  user addressable sectors:   976773168
21   Logical  Sector size:                   512 bytes
22   Physical Sector size:                  4096 bytes
23   Logical Sector-0 offset:                  0 bytes
24   device size with M = 1024*1024:      476940 MBytes
25   device size with M = 1000*1000:      500107 MBytes (500 GB)
26   cache/buffer size  = unknown
27   Form Factor: 2.5 inch
```

[16] esiste anche il flag `-vvv` in cui in output è mostrato qualsiasi dato anche se non ritenuto interessante dal tool

[17] ***D**irect **m**emory **a**ccess*, caratteristica di dispositivi hardware per l'accesso alla memoria principale indipendentemente della CPU

```
28   Nominal Media Rotation Rate: Solid State Device
29 Capabilities:
30   LBA, IORDY(can be disabled)
31   Queue depth: 32
32   Standby timer values: spec'd by Standard, with device specific minimum
33   R/W multiple sector transfer: Max = 1 Current = 1
34   Advanced power management level: 128
35   DMA: mdma0 mdma1 mdma2 udma0 udma1 udma2 udma3 udma4 udma5 *udma6
36        Cycle time: min=120ns recommended=120ns
37   PIO: pio0 pio1 pio2 pio3 pio4
38        Cycle time: no flow control=120ns  IORDY flow control=120ns
39 Commands/features:
40   Enabled Supported:
41      *  SMART feature set
42         Security Mode feature set
43      *  Power Management feature set
44      *  Write cache
45      *  Look-ahead
46      *  WRITE_BUFFER command
47      *  READ_BUFFER command
48      *  NOP cmd
49      *  DOWNLOAD_MICROCODE
50      *  Advanced Power Management feature set
51      *  48-bit Address feature set
52      *  Mandatory FLUSH_CACHE
53      *  FLUSH_CACHE_EXT
54      *  SMART error logging
55      *  SMART self-test
56 ...
57 Device Sleep:
58   DEVSLP Exit Timeout (DETO): 100 ms (drive)
59   Minimum DEVSLP Assertion Time (MDAT): 10 ms (drive)
60 Checksum: correct
```

Esempio 21.18: *Esecuzione di* `hdparm -I /dev/sda`

Per eseguire un benchmark sul dispositivo:

```
1 # hdparm -tT /dev/sda6
2 /dev/sda6:
3  Timing cached reads:   8058 MB in  1.99 seconds = 4045.20 MB/sec
4  Timing buffered disk reads: 880 MB in  3.01 seconds = 292.77 MB/sec
```

Esempio 21.19: *Esecuzione di* `hdparm -tT /dev/sda6`

in cui i flag:

`-t`
: esegue test sui tempi di lettura del dispositivo, mostrando la velocità di lettura dalla cache senza aver precedentemente spostato tali dati in cache[18];

`-T`
: esegue test sui tempi di lettura della cache del buffer del sistema, senza accesso al disco[19].

Per ottenere valori significativi, l'operazione appena effettuata dovrebbe essere ripetuta più volte su un dispositivo e sistema nel complesso (per quanto possibile) inattivi e con uno spazio libero di diversi MB.
Per una più approfondita conoscenza dello strumento si invita a consultare il manuale[20].

[18] tale valore è un'indicazione di quanto sia veloce l'accesso al disco senza l'*overhead* del file-system. Affinché la misura sia accurata, il buffer è precedentemente svuotato tramite l'operazione `BLKFLSBUF`

[19] tale grandezza è un'indicazione di quanto sia elevato il throughput del processore, cache e memoria del sistema

[20] `man hdparm`

21.8 SMART

Il tool **SMART**[21] è una tecnologia di monitoraggio presente sulla stragrande maggioranza[22] dei dischi. Semplificando, essa memorizza diversi parametri per controllare l'operatività e prevenire alcuni tipi di problematiche o guasti.
Il pacchetto è `smartmontools` che contiene due software:

`smartctl`, per il monitoraggio puntuale dei dischi tramite l'invio di comandi;

`smartd`, demone[23] per il monitoraggio continuativo dei dischi.

Trascurando il demone comunque facilmente configurabile[24] da amministratori interessati[25], possiamo eseguire qualche test tramite il software `smartctl`. Come accennato precedentemente, affinché l'interazione vada a buon fine, i comandi inviati dall'utilità devono essere recepiti da uno storage abilitato al controllo SMART[26].
Le principali[27] opzioni sono in ***Tabella 21.2***.

OPZIONI		DESCRIZIONE
BREVE	ESTESA	
`-i`	`--info`	mostra informazioni sul disco (device model number, ...), informazioni sul supporto SMART (available? enabled?)
`-s` *VALUE*	`--smart=`*VALUE*	abilita o disabilita SMART sul device. Il valore *VALUE* dovrà rispettivamente essere `on` oppure `off`
`-H`	`--health`	invia in output lo *stato di salute* del device segnalando eventuali anomalie
`-t` *TEST*	`--test=`*TEST*	esegue un test del tipo *TEST* sul device. Per un elenco completo consultare il manuale, in ogni caso i principali sono `short` e `long`
`-l` *TYPR*	`--log=`*TYPE*	mostra i log del device del tipo *TYPE*. I principali *TYPE* sono `selftest` e `error`

Tabella 21.2: *Opzioni principali per* `smartctl`

Ad esempio, per controllare se il disco `/dev/sda` è abile alle chiamate SMART:

```
1 # smartctl -i /dev/sda
2 ...
3 === START OF INFORMATION SECTION ===
4 Model Family:     Crucial/Micron BX/MX1/2/3/500, M5/600, 1100 SSDs
```

[21] ***S**elf-**M**onitoring, **A**nalysis, and **R**eporting **T**echnology*
[22] probabilmente tutti quelli recenti
[23] i *demoni* saranno approfonditi nel capitolo **??**
[24] come spesso accade, nel pacchetto c'è la documentazione e tanti esempi di configurazione. Una valida guida è [comb]
[25] tendenzialmente per amministratori di storage importanti costituito da molte unità
[26] qualora non fosse già abilitato, dalle impostazioni del BIOS del sistema si potrà effettuare tale abilitazione
[27] si consulti il manuale tramite `man smartctl` per ottenere la lista esaustiva

```
Device Model:     CT500MX500SSD1
Serial Number:    2150E5F10363
LU WWN Device Id: 5 00a075 1e5f10363
Firmware Version: M3CR043
User Capacity:    500.107.862.016 bytes [500 GB]
Sector Sizes:     512 bytes logical, 4096 bytes physical
Rotation Rate:    Solid State Device
Form Factor:      2.5 inches
Device is:        In smartctl database [for details use: -P show]
ATA Version is:   ACS-3 T13/2161-D revision 5
SATA Version is:  SATA 3.3, 6.0 Gb/s (current: 6.0 Gb/s)
Local Time is:    Wed May 15 16:12:31 2024 CEST
SMART support is: Available - device has SMART capability.
SMART support is: Enabled
```

Esempio 21.20: *Esecuzione di* `smartctl -i /dev/sda`

Qualora il supporto SMART fosse *available* ma non fosse *Enable*, lo si potrà abilitare con:

```
# smartctl -s on /dev/sda
...

=== START OF ENABLE/DISABLE COMMANDS SECTION ===
SMART Enabled.
```

Esempio 21.21: *Esecuzione di* `smartctl -s on /dev/sda`

invece, qualora lo si voglia disabilitare[28]:

```
# smartctl -s off /dev/sda
...
=== START OF ENABLE/DISABLE COMMANDS SECTION ===
SMART Disabled. Use option -s with argument 'on' to enable it.
```

Esempio 21.22: *Esecuzione di* `smartctl -s off /dev/sda`

che subito sarà riabilitato per continuare con gli esempi.
Per ottenere lo *stato di salute* del device:

```
# smartctl -H /dev/sda
...
=== START OF READ SMART DATA SECTION ===
SMART overall-health self-assessment test result: PASSED
```

Esempio 21.23: *Esecuzione di* `smartctl -H /dev/sda`

in cui `PASSED` segnala un *buono stato di salute*, ma se sono segnalati anomalie/errori, il disco avrà possibili guasti nell'immediato futuro.
Per eseguire un test di qualche minuto:

```
# smartctl -t short /dev/sda
...
=== START OF OFFLINE IMMEDIATE AND SELF-TEST SECTION ===
Sending command: "Execute SMART Short self-test routine immediately in off-line mode".
Drive command "Execute SMART Short self-test routine immediately in off-line mode" successful.
Testing has begun.
Please wait 2 minutes for test to complete.
Test will complete after Wed May 15 17:10:50 2024 CEST
Use smartctl -X to abort test.
```

Esempio 21.24: *Esecuzione di* `smartctl -t short /dev/sda`

[28] perché ciò? Sicuro??

e il test breve del disco è partito in background. Dopo diversi minuti:

```
# smartctl -l selftest /dev/sda
...
=== START OF READ SMART DATA SECTION ===
SMART Self-test log structure revision number 1
Num  Test_Description    Status                  Remaining  LifeTime(hours)  LBA_of_first_error
# 1  Short offline       Completed without error       00%     12872
```

Esempio 21.25: *Esecuzione di* `smartctl -l selftest /dev/sda`

consultanto il log si osserva che lo stato è[29] `Completed without error`.

21.9 `lsusb`

Nel pacchetto `usbutils` c'è il tool `lsusb` che semplicemente elenca i dispositivi e controller USB della propria macchina. Ad esempio:

```
$ lsusb
Bus 002 Device 011: ID 0951:1666 Kingston Technology DataTraveler 100 G3/G4/SE9 G2
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 004: ID 0bda:0129 Realtek Semiconductor Corp. RTS5129 Card Reader Controller
Bus 001 Device 061: ID 13d3:3423 IMC Networks
Bus 001 Device 002: ID 04f2:b483 Chicony Electronics Co., Ltd USB2.0 VGA UVC WebCam
Bus 001 Device 062: ID 1c4f:0051 SiGma Micro Usb Mouse
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Esempio 21.26: *Esecuzione di* `lsusb`

si nota che ogni dispositivo è identificato da una coppia di numeri esadecimali, ad esempio lo storage usb nella prima riga da `0951:1666`, in cui:

- le prime 4 cifre identificano l'id del produttore[30];
- le ultime 4 cifre identificano l'id del dispositivo[31].

Dalle prime due linee si comprende che lo storage è stato inserito correttamente sul `Bus 002` che è un `hub 3.0`, quindi il dispositivo di storage comunicherà attraverso il protocollo *usb 3*. Per verificarlo, si può chiedere un check verboso[32] come di seguito:

```
$ lsusb -vs 2:11
lsusb -vs 2:11 2> /dev/null

Bus 002 Device 011: ID 0951:1666 Kingston Technology DataTraveler 100 G3/G4/SE9 G2
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               3.20
  bDeviceClass            0
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0         9
  idVendor           0x0951 Kingston Technology
  idProduct          0x1666 DataTraveler 100 G3/G4/SE9 G2
  bcdDevice            1.10
  iManufacturer           1
  iProduct                2
  iSerial                 3
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                9
    bDescriptorType        2
    wTotalLength       0x002c
```

[29] fortunatamente per il sottoscritto ☺

[30] nel caso `0951` corrisponde a *Kingston Technology*, sarà verificato a breve

[31] nel caso `1666` corrisponde a *DataTraveler 100 G3/G4/SE9 G2*, sarà verificato a breve

[32] v. `man lsusb`

```
24      bNumInterfaces          1
25      bConfigurationValue     1
26      iConfiguration          0
27      bmAttributes         0x80
28        (Bus Powered)
29      MaxPower              504mA
30      Interface Descriptor:
31        bLength                 9
32        bDescriptorType         4
33        bInterfaceNumber        0
34        bAlternateSetting       0
35        bNumEndpoints           2
36        bInterfaceClass         8 Mass Storage
37        bInterfaceSubClass      6 SCSI
38        bInterfaceProtocol     80 Bulk-Only
39        iInterface              0
40        Endpoint Descriptor:
41          bLength                 7
42          bDescriptorType         5
43          bEndpointAddress     0x81  EP 1 IN
44          bmAttributes            2
45            Transfer Type            Bulk
46            Synch Type               None
47            Usage Type               Data
48          wMaxPacketSize     0x0400  1x 1024 bytes
49          bInterval               0
50          bMaxBurst               3
51        Endpoint Descriptor:
52          bLength                 7
53          bDescriptorType         5
54          bEndpointAddress     0x02  EP 2 OUT
55          bmAttributes            2
56            Transfer Type            Bulk
57            Synch Type               None
58            Usage Type               Data
59          wMaxPacketSize     0x0400  1x 1024 bytes
60          bInterval               0
61          bMaxBurst               3
```

Esempio 21.27: *Esecuzione di* `lsusb -s`

in cui `2:11` specifica[33] l'id del bus e l'id del device. Riporto alcune linee di interesse:

`bcdUSB 3.20`
: identifica il protocollo di comunicazione utilizzato;

`idVendor 0x0951 Kingston Technology`
: identifica il produttore;

`idProduct 0x1666 DataTraveler 100 G3/G4/SE9 G2` identifica il device.

Qualora lo storage fosse stato inserito nel bus dell'hub 2.0, da `lsusb` si sarebbe ottenuto:

```
1 $ lsusb
2 ...
3 Bus 001 Device 065: ID 0951:1666 Kingston Technology DataTraveler 100 G3/G4/SE9 G2
4 Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Esempio 21.28: *Esecuzione di* `lsusb` *con storage messo su hub con potenzialità minori*

Per verificarlo, si può chiedere un check verboso[34] come di seguito:

```
1 $ lsusb -vs 1:65 2>/dev/null | grep bcdUSB
2   bcdUSB               2.10
```

Esempio 21.29: *Esecuzione di* `lsusb -s`

[33] analogamente si sarebbe potuto scrivere `lsusb -vd 0951:1666` con l'id del produttore e del device ma, ad esempio, per evitare problemi con eventuale storage omogeneo (esempio due pen usb identiche), è da preferire la versione utilizzata nell'esempio

[34] v. `man lsusb`

21.10 Network interface

Uno dei maggiori interessi hardware di un sistemista riguardano la lista delle interfacce di rete. Di seguito sarà fornita una lista di comandi, alcuni già incontrati[35], per visualizzare l'elenco di tali interfacce.

inxi

Da 21.1 a pag. 295, filtrando tramite l'attivazione del flag `--network` (o equivalentemente `-N`) si ottiene la lista dei device di rete:

```
1 $ inxi --network
2 Network:
3   Device-1: Realtek RTL8111/8168/8411 PCI Express Gigabit Ethernet driver: r8169
4   Device-2: Qualcomm Atheros QCA9565 / AR9565 Wireless Network Adapter driver: ath9k
```

Esempio 21.30: *Esecuzione di* `inxi --network`

Nella lista precedente c'è l'elenco dei device incluso il driver, ma attivando il flag avanzato `--network-advanced` (o equivalentemente `-n`) è possibile ottenere informazioni aggiuntive quali il nome delle interfacce, la velocità, mac address, ...:

```
1 $ inxi --network-advanced
2 Network:
3   Device-1: Realtek RTL8111/8168/8411 PCI Express Gigabit Ethernet driver: r8169
4   IF: enp2s0 state: down mac: d0:17:c2:11:41:c3
5   Device-2: Qualcomm Atheros QCA9565 / AR9565 Wireless Network Adapter driver: ath9k
6   IF: wlp3s0 state: up mac: 80:a5:89:d3:fd:a7
7   IF-ID-1: br-446c25032b35 state: down mac: 02:42:69:d7:57:7d
8   IF-ID-2: docker0 state: down mac: 02:42:ef:66:00:ba
9   IF-ID-3: lxcbr0 state: up speed: 10000 Mbps duplex: unknown mac: 00:16:3e:00:00:00
```

Esempio 21.31: *Esecuzione di* `inxi --network-advanced`

hwinfo

Da 21.2 a pag. 297, filtrando tramite l'attivazione del flag `--netcard` si ottiene la lista dei device di rete:

```
1 $ hwinfo --short --netcard
2 network:
3   wlp3s0               Qualcomm Atheros QCA9565 / AR9565 Wireless Network Adapter
4   enp2s0               Realtek RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller
```

Esempio 21.32: *Esecuzione di* `hwinfo --short --netcard`

in cui si è utilizzato il flag `--short` per ottenere un report contenuto.

lshw

Da 21.3 a pag. 299, tramite il tool `lshw` si otterrà un analogo report a quello della precedente sezione filtrando tramite la classe `network`:

```
1 $ lshw -class -network -short
2 H/W path          Device          Class          Description
3 ============================================================
4 /0/100/1c.2/0     enp2s0          network        RTL8111/8168/8411 PCI Express Gigabit Eth
5 /0/100/1c.3/0     wlp3s0          network        QCA9565 / AR9565 Wireless Network Adapter
```

Esempio 21.33: *Esecuzione di* `lshw -class -network -short`

[35] per lettura complementare, alcuni saranno nuovi e approfonditi nei prossimi capitoli

oppure più verboso e interessante:

```
$ lshw -class -network -sanitize
  *-network
       description: Ethernet interface
       product: RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller
       vendor: Realtek Semiconductor Co., Ltd.
       physical id: 0
       bus info: pci@0000:02:00.0
       logical name: enp2s0
       version: 10
       serial: [REMOVED]
       capacity: 1Gbit/s
       width: 64 bits
       clock: 33MHz
       capabilities: pm msi pciexpress msix vpd bus_master cap_list ethernet physical tp mii 10bt
      10bt-fd 100bt 100bt-fd 1000bt-fd autonegotiation
       configuration: autonegotiation=on broadcast=yes driver=r8169 driverversion=5.15.0-105-
      generic firmware=rtl8168g-3_0.0.1 04/23/13 latency=0 link=no multicast=yes port=twisted pair
       resources: irq:18 ioport:4000(size=256) memory:99904000-99904fff memory:99900000-99903fff
  *-network
       description: Wireless interface
       product: QCA9565 / AR9565 Wireless Network Adapter
       vendor: Qualcomm Atheros
       physical id: 0
       bus info: pci@0000:03:00.0
       logical name: wlp3s0
       version: 01
       serial: [REMOVED]
       width: 64 bits
       clock: 33MHz
       capabilities: pm msi pciexpress bus_master cap_list rom ethernet physical wireless
       configuration: broadcast=yes driver=ath9k driverversion=5.15.0-105-generic firmware=N/A ip=[
      REMOVED] latency=0 link=yes multicast=yes wireless=IEEE 802.11
       resources: irq:19 memory:99800000-9987ffff memory:99880000-9988ffff
```

Esempio 21.34: *Esecuzione di* `lshw -class -network -sanitize`

in cui il flag `-sanitize` è stato nuovamente utilizzato per sostituire testo contenente informazioni potenzialmente sensibili con la stringa `REMOVED`.

`lspci`

Da 21.6 a pag. 303, filtrando adeguatamente si ottiene la lista dei device di rete:

```
$ lspci | grep -Ei 'eth|network|ethernet|wireless|wifi'
02:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit
      Ethernet Controller (rev 10)
03:00.0 Network controller: Qualcomm Atheros QCA9565 / AR9565 Wireless Network Adapter (rev 01)
```

Esempio 21.35: *Esecuzione di* `lspci | grep -Ei 'eth|network|ethernet|wireless|wifi'`

e dall'elenco appaiono due dispositivi di rete. Per consultare i dettagli della prima:

```
$ lspci -vvs 02:00.0
02:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit
      Ethernet Controller (rev 10)
  Subsystem: ASUSTeK Computer Inc. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller
  Flags: bus master, fast devsel, latency 0, IRQ 18
  I/O ports at 4000 [size=256]
  Memory at 99904000 (64-bit, non-prefetchable) [size=4K]
  Memory at 99900000 (64-bit, non-prefetchable) [size=16K]
  Capabilities: <access denied>
  Kernel driver in use: r8169
  Kernel modules: r8169
```

Esempio 21.36: *Esecuzione di* `lspci -vvs 02:00.0`

ip

Il comando `ip` è utilizzato per consultare o manipolare il routing, i device, le interfacce e i tunnel. Senza entrare nei dettagli[36], per ottenere la lista di interfacce:

```
$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen
    1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp2s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT
    group default qlen 1000
    link/ether d0:17:c2:11:41:c3 brd ff:ff:ff:ff:ff:ff
3: wlp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DORMANT group
    default qlen 1000
    link/ether 80:a5:89:d3:fd:a7 brd ff:ff:ff:ff:ff:ff
4: lxcbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group
    default qlen 1000
    link/ether 00:16:3e:00:00:00 brd ff:ff:ff:ff:ff:ff
5: br-446c25032b35: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode
    DEFAULT group default
    link/ether 02:42:69:d7:57:7d brd ff:ff:ff:ff:ff:ff
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT
    group default
    link/ether 02:42:ef:66:00:ba brd ff:ff:ff:ff:ff:ff
```

Esempio 21.37: *Esecuzione di* `ip link`

in cui:

`lo`
: è la *loopback interface*;

`enp2s0`
: la prima interfaccia di rete ethernet sulla macchina. Storicamente e talvolta su alcune distribuzioni (dipende dal driver) l'interfaccia è `eth0`;

`wlp3s0`
: l'interfaccia wireless. Storicamente e talvolta su alcune distribuzioni (dipende dal driver) l'interfaccia è `wlan0`;

`lxcbr0`
: interfaccia del container attivo su tecnologia *lxc*[37] configurato in *bridge mode*;

`br-446c25032b35` e `docker0`
: sono le interfacce create da docker[38] configurate in *bridge mode*.

Altri nomi di interfacce che spesso si incontrano sono:

`ppp`*n*
: identificano interfacce configurate in *Point to Point Protocol*, tendenzialmente utilizzate per connessioni *dial up modem* o anche per modem USB su connessioni wireless, in cui ad *n* si sostituirà tendenzialmente 0;

`vmnet`*n* o `vboxnet`*n*
: identificano interfacce di *virtual machine*, in cui ad *n* si sostituiranno numeri tendenzialmente a partire da 0.

[36] v. **??** a pag. **??**
[37] approfondito in **??** a pag. **??**
[38] approfondite in **??** a pag. **??**

ifconfig

Il tool `ip` oramai rimpiazza il vecchio ma ottimo comando `ifconfig` poiché quest'ultimo è stato deprecato sulle distribuzioni moderne.
In ogni caso `ifconfig` è ancora molto utilizzato, per cui di seguito sarà utilizzato per ottenere la lista delle interfacce:

```
$ ifconfig -a
br-446c25032b35: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.24.0.1  netmask 255.255.0.0  broadcast 172.24.255.255
        ether 02:42:69:d7:57:7d  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ef:66:00:ba  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp2s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether d0:17:c2:11:41:c3  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Loopback locale)
        RX packets 754117  bytes 85040419 (85.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 754117  bytes 85040419 (85.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lxcbr0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.3.1  netmask 255.255.255.0  broadcast 10.0.3.255
        inet6 fe80::216:3eff:fe00:0  prefixlen 64  scopeid 0x20<link>
        ether 00:16:3e:00:00:00  txqueuelen 1000  (Ethernet)
        RX packets 53240  bytes 3352231 (3.3 MB)
        RX errors 0  dropped 12  overruns 0  frame 0
        TX packets 77771  bytes 89596608 (89.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.20  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::cdb5:ff49:3845:21e8  prefixlen 64  scopeid 0x20<link>
        ether 80:a5:89:d3:fd:a7  txqueuelen 1000  (Ethernet)
        RX packets 19962857  bytes 20985216620 (20.9 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9499694  bytes 6408357193 (6.4 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Esempio 21.38: *Esecuzione di* `ifconfig -a`

in cui è stato utilizzato il flag `-a` per ottenere tutte le interfacce disponibili, anche se *down*. L'output prodotto è sovrapponibile a quanto esaminato precedentemente per cui non sarà riesaminato.

nmcli

Se sul sistema è presente il tool `nmcli`[39] sarà sufficiente eseguire il comando di seguito:

```
$ nmcli device status
DEVICE           TYPE      STATE           CONNECTION
wlp3s0           wifi      collegato       iliadbox-69F2E3
```

[39] ***N**etwork**M**anager **C**ommand **L**ine **I**nterface*, argomentato in **??** a pag. **??**

```
4 lxcbr0           bridge    collegato        lxcbr0
5 br-446c25032b35  bridge    collegato        br-446c25032b35
6 docker0          bridge    collegato        docker0
7 enp2s0           ethernet  non disponibile  --
8 lo               loopback  non gestito      --
```

Esempio 21.39: *Esecuzione di* `nmcli device status`

netstat

Il comando `netstat`[40] è anch'esso obsoleto ma ancora molto popolare, per cui di seguito sarà utilizzato per ottenere la lista delle interfacce attivando il flag `-i`:

```
1 $ netstat -i
2 Tabella dell'interfaccia del kernel
3 Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR
      Flg
4 br-446c2  1500        0      0      0 0             0      0      0      0
      BMU
5 docker0   1500        0      0      0 0             0      0      0      0
      BMU
6 enp2s0    1500        0      0      0 0             0      0      0      0
      BMU
7 lo       65536   761763      0      0 0        761763      0      0      0
      LRU
8 lxcbr0    1500    53262      0     12 0         78129      0      0      0
      BMRU
9 wlp3s0    1500 20201206      0      0 0       9643841      0      0      0
      BMRU
```

Esempio 21.40: *Esecuzione di* `netstat -i`

tcpdump

Il tool `tcpdump`[41] è uno strumento progettato per l'analisi e il dump del traffico di rete. Di seguito sarà utilizzato semplicisticamente per ottenere la lista delle interfacce attivando il flag `--list-interfaces`:

```
1 $ tcpdump --list-interfaces
2 1.wlp3s0 [Up, Running]
3 2.lxcbr0 [Up, Running]
4 3.lo [Up, Running, Loopback]
5 4.any (Pseudo-device that captures on all interfaces) [Up, Running]
6 5.enp2s0 [Up]
7 6.docker0 [Up]
8 7.br-446c25032b35 [Up]
9 8.bluetooth-monitor (Bluetooth Linux Monitor) [none]
10 9.nflog (Linux netfilter log (NFLOG) interface) [none]
11 10.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
12 11.bluetooth0 (Bluetooth adapter number 0) [none]
```

Esempio 21.41: *Esecuzione di* `tcpdump --list-interfaces`

in cui nella lista appaiono più interfacce delle precedenti poiché il tool può mettersi in ascolto anche su interfacce virtuali, la cui utilità è per ciascuna descritta nella stessa linea di output.

[40] ***Net****work* ***Stat****istic*, argomentato in **??** a pag. **??**

[41] esaminato in **??** a pag. **??**

/proc/net/dev

Il suddetto file è prodotto dal kernel è contiene informazioni riguardo lo stato dei dispositivi di network. Tra le statistiche sono riportate il numero di pacchetti ricevuti e spediti, il numeri degli errori e collisioni avvenute, ...

```
1 $ cat /proc/net/dev
2 Inter-|   Receive                                                |  Transmit
3  face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets errs drop fifo
       colls carrier compressed
4     lo: 89005462  784369    0    0    0     0          0         0 89005462  784369    0    0    0
          0       0          0
5 enp2s0:       0       0    0    0    0     0          0         0        0       0    0    0    0
          0       0          0
6 wlp3s0: 21437132789 20443847    0    0    0     0          0         0 6457848546 9806477    0    0
          0     0       0          0
7 lxcbr0: 3358539   53289    0   12    0     0          0       230 89763253   78597    0    0    0
          0       0          0
8 br-446c25032b35:       0       0    0    0    0     0          0         0        0       0    0
       0    0     0       0          0
9 docker0:       0       0    0    0    0     0          0         0        0       0    0    0    0
          0       0          0
```

Esempio 21.42: *Esecuzione di* `cat /proc/net/dev`

Certamente è da considerare una visione mediocre ma è estremamente funzionale, ricordandoci che stiamo direttamente consultando un file di statistiche direttamente prodotto dal kernel, quindi non utilizzando tool intermedi.

21.11 Comandi introdotti nel capitolo

`dmidecode` ⇒ effettua il decoding delle tabelle DMI (descrizione componenti hardware)
`inxi` ⇒ tool per ottenere informazioni sul sistema e non solo
`hdparm` ⇒ tool per ottenere e modificare parametri su hd
`hwinfo` ⇒ diagnostica sull'hardware con report
`ip` ⇒ per consultazione e manipolazione del networking
`ifconfig` ⇒ rimpiazzato da `ip`, ma ancora popolare
`lscpu` ⇒ mostra informazioni riguardanti le CPU
`lshw` ⇒ report dell'hardware della macchina
`lspci` ⇒ mostra informazioni relative ai bus PCI
`lsusb` ⇒ elenca i dispositivi e controller USB
`netstat` ⇒ tool obsoleto ma ancora molto popolare per le statistiche di rete
`nmcli` ⇒ ***N**etwork**M**anager **C**ommand **L**ine **I**nterface*
`tcpdump` ⇒ strumento per il dump del traffico di rete

Parte III

Programmazione

Capitolo 22

Introduzione al Bash scripting

FINORA sono stati inseriti solo comandi uno per volta, quindi al ritorno del prompt si immette il comando successivo, oppure concatenati tramite pipeline. Questo capitolo sarà un'introduzione[1] allo scripting via *Bash* attraverso l'inserimento in un file, creato con un editor qualunque, di una successione di comandi in maniera programmatica tali da indirizzarci verso task più complessi e anche l'automazione.

22.1 La shell come linguaggio di scripting

Questo capitolo è incentrato sulla shell *Bash*, ma ovviamente si potrà optare differentemente con sostanzialmente poche differenze. Per ciascuna shell si consulti la documentazione appopriata, in particolare per la *Bash* è previsto il manuale `man bash`.

22.2 Command line

Nei precedenti capitoli si è utilizzata la shell solo in modalità *command line*, con una sintassi generica del tipo `command [OPTIONS] arguments`, ad esempio:

```
1 $ comando1
2 $ comando2 -options
3 $ comando3 arguments
4 output3
5 $
```

Esempio 22.1: *Esecuzione di una sequenza di comandi*

in cui i comandi `comando1`, `comando2` e `comando3` (con eventuali opzioni e/o argomenti) hanno svolto il proprio rispettivo task, eventualmente restituendo un *output* all'utilizzatore.
Questo modo di interagire con la shell è definito *interattivo*, in cui la sequenza dell'utilizzatore procede dopo ogni restituzione del prompt da parte della shell, quindi la shell comunica il completamento del task assegnato mostrando il prompt per l'attesa di un nuovo comando.

[1] per un'argomentazione approfondita si consulti il manuale `man bash` o [Foua]

22.3 Creazione file

Per la creazione del file sarà necessario utilizzare un editor di testo[2], poi si sceglierà un nome file il più significativo e possibilmente aderente l'oggetto del task.

22.3.1 Estensione

Come per tutti i file, l'estensione utilizzata non è significativa per il sistema, ma per convenzione essi terminano con `.sh` o `.bash`.

22.4 Script eseguibile

Per eseguire lo script con nome `test.bash`, si hanno due opzioni:

eseguire l'interprete con argomento il file creato come nell'esempio di seguito:

```
1 $ bash test.bash
2 ...
```

Esempio 22.2: *Eseguire lo script lanciando l'interprete ed indicando come argomento il testo creato*

e con `...` eventuale output del task.

rendere lo script eseguibile, in maniera che sia lo script stesso a eseguire l'interprete come nel seguente esempio:

```
1 $ ./test.bash
2 ...
```

Esempio 22.3: *Eseguire lo script come argomento dell'interprete*

e con `...` eventuale output del task.

Questa sezione descriverà i passaggi necessari per rendere lo script eseguibile.

22.4.1 Intestazione

In generale gli script iniziano con uno *shabang*[3] che consiste nella sequenza di caratteri `#!` collocata esattamente all'inizio di ciascuno.
Immediatamente di seguito allo *shebang* è indicato il path assoluto dell'interprete da utilizzare per eseguire lo script medesimo.
Nel caso del *bash*, lo script inizierà con la sequenza:

```
1 #!/usr/bin/bash
2 ...
```

Esempio 22.4: *Intestazione del Bash*

con:

`/usr/bin/bash` il path assoluto[4];

[2] per dubbi si consulti il capitolo 6
[3] definito anche *shebang* o *hashbang*
[4] trovato tramite `which bash`

. . . il contenuto.

Poiché non c'è consenso tra le varie distribuzioni nel collocare l'interprete, per una maggior portabilità si può utilizzare l'intestazione alternativa di seguito:

```
1 #!/usr/bin/env bash
2 ...
```

Esempio 22.5: *Intestazione del Bash tramite* `env`

che può essere facilmente adattata per i vari interpreti[5].

22.4.2 Script eseguibile

Per tale operazione è sufficiente digitare dal prompt il comando `chmod`[6] come mostrato:

```
1 $ chmod +x script.sh
```

Esempio 22.6: *Intestazione del Bash tramite* `env`

22.5 Contenuto

Nelle prossime sezioni saranno mostrati i costrutti base per lo scripting.

22.5.1 Sequenza di comandi

Di seguito un semplicissimo script con una breve sequenza di comandi:

```
1 #!/usr/bin/env bash
2
3 date
4 echo "Hello world!!"
```

Esempio 22.7: *Script con sequenza di comandi*

in cui nello script è presente, dopo l'intestazione, una sequenza di comandi.

22.5.2 Commenti

I commenti iniziano con il carattere # come mostrato di seguito:

```
1 #!/usr/bin/env bash
2 # Questo è un semplice script in cui è inserita una sequenza di
      comandi
3
4 ...
```

Esempio 22.8: *Script con commento*

è stato aggiunto un commento rispetto quanto mostrato nell'***Esempio 22.7***.

Per inserire più linee di commento di seguito, oltre che ripetendo il carattere # per ciascuna linea, è possibile inserire commenti *multiline*:

[5] per alcuni esempi v. 22.9.2 a pag. 342

[6] la trasformazione di un file in eseguibile è argomentata in 18.5 a pag. 235

```
1 #!/usr/bin/env bash
2 :  '
3 Questo è un semplice script in cui
4 è inserita una sequenza di comandi
5 '
6
7 ...
```

Esempio 22.9: *Script con commento multiline*

22.5.3 Variabili

In *Bash* le variabili non sono tipizzate, possono contenere numeri, caratteri o stringhe. I nomi delle variabili sono *case sensitive* e anche se l'interprete non è restrittivo, la convenzione prevede che i nomi delle variabili:

- dovrebbero iniziare con una lettera o con il carattere _[7];
- possono contenere lettere, cifre e il carattere _;
- non dovrebbero contenere spazi o caratteri speciali;
- dovrebbero essere descrittivi per riflettere lo scopo della variabile;
- non dovrebbero utilizzare keywords (quali `if`, `then`, `else`, `fi`, `while`, ...

Di seguito un esempio:

```
1 #!/usr/bin/env bash
2
3 a=5
4 b=$a
5 echo "b=$b"
```

Esempio 22.10: *Uso di variabili*

in cui si evidenziache il formalismo non prevede ci siano spazi né prima né dopo il segno di uguaglianza.
Per evitare problemi di interpretazione delle variabili (in particolare nelle stringhe) si può utilizzare il metodo di espansione tramite *parentesi graffe*, sostituendo a *$variabile* l'espressione *${variabile}*. In molte situazioni non sarà necessario ma, per non incorrere in problemi di interpretazione, si utilizzerà diffusamente nel seguito.
L'esempio precedente può quindi essere riscritto:

```
1 #!/usr/bin/env bash
2
3 a=5
4 b=$a
5 echo "b=${b}"
```

Esempio 22.11: *Uso di variabili tramite espansione*

e l'output generato sarà equivalente.

[7] *underscore*

Variabili speciali

Oltre le variabili che l'utente creerà, possono essere utilizzate in lettura le seguenti variabili predefinite denominate *parametri speciali*:

`$0`
: è memorizzato il nome dello script eseguito;

da `$1` a `$9`
: sono memorizzati i primi 9 argomenti passati da linea di comando[8];

`$#`
: il numero dei parametri passati allo script[9];

`$@`
: è l'array contenente tutti i parametri mantenendo l'ordine di immissione[10];

`$*`
: meno utile della precedente variabile, contiene la stringa in cui sono concatenati tutti i prametri passati allo script;

`$?`
: è mostrato l'*exit status* restuito dall'ultimo comando[11];

`$$`
: il PID[12] della shell corrente;

`$!`
: il PID dell'ultimo comando in background;

`$-`
: le opzioni correnti impostate per la shell.

Ad esempio:

```
1 #!/usr/bin/env bash
2
3 echo "Il nome dello script è:  $0"
4 echo "Le opzioni correnti sono:  $-"
```

Esempio 22.12: *Nome dello script e opzioni correntemente impostate*

e eseguendo lo script precedente `show_name_and_options.sh`:

```
1 $ ./show_name_and_options.sh
2 Il nome dello script è:  ./show_name_and_options.sh
3 Le opzioni correnti sono: hB
```

Esempio 22.13: *Esecuzione dello script 22.12*

in cui la prima linea riporta in pathname dello script eseguito, la seconda le opzioni impostate per la shell corrente. Quest'ultima linea sarà esaminata nella prossima sezione.

[8] v. 22.5.6 a pag. 327 e relativi esempi
[9] v. ***Esempio 22.40***
[10] v. ***Esempio 22.61***
[11] già utilizzato in. 13.4 a pag. 156, sarà approfondito in **??** a **??**
[12] *process id*, v. 16.5 a pag. 198

22.5.4 Opzioni

Per consultare le opzioni[13] è istruttivo confrontare l'***Esempio 22.12*** con l'output mostrato dalla propria shell:

```
1 $ echo $-
2 himBHs
```

Esempio 22.14: *Valutazione interattiva di* `$-`

Per attivare un nuovo flag si potrà digitare quanto segue:

```
1 $ bash -x
2 ...
3 $ echo $-
4 ...
5 himxBHs
```

Esempio 22.15: *Attivazione di un flag*

in cui, confrontando l'ultima stringa riportata negli ultimi due esempi, si noterà nella seconda l'aggiunta del flag `-x`.
I flag più comuni e qui utilizzati sono:

`h`
: per *hashall*, per tener traccia della locazione di ciascun comando[14] nella `PATH`;

`i`
: per *interactive* poiché la shell corrente è interattiva;

`m`
: flag per abilitare il *job control*[15];

`B`
: per *braceexpand*, abilita la *brace expansion*[16];

`H`
: per *histexpand*, abilita l'espansione dall'history utilizzando il carattere `!`[17];

`s`
: per *source command from stdin*, il flag abilita la lettura da `stdin` qualora non ci siano argomenti successivi alle opzioni immesse;

`x`
: per *xtrace*, il flag attiverà la modalità estremamente verbosa utile per il debug[18].

Per verificare eventuali opzioni impostate dalla shell corrente, v. l'***Esempio 22.37***.

[13] v. sezione `OPTIONS` in `man bash`, più esaustiva sarà la sezione `SHELL BUILTIN COMMANDS` nella trattazione del comando `set` nel medesimo manuale
[14] v. 11.3 a pag. 126
[15] v. 14.6 a pag. 174
[16] v. 4.7.1 a pag. 44, iù esaustiva sarà la sezione`Brace Expansion` in `man bash`
[17] v. 13.4 a pag. 159
[18] v. 22.8 a pag. 341

22.5.5 Lettura

La *keyword* per l'assegnazione di un valore immesso dall'utente o letto da file è `read`.

Input da tastiera

Di seguito una immissione da tastiera e memorizzazione in una variabile:

```
1 #!/usr/bin/env bash
2
3 echo -ne "Inserisci il tuo nome:\t"
4 read name
5 echo "Il tuo nome è:  $name"
```

Esempio 22.16: *Immissione da tastiera*

Lettura da file

Di seguito lettura di ogni linea dal file `read.txt` e scrittura su terminale.

```
1 #!/usr/bin/env bash
2
3 while read line
4 do
5     echo $line
6 done < read.txt
```

Esempio 22.17: *Lettura da file*

22.5.6 Argomenti da linea di comando

Gli argomenti passati allo script (o ad una funzione) sono:

`$1` per il primo argomento;

`$2` per il secondo argomento;

...

`$n` per l'ennesimo argomento.

con `n` fino a *9*. Ad esempio:

```
1 #!/usr/bin/env bash
2
3 echo "Hello $1!"
```

Esempio 22.18: *Mostra gli argomenti*

e eseguendo lo script precedente `show_argument.sh`:

```
1 $ ./show_argument.sh ms
2 Hello ms!
3 $ ./show_argument.sh
4 Hello !
```

Esempio 22.19: *Esecuzione dello script 22.18*

in cui la prima esecuzione ha riportato l'argomento, la seconda non ha riportato ovviamente nessun argomento, ma si sarà notato che non sono stati restituiti errori.

Il numero di argomenti si ottiene consultando la variabile `$#` che sarà utilizzata successivamente nell'***Esempio 22.5.6***.

22.5.7 Gestione stringhe

Di seguito operazioni generiche su stringhe.

Concatenazione

Esempio di concatenazione di stringhe:

```
1 #!/usr/bin/bash
2
3 str1="Hello"
4 str2="world!"
5 stringa="${str1} ${str2}"
6 stringa2=$str1' '$str2
7 echo "str1 $str1 str2 $str2"
8 echo "Concatenazione1 -${stringa}-"
9 echo "Concatenazione2 -${stringa2}-"
```

Esempio 22.20: *Script per concatenazione di stringhe*

la cui esecuzione mostrerà in output:

```
1 str1 Hello str2 world!
2 Concatenazione1 -Hello world!-
3 Concatenazione2 -Hello world!-
```

Esempio 22.21: *Concatenazione di stringhe*

Lunghezza

Per ottenere la lunghezza è sufficiente inserire il carattere `#` nell'espansione della variabile prima del nome, come di seguito:

```
1 #!/usr/bin/bash
2
3 stringa="Hello world!"
4 echo "Lunghezza della stringa -$stringa- è ${#stringa}"
```

Esempio 22.22: *Script per ottenere lunghezza di stringhe*

la cui esecuzione mostrerà in output:

```
1 Lunghezza della stringa -Hello world!- è 12
```

Esempio 22.23: *Esecuzione di 22.22*

Replacement

Per il *replacement* andrà utilizzata l'espressione `$<variable_name>/<string_to_replace>/<new_string>`, e per il replacement ripetuto per ogni occorrenza `$<variable_name>//<string_to_replace>/<new_string>`, come mostrato di seguito:

```
1 #!/usr/bin/bash
2
3 stringa="Hello world!"
4 echo "stringa prima del replacement '$stringa'"
5 echo "stringa dopo il replacement '$stringa/world/ms'"
6 echo "String dopo il replacement '$stringa//l/i'"
```

Esempio 22.24: *Replacement di stringhe*

la cui esecuzione mostrerà in output:

```
1 stringa prima del replacement 'Hello world!'
2 stringa dopo il replacement 'Hello ms!'
3 String dopo il replacement 'Heiio worid!'
```

Esempio 22.25: *Esecuzione di 22.24*

Substring

Per le operazioni di *substring* andrà utilizzata l'espressione `${stringa:$position:$length}'"` con:

`$position` parametro per definire la posizione nella `stringa`, può essere negativo (default 0);

`$length` parametro per definire la lunghezza (parametro non obbligatorio);

ed entrambi i parametri da inserire dopo un carattere `:` dentro l'espansione della variabile. Ad esempio:

```
1 #!/usr/bin/bash
2
3 stringa="Hello world!"
4
5 position=4
6 echo "Estrazione caratteri dalla stringa '$stringa' dalla posizione
      $position '${stringa:$position}'"
7 length=5
8 echo "Estrazione di $length caratteri da '$stringa' dalla posizione
      $position '${stringa:$position:$length}'"
9 echo "Estrazione di $length caratteri dalla fine '$stringa' alla
      posizione -$position '${stringa:(-$position):$length}'"
```

Esempio 22.26: *Esempi di substring*

la cui esecuzione mostrerà in output:

```
1 Estrazione caratteri dalla stringa 'Hello world!'  dalla posizione 4 'o
     world!'
2 Estrazione di 5 caratteri da 'Hello world!'  dalla posizione 4 'o wor'
3 Estrazione di 5 caratteri dalla fine 'Hello world!'  alla posizione -4
     'rld!'
```

Esempio 22.27: *Esecuzione di 22.26*

22.5.8 Valutazione aritmetica

Di default le stringhe sono valutate come stringhe, per attivare la valutazione di un'espressione aritmetica andrà utilizzata l'espressione:

```
(( espressione ))
```

Esempio 22.28: *Valutazione aritmetica*

come nello script `numeri.bash` di seguito:

```
#!/usr/bin/bash

area=5*4
echo $area+5

((area=5*5))
echo $area
```

Esempio 22.29: *Esempio di valutazione aritmetica*

che mostrerà:

```
$ ./numeri.bash
5*4+5
25
```

Esempio 22.30: *Esecuzione aritmetica*

22.5.9 Espressioni

Nella ***Tabella 22.1*** sono elencati i principali operatori utilizzabili per formare le espressioni che riporteranno un valore booleano.

22.5.10 Espressioni aritmetiche

Nella ***Tabella 22.2*** sono elencati i principali operatori aritmetici utilizzabili per formare le espressioni che riporteranno un valore booleano. Un banale esempio:

```
#!/usr/bin/bash

i=4
echo $((i++))
echo $((++i))

((divisione=5/2))
echo $divisione

((resto=5%2))
echo $resto

((potenza=5**2))
echo $potenza
```

Esempio 22.31: *Alcune operazioni aritmetiche*

la cui esecuzione resituirà:

ESPRESSIONE	DESCRIZIONE
`-d file`	*True* se il file esiste ed è una directory
`-e file`	*True* se il file esiste
`-f file`	*True* se il file esiste ed è un *regular file*
`-r file`	*True* se il file esiste ed è leggibile
`-s file`	*True* se il file esiste ed ha dimensione maggiore di 0
`-w file`	*True* se il file esiste ed è scrivibile
`-x file`	*True* se il file esiste ed è eseguibile
`file1 -ef file2`	*True* se i file si riferiscono allo stesso device e numero di inode
`file1 -nt file2`	*True* se il `file1` è più recente (*modification date*) del `file2`, oppure se `file1` esiste e `file2` no
`file1 -ot file2`	*True* se il `file1` è meno recente (*modification date*) del `file2`, oppure se `file2` esiste e `file2` no
`-v varname`	*True* se `varname` è stata valorizzata
`-z varname`	*True* se la lunghezza di `varname` è zero
`-n varname`	*True* se la lunghezza di `varname` non è zero
`string`	*True* se la lunghezza di `string` non è zero
`string1 == string2`	*True* se le stringhe sono uguali
`string1 != string2`	*True* se le stringhe non sono uguali
`string1 < string2`	*True* se `string1` è lessicograficamente precedente a `string2`
`string1 > string2`	*True* se `string1` è lessicograficamente successiva a `string2`
`arg1 OP arg2`	`OP` è un generico operatore binario aritmetico da sostituire con: `-eq` *True* se argomenti sono uguali `-ne` *True* se argomenti non sono uguali `-lt` *True* se `arg1` $<$ `arg2` `-le` *True* se `arg1` $\leq$ `arg2` `-gt` *True* se `arg1` $>$ `arg2` `-ge` *True* se `arg1` $\geq$ `arg2`

Tabella 22.1: *I principali operatori unari e binari utilizzabili per ottenere un booleano*

```
1 4
2 6
3 2
4 1
5 25
```

OPERATORE	DESCRIZIONE
`i++ i--`	rispettivamente *postincremento* e *postdecremento*
`++i --i`	rispettivamente *preincremento* e *predecremento*
`+ -`	unario rispettivamente per *positivo* e *negativo*
`! ~`	negazione *logica* e *bitwise*
`**`	elevazione a potenza
`* / %`	moltiplicazione, divisione e resto
`+ -`	binario rispettivamente per *addizione* e *sottrazione*
`<= >= < >`	comparatori
`== !=`	eguaglianza e disuguaglianza
`&`	bitwise *AND*
`^`	bitwise *XOR*
`|`	bitwise *OR*
`&&`	*AND* logico
`||`	*OR* logico
`= *= /= %= += -= <<= >>= &= ^= |=`	assegnazioni

Tabella 22.2: *I principali operatori unari e binari aritmetici utilizzabili per ottenere un booleano*

Esempio 22.32: *Esecuzione di operazioni aritmetiche*

22.5.11 Strutture di controllo

Nelle sequenti sezioni sono illustrati i costrutti principali. Da notare che il carattere `;` nella sintassi potrà essere sostituito con *newline*.

Rami alternativi

Solitamente le strutture di controllo *alternative* consentono di specificare che una data istruzione o blocco di istruzioni dovranno essere eseguiti nel caso sia vera l'espressione utilizzata.
In questo contesto, il percorso tra i rami sarà alternativo in base all'*exit status* riportato dall'espressione, in particolare il ramo sarà percorso se l'*exit status*[19] sarà 0.
La sintatssi del costrutto generico è la seguente:

```
1 if expression1; then
2     statement
3 elif expression2; then
4     statement
5 else
6     ramo di default
7 fi
```

[19] si ricorda che l'*exit status* di un comando potrà essere visualizzato consultando il contenuto della variabile `$?`. Per approfondimenti si v. **??** a **??**

Esempio 22.33: *Costrutto generico* if-then-eif-else

che potrebbe essere scritta anche nel seguente modo:

```
1 if test expression1; then
2     statement
3 elif test expression2; then
4     statement
5 else
6     ramo di default
7 fi
```

Esempio 22.34: *Costrutto generico* if-then-eif-else *con* `test`

oppure nel seguente modo:

```
1 if [ expression1 ]; then
2     statement
3 elif [ expression2 ]; then
4     statement
5 else
6     ramo di default
7 fi
```

Esempio 22.35: *Costrutto generico* if-then-eif-else *con doppia parentesi quadra*

oppure nel seguente modo:

```
1 if [[ expression1 ]]; then
2     statement
3 elif [[ expression2 ]]; then
4     statement
5 else
6     ramo di default
7 fi
```

Esempio 22.36: *Costrutto generico* if-then-eif-else *con doppia parentesi quadra*

e ovviamente, anche se non coerente, i rami potrebbero essere mixati quindi, ad esempio, il primo ramo potrebbe essere scritto con parantesi singole e il secondo con doppia parentesi quadra. Essenzialmente[20]:

- il ramo con direttamente l'`expression`, quindi senza l'utilizzo di partentesi quadre, sarà direttamente costituito da un comando che restituirà una *exit status*;
- il ramo con l'utilizzo di `test`[21] o equivalentemente singole parentesi quadre[22] `[ ...]` permette di comporre la condizione utilizzando gli operatori logici `!` per la negazione, `-a` per l'*and* e `-o` per l'*or*;

[20] per una miglior comprensione si consulti la sezione `CONDITIONAL EXPRESSIONS` in `man bash`

[21] v. `man test`

[22] v. `man [`

- il ramo con doppie parentesi quadre `[[ ...]]` costituisce una evoluzione nell'espressività e permette di comporre tra loro più condizioni utilizzando gli stessi operatori logici usati nel linguaggio `C`, quindi `!` per la negazione, `&&` per l'*and* e `||` per l'*or*.

In tutti i casi precedenti sarà possibile utilizzare le parentesi tonde per raggruppare subespressioni, quindi modificare la precedenza nell'espressione globale.

Lo script di esempio `if_then_else_interattivo.sh` per valutare se la nostra shell è interattiva:

```
1 if [[ $- == *i* ]]
2 then
3  echo 'La bash è interattiva'
4 else
5  echo 'La bash non è interattiva'
6 fi
```

Esempio 22.37: *Esempio con costrutto* if-then-else *per verifica dell'interattività della shell*

le cui esecuzioni di seguito:

```
1 $ bash ./if_the_else_interattivo.sh
2 La bash non è interattiva
3 $ bash -i ./if_the_else_interattivo.sh
4 La bash è interattiva
```

Esempio 22.38: *Verifica se la modalità è interattiva*

Uno script con più branch alternativi:

```
1 #!/usr/bin/bash
2
3 a=2
4 if [[ $a == 1 ]]; then
5     echo "ramo if"
6 elif [[ $a == 2 && 3 == 3 ]]; then
7     echo "ramo elif"
8 else
9     echo "ramo else"
10 fi
```

Esempio 22.39: *Esempio con costrutto* if-then-eif-else

che in output mostrerà `ramo elif`.

Ricordando che il numero di argomenti si ottiene consultando la variabile `$#`, si può voler eseguire un ramo solo se il numero degli argomenti è corretto:

```
1 #!/usr/bin/env bash
2
3 if [[ $# -eq 1 ]]
4 then
5     echo "Hello $1!"
6 else
7     echo "Solo un nome!"
8 fi
9
```

```
10 echo "Numero degli argomenti: $#"
```

Esempio 22.40: *Controlla il numero di argomenti*

e, eseguendo lo script precedente `one_argument.sh`:

```
1 $ ./one_argument.sh ms
2 Hello ms!
3 Numero degli argomenti: 1
4 $ ./one_argument.sh
5 Solo un nome!
6 Numero degli argomenti: 0
```

Esempio 22.41: *Esecuzione dello script 22.40*

Un esempio senza uso di parentesi (poiché l'*exit status* è fornito direttamente dai comandi) è il seguente:

```
1 #!/usr/bin/env bash
2
3 if test ! -f /usr/share/dict/italian; then
4     echo "the file does not exist"
5 elif grep -q ciao /usr/share/dict/italian; then
6     echo "the word is in the dictionary"
7 else
8     echo "the word is not in the dictionary"
9 fi
```

Esempio 22.42: *Controlla l'*exit status

in cui, tenendo conto che il file `/usr/share/dict/italian` è il percorso del dizionario presente nelle distribuzioni con installata la lingua italiana, eseguendo lo script, in tale scenario si otterrà l'output del ramo `elif`.

Cicli

Le strutture cicliche sono `while`, `until` e `for`. Di seguito le strutture sintattiche più popolari.

`while`

Sintassi generale per il ciclo `while` è la seguente:

```
1 while test
2 do
3     ...
4 done
```

Esempio 22.43: *Costrutto generico* while

con `...` una sequenza di comandi eseguiti se il `test` è `True`[23].
Un esempio in `while.bash`:

```
1 #!/usr/bin/bash
2
3 x=0
```

[23] più correttamente, un *exit status 0*

```
4 z=5
5 while [[ $x -le $z ]]
6 do
7   echo "$x/$z"
8   x=$(( $x + 1 ))
9 done
```

Esempio 22.44: *Esempio con costrutto* while

la cui esecuzione mostrerà in output:

```
1 $ ./while.bash
2 0/5
3 1/5
4 2/5
5 3/5
6 4/5
7 5/5
```

Esempio 22.45: *Esecuzione del* while

`until`

Sintassi generale per il ciclo `until` è la seguente:

```
1 until test; do
2     ...
3 done
```

Esempio 22.46: *Costrutto generico* until

con `...` una sequenza di comandi eseguiti se il `test` è `False`[24].
Un esempio in `until.bash`:

```
1 #!/usr/bin/bash
2
3 x=0
4 z=5
5 until [[ $x -gt $z ]]; do
6   echo "$x/$z"
7   x=$(( $x + 1 ))
8 done
```

Esempio 22.47: *Esempio con costrutto* until

la cui esecuzione mostrerà in output:

```
1 $ ./until.bash
2 0/5
3 1/5
4 2/5
5 3/5
6 4/5
7 5/5
```

Esempio 22.48: *Esecuzione del* until

[24] più correttamente, un *exit status non 0*

for

Sintassi generale moderna per il ciclo `for` è la seguente:

```
1 for (( init; test; step )); do
2     ...
3 done
```

Esempio 22.49: *Costrutto generico* for

in cui:

`init` sarà la condizione iniziale;

`test` sarà il test effettuato ad ogni ciclo;

`step` sarà l'incremento per ogni ciclo;

`...` una sequenza di comandi che saranno eseguiti per ciascun ciclo.

Un esempio in `for.bash`:

```
1 #!/usr/bin/bash
2
3 x=0
4 z=5
5 for ((i=$x; i<=$z; i++)); do
6   echo "$i/$z"
7 done
```

Esempio 22.50: *Esempio con costrutto* for *moderno*

la cui esecuzione mostrerà l'output:

```
1 $ ./for.bash
2 0/5
3 1/5
4 2/5
5 3/5
6 4/5
7 5/5
```

Esempio 22.51: *Esecuzione del* for

Questo costrutto è usabile anche per il ciclo infinito come il seguente:

```
1 #!/usr/bin/bash
2
3 i=0
4 for (( ; ; ))
5 do
6    ((i++))
7    echo "$i [ hit CTRL+C to stop]"
8 done
```

Esempio 22.52: *Esecuzione del* for *infinito*

for/2

Una sintassi con sequenza per il ciclo `for` è la seguente:

```
1 for variabile in lista; do
2     ...
3 done
```

Esempio 22.53: *Costrutto generico* for

in cui:

`lista` può essere una:

sequenza: ad esempio `1 2 3 4 5`;

range: ad esempio `{1..5}` oppure `{0..9..2}` (l'ultimo numero indica lo step di incremento);

con `...` una sequenza di comandi che saranno eseguiti per ciascun membro della sequenza (estremi inclusi).

Un esempio:

```
1 #!/usr/bin/bash
2
3 for i in {0..9..2}; do
4   echo $i
5 done
```

Esempio 22.54: *Esempio di* for *con sequenza*

oppure una sequenza derivante da un altro comando:

```
1 #!/usr/bin/bash
2
3 for i in $(ls ?.txt); do
4   echo "comprimo $i..."
5   gzip $i
6 done
7 echo "done"
```

Esempio 22.55: *Esempio di* for *con sequenza generata da un altro comando*

for/3†

Un ultimo esempio in `for_seq.bash` ancora popolare ma con comando `seq` obsoleto:

```
1 #!/bin/bash
2
3 x=0
4 z=5
5 for i in $(seq $x $z); do
6   echo "$i/$z"
7 done
```

Esempio 22.56: *Esempio con costrutto* for *utilizzando* `seq`†

la cui esecuzione mostrerà in output:

```
$ ./for_seq.bash
0/5
1/5
2/5
3/5
4/5
5/5
```

Esempio 22.57: *Esecuzione del* until

22.6 Array

Un array è una collezione di entità (stringhe, numeri, ...) immagazzinate in un contenitore, in cui ogni entità si identifica con la posizione all'interno dell'array.
Un esempio:

```
#!/bin/bash

somePower=(1 2 4 8 16 32 64)

echo ${somePower[0]}
echo ${somePower[1]}
echo ${somePower[${#somePower[@]}-1]}

somePower+=( 128 256 )
echo ${somePower[${#somePower[@]}-1]}
```

Esempio 22.58: *Stampa di array*

in cui:

`somePower=(1 2 4 8 16 32 64)`
: si inizializza l'array;

`echo $somePower[n]`
: stampa i valori, rispettivamente con indice 0, indice 1, e l'ultimo (con indice uguale al numero dei componenti l'array meno 1);

`somePower+=( 128 256 )`
: aggiunge due elementi all'array;

`${somePower[${#somePower[@]}-1]}`
: mostra nuovamente l'ultimo.

per cui l'esecuzione mostrerà in output:

```
1
2
64
256
```

Esempio 22.59: *Stampa di array*

Un esempio con file e directory:

```
1 #!/bin/bash
2
3 allFiles=$(ls)
4 for f in ${allFiles[@]}; do
5   echo $f
6 done
```

Esempio 22.60: *Stampa di array*

che eseguito mostrerà il nome dei file nella directory.
Il seguente esempio utilizza l'array che contiene tutti i parametri passati allo script `array2.sh`:

```
1 #!/bin/bash
2
3 for t in $@; do
4   echo $t
5 done
```

Esempio 22.61: *Array con i parametri passati*

che eseguito mostrerà tali parametri.

```
1 $ ./array2.sh 1 2 e 4 f
2 1
3 2
4 e
5 4
6 f
```

Esempio 22.62: *Esecuzione di 22.60*

22.7 Funzioni

Le funzioni hanno un costrutto molto semplice includendo anche le variabili locali

```
1 ...
2 nomefunzione () {
3    # definisco variabili locali
4    local var1; local var2; ...
5    # argomenti passati alla funzione
6    var1=$1
7    var2=$2
8    ...
9    # body
10   ...
11 }
```

Esempio 22.63: *Costrutto generico di una funzione*

in cui:

`nomefunzione` è conpoca sorpresa il nome sceltoper la funzione;

`var1` e `var2` sono le variabili locali della funzione;

`$1` e `$2` sono le variabili passate alla funzione.

Nell'esempio seguente `somma.sh` si crea una funzione che esegue la somma di tutti i parametri passati:

```
1  #!/bin/bash
2
3  somma () {
4      a=0
5      for element in "$@"
6      do
7          a=$(($a+$element));
8      done
9    echo $a;
10 }
11
12 somma $@
```

Esempio 22.64: *Funzione per la somma dei parametri*

che eseguito restituirà la somma.

```
1 $ ./somma.sh 12 2 4 5
2 23
```

Esempio 22.65: *Esecuzione di* **??**

22.8 Debugging

Le fasi di *debugging* è essenziale per gli script che mostrano comportamenti inaspettati. Probabilmente la più utile tecnica offerta per tale attività è l'attivazione dell'opzione `-x`, la quale mostrerà ogni comando eseguito al terminale immediatamente dopo il simbolo + e uno spazio.
Riprendendo l'***Esempio 22.43*** modificandolo attivando l'opzione di debug:

```
1  #!/usr/bin/bash
2
3  set -x
4
5  x=0
6  z=5
7  while [[ $x -le $z ]]
8  do
9    echo "$x/$z"
10   x=$(( $x + 1 ))
11 done
```

Esempio 22.66: *Esempio con costrutto* while *in modalità debug*

la cui esecuzione mostrerà in output:

```
1 $ ./while.bash
2 0/5
3 1/5
4 2/5
5 3/5
6 4/5
7 5/5
```

Esempio 22.67: *Esecuzione del* while

da confrontare con quanto riportato in ***Esempio 22.67***.

22.9 Approfondimenti

22.9.1 Shebang

Poiché nei linguaggi di scripting il carattere # indica comunemente l'inizio di un commento, l'interprete ignorerà il contenuto della riga che contiene nell'intestazione lo shabang. La linea ha infatti solo la funzione di indicare quale interprete è necessario, concettualmente non è parte del contenuto dello script.

Lo shabang è sostanzialmente una versione leggibile di un *magic number* nel file eseguibile. La stringa è la sequenza di byte *0x23 0x21* all'inizio del file, che corrisponde alla rappresentazione secondo la codifica ASCII dei caratteri `#!`.
I file eseguibili che non richiedono interpreti hanno altri *magic number*.

22.9.2 Portabilità

Come si è accennato in 22.4.1, non c'è assoluto consenso tra le varie distribuzioni nel collocare l'interprete e, per questo motivo, nel passaggio tra sistemi, è spesso necessario modificare l'intestazione dello script proprio per permettere l'usabilità dello stesso tra sistemi diversi.
Per ottenere una maggior portabilità si può ricorrere al comando `env` con l'intestazione alternativa di seguito:

```
1 #!/usr/bin/env interprete
```

Esempio 22.68: *Intestazione generica tramite* `env`

con `interprete` da sostituire con l'interprete desiderato, quindi `bash` per il *bash*, `perl` per il *Perl*, `python` per il *Python*, `php` per il *PHP*, ...
L'utilizzo del comando `env` come appena mostrato conduce ad un secondo problema di portabilità legato alla gestione di eventuali parametri specificati, come ad esempio nel seguente script `test.pl`:

```
1 #!/usr/bin/env perl -w -T
```

Esempio 22.69: *Intestazione errata tramite* `env` *con parametri aggiuntivi*

poiché, quando lo script sarà eseguito, si otterrà come output
`/usr/bin/env: "perl -w -T": File o directory non esistente`
in quanto `perl` e `-w -T` hanno formato un unico parametro.
Per ovviare si può ricorrere all'uso dell'opzione `-S`[25] nell'intestazione come mostrato:

```
1 #!/usr/bin/env -S perl -w -T
```

Esempio 22.70: *Intestazione corretta tramite* `env` *con parametri aggiuntivi*

e, quando lo script sarà eseguito, l'espansione sarà corretta in `perl -w -T test.pl`.

[25] anche estesa in `--split-string`

22.9.3 Fork bomb

Il *fork bomb* è un attacco di tipo *denial of service* contro un computer tramite la rapida ripetizione della funzione `fork`, saturando lo spazio disponibile della lista dei processi mantenuta dal SO. In particolare, riempiendo la tabella dei processi, non sarà possibile avviarne ulteriori finché non terminerà uno in esecuzione. Essendo il processo *fork bomb* attivo, quest'ultimo starà creando istanze in attesa di slot della tabella dei processi, per cui sarà improbabile che si attiverà un nuovo processo utile.
Un programma *fork bomb* può essere scritto con diversi linguaggi di programmazione, in Bash il tradizionale sorgente è di seguito:

```
1 :(){ :|:& };:
```

Esempio 22.71: *Script tradizionale del* fork bomb *in Bash. Nota: non eseguire!!*

che per chiarezza può essere ritrascritto come segue:

```
1 fork() {
2  fork | fork &
3 }
4 fork
```

Esempio 22.72: *Trascrizione del* fork bomb *in Bash. Nota: non eseguire!!*

In quest'ultimo esempio appare più evidente che nello script è definita la funzione `fork()` che, nella seconda linea, richiama se stessa con `fork` ed invia in piping tramite `|` il risultato a se stessa, inviando il job in background tramite `&`.

Nell'***Esempio 22.71***, la funzione è cripticamente definita con `:`, utilizzo permesso nella *GNU Bash*, ma ad esempio non valido nella *Bourne shell* come definito nello standard POSIX.
Una versione Bash più moderna è mostrata in ***Figura 22.1***, in cui rispetto all'***Esempio 22.71*** si è sostituita la criptica funzione nominata `:` con una denominata tramite emoji.



Figura 22.1: *Versione moderna della* fork bomb *in* Bash

22.10 Concetti chiave

bitwise: bit a bit

booleano: valore *True* o *False*, anche 1 o 0

command line: interfaccia a riga di comandi

debug: individuazione e correzione di errori di programmazione

interprete: software in grado di eseguire programmi a partire da codice sorgente scritto in un linguaggio di alto livello

magic number: particolari byte del file. L'elenco dei *magic number* e la loro posizione relativa all'inizio del file sono contenuti nel file `/usr/share/misc/magic.mgc`.

portabilità: proprietà di software di essere impiegato su altri sistemi (oltre quello in cui è stato creato) senza subire modifiche

script: codice sorgente di linguaggio ad alto livello che sarà interpretato

shabang: sequenza `#!` seguita dal path assoluto dell'interprete del file. Concettualmente è un commento quindi non inglobata nello script

22.11 Comandi introdotti nel capitolo

`bash` ⇒ interprete *Bash*

`env` ⇒ esegue un software con ambiente modificato

`expr` ⇒ valuta espressioni (per molti linguaggi analogo a `eval`)

`seq` ⇒ genera una sequenza numerica

Parte IV

Intermedio

Capitolo 23

Startup e shutdown

QUESTO capitolo tratta principalmente l'argomento dell'*avvio*, dal *boot loader* allo start dei processi core del sistema. In ultimo ci sarà una breve trattazione della fase opposta, lo *spegnimento* del sistema.

23.1 Panoramica

Banalmente il processo di avvio inizia con la pressione del bottone di accensione e procede per diversi step che vanno dall'inizializzazione hardware fino all'esecuzione di decine o centinaia di software che costituiscono il sistema. Nello specifico la fase iniziale sarà legata all'hardware del sistema, quindi sarà indipendente dal SO, successivamente le fasi potranno essere simili solo da un punto di vista astratto, ma tecnicamente si differenzieranno a seconda del sistema. In un dispositivo hardware con sistema GNU/Linux solitamente si avrà: I principali step sono:

1. inizializzazione della CPU;
2. la stessa CPU esegue del codice denominato *firmware* che contiene le istruzioni iniziali per procedere (inizializza il sistema hardware, esegue test della memoria, ...);
3. al termine delle precedenti fasi, il firmware stesso esegue lo *stage 1* del *boot loader*: secondo la sequenza dei dispositivi[1] di boot definita, cercherà codice che avvii il boot del sistema e, appena trovato, lo eseguirà;
4. lo *stage 1* del boot loader indicherà al sistema di eseguire ulteriori *stage* differentemente a seconda del SO, in ogni caso sarà caricato in memoria il kernel del sistema ed eseguito. Da questo stadio, sarà il kernel ad avere il controllo della macchina;
5. il kernel, tra i suoi task, eseguirà il primo processo che sarà `/sbin/init` denominato semplicemente *init*;
6. il processo *init* leggerà file di configurazione dai quali eseguirà altri software del sistema.

[1] quindi l'ordine può variare ma solitamente comprenderà hard disk, storage removibile, dischi ottici, ...

Il processo *init* tipicamenete eseguirà, tra gli altri, il modulo di login, testuale o grafico tramite l'**XDM**[2].

23.2 Avvio tradizionale

Replicando i passaggi della sezione precedente, di seguito sarà presentata la sequenza di *startup* tradizionale.

BIOS

Acronimo di ***B**asic **I**nput/**O**utput **S**ystem*:

- esegue test iniziale di integrità del sistema denominato **POST**[3];
- solitamente premendo un qualche tasto[4] durante questa fase si potrà accedere a modifiche di configurazione, spesso utilizzata la ridefiniziome della sequenza dei dispositivi di boot;
- cerca un *boot loader* secondo la sequenza dei dispositivi di storage definita, e appena trovato sarà caricato in memoria e eseguito.

In ***Figura 23.1*** la tipica schermata di configurazione del BIOS su PC.

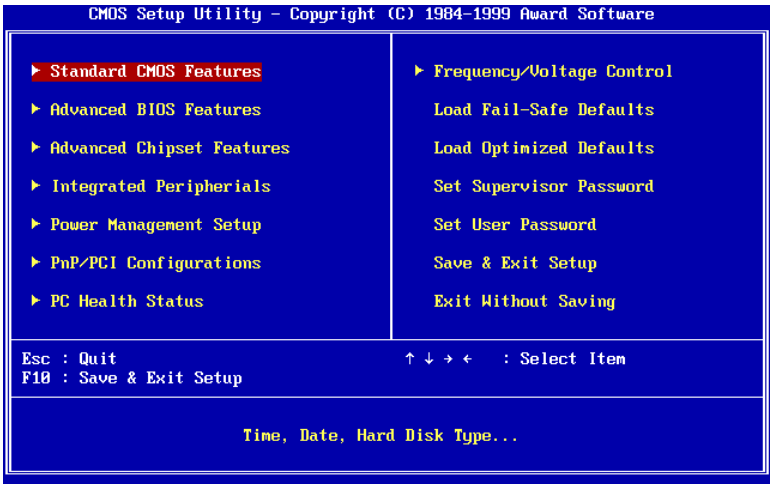


***Figura 23.1:** Tipica schermata di configurazione del* BIOS *su PC*

MBR

Acronimo di ***M**aster **B**oot **R**ecord*, sarà la sequenza che prenderà il controllo del sistema nel momento della sua esecuzione. Essenzialmente:

[2] ***X D**isplay **M**anager*

[3] ***p**ower-**o**n **s**elf-**t**est*, frequente fase di autodiagnosi in molti dispositivi per testare il corretto funzionamento dell'hardware (non solo scheda madre, ma anche periferiche comuni) prima delle successive fasi del processo di *bootstrap*

[4] tipicamente F2 o F10 o F12, varia tra sistemi

- è localizzato nel primo settore del disco[5] dal quale sarà effettuato il boot;
- come già scritto nel precedente punto, è composto dai primi 512 byte suddivisi come segue:
  - 446 byte è l'area contenente il **MBP**[6], la *bootstrap code area*;
  - 64 byte per la *partition table*, cioè lo schema di partizionamento del disco;
  - 2 byte per il *boot signature* o *MBR validation check*[7].

Nelle installazioni tradizionali, la *bootstrap code area* è essenzialmente un programma scritto in *Assembly*, corrisponde allo *stage 1* del *bootloader GRUB*, al termine si passerà agli *stage* successivi del *bootloader*.
Per quanto riguarda la *partition table*, i punti essenziali e limitanti sono di seguito:

- contiene informazioni ciascuna di 16 bytes sulle 4 partizioni che possono essere definite;
- la limitazione del precedente punto può essere superata non utilizzando una *partizione primaria* ma definendola *estesa*[8].
- dovendo definire per ciascuna partizione il numero di settori, il valore massimo esprimibile è $2.2TB$[9] di spazio.

GRUB

Lo step dell'esecuzione del *boot loader* è l'iniziale diramazione del caricamento del SO, in particolare in un sistema GNU/Linux sono utilizzabili diversi[10] *boot loader*.
Il bootloader più diffuso per i sistemi GNU/Linux è **GRUB**, contrazione di ***G**rand **U**nified **B**ootloader*. Le sue caratteristiche principali sono:

- se si hanno più immagini installate nel proprio sistema, fornirà la scelta di quale avviare;
- di default mostrerà uno screen di scelta, attenderà per qualche secondo e, se non si sarà premuto nulla, sarà eseguito quello impostato di default nel file di configurazione (v. ***Figura 23.2***);
- a differenza del predecessore[11], ha la conoscenza necessaria di taluni filesystem, quindi possono essere memorizzati nella directory `/boot/grub`.

Le versioni di GRUB maggiormente utilizzate sono:

---

[5] tipicamente `/dev/sda` o `/dev/vda`

[6] ***M**aster **B**oot **P**rogram*

[7] il termine *check* potrebbe indurre a pensare ad un *checksum*, ma in realtà la signature è la costante trascritta con `0x55AA`

[8] sarà definito il record **EBR** (acronimo di ***E**xtended **B**oot **R**ecord*) nel primo settore di ogni partizione logica. La struttura del record è identica all'MBR, ma lo spazio riservato alla *code area* è inutilizzato. L'*EBR* contiene il settore di inizio della partizione logica e il puntatore alla successiva partizione logica

[9] v. **??**

[10] v. **??**

[11] v. **??**

`version 0`,
: conosciuta anche come *GRUB Legacy*, disponibile fino alla versione 0.97 e non più sviluppata;

`version 2`,
: quella maggiormente diffusa e attualmente in sviluppo.

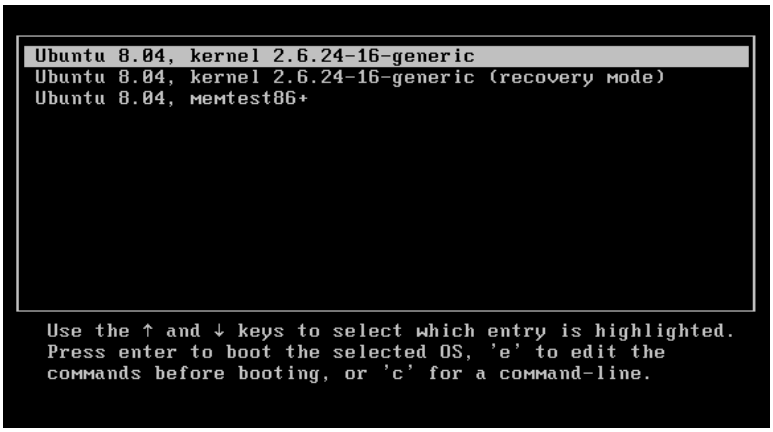


Figura 23.2: *Uno screenshot datato di* GRUB Legacy*, essenzialmente rimasto inalterato nella configurazione di default per* GRUB 2

Non entrando nei dettagli, le due versioni sono abbastanza simili, in particolare nell'interazione con l'utilizzatore. Ad esempio:

- per le selezioni tra le voci di menù si utilizzano i tasti ↑ e ↓;
- posizionati sulla voce desiderata, premendo ↵ si avvierà il boot indicato.

La configurazione è però abbastanza differente, di seguito saranno esaminate le caratteristiche principali.

GRUB Legacy Il file principale per la configurazione è `/boot/grub/menu.lst`[12], e la comprensione e personalizzazione è più immediata.
Di seguito un esempio:

```
 1 default=0
 2 #hiddenmenu
 3 timeout=10
 4 splashimage=/grub/bootimage.xpm.gz
 5
 6 title  Ubuntu
 7 root   (hd0,1)
 8 kernel /boot/vmlinuz-2.6.20-15-generic root=/dev/sda2 ro locale=
       it_IT splash
 9 initrd /boot/initrd.img-2.6.20-15-generic
10
11 title  Debian
12 root   (hd0,2)
13 kernel /boot/bzImage-2.6.36-experimental ro root=/dev/sda3
```

[12] per alcune distribuzioni è `/boot/grub/grub.conf`

```
14
15 title  FreeBSD
16 root   (hd0,3,a)
17 kernel /boot/loader
18
19 # Winzoz
20 #title Windows
21 #rootnoverify (hd0,0)
22 #makeactive
23 #chainloader +1
```

Esempio 23.1: *Spezzone del file* `/boot/grub/menu.lst`

Nell'esempio:

- le prime sono impostazioni globali:

  `default=0`
  : dopo un lasso di tempo successivamente definito, si avvierà automaticamente il primo[13] SO presente nel file di configurazione;

  `timeout=10`
  : qualora entro il lasso di tempo indicato in secondi non sia premuto nessun tasto, sarà avviato il SO indicato nel precedente punto;

  `#hiddenmenu` la voce è commentata, per cui il menu sarà visibile (di default non è visibile);

  `splashimage=/grub/bootimage.xpm.gz` per utilizzare un'immagine di sfondo durante l'avvio;

- le altre sono blocchi che si applicano per ciascuna voce di menu, in particolare i termini chiave utilizzati sono:

  `title`
  : indica la entry che sarà visualizzata nel menu e, soprattutto, le linee successive contenute nel file di configurazione saranno ricondotte a tale SO fino al prossimo `title`;

  `root`
  : imposta il disco fisso[14] e partizione radice contenente la configurazione di GRUB[15].
  In particolare `hd0` sarà il primo disco fisso rilevato, `hd1` il secondo e così via. La numerazione delle partizioni parte da 0 invece di 1 utilizzato dal kernel Linux, quindi il numero da inserire nel file di configurazione è di una posizione minore rispetto al numero della partizione.

[13] 0 per il primo, 1 per il secondo, ...

[14] *GRUB* non distingue tra tecnologie, quindi *PATA* (Parallel Advanced Technology Attachment), *SATA* (Serial ATA), *SCSI*, ..., saranno tutti `hd`

[15] corrisponde alla directory `/boot`. La partizione di root di GRUB è la partizione contenente i file di configurazione di GRUB. Tali file sono nella directory `/boot/grub/`, quindi la partizione radice di GRUB e la partizione radice di Linux coincideranno a meno che non sia stata utilizzata una partizione differente per `/boot` oppure per `/boot/grub`. In altri termini, in talune configurazioni si utilizza per `/boot` una partizione separata, in questo caso la partizione radice di GRUB sarà proprio la partizione indicata per montare nel sistema la directory `/boot` (cfr. il contenuto del file `/etc/fstab`)

Ad esempio, se la partizione sarà `/dev/sda3`, la stringa da impostare nel file di configurazione sarà `root (hd0,2)`. Il mapping generato da GRUB è localizzato nel file `/boot/grub/device.map`.
Qualora il sistema sia dotato di floppy disk, il primo device floppy sarà `fd0`, l'eventuale secondo sarà `fd1`, senza indicare la partizione poiché tali device non sono partizionati;

`kernel`
indica il path completo dell'immagine del kernel[16] con i relativi argomenti, i principali sono:

`ro`
accede in *readonly*[17] alla partizione contenente la directory `/`;

`root`
indica al kernel qual è la partizione contenente la directory `/` del sistema.

`quiet`
sono disabilitati la maggior parte dei messaggi di log[18];

`locale`
indica l'impostazione `locale` da utilizzare;

`splash`
in realtà non è un parametro ufficiale del kernel, ma nella maggior parte delle distribuzioni sarà mostrato a video l'immagine impostata durante il caricamento del sistema, fino allo step di *login*.

`initrd`
indica il path completo dell'immagine `initrd`[19] da avviare;

- le ultime sono commentate ma frequentemente attive in caso di *dual boot*, in particolare decommentate avvierebbero un mediocre ma purtroppo ancora inflazionato SO alternativo. In dettaglio sono state utilizzate:

`rootnoverify`
per accedere in sola lettura[21] alla partizione contenente la directory radice. L'opzione si utilizza per segnalare a GRUB stesso di non provare ad accedere ai file della partizione e caricare autonomamente il kernel;

`makeactive`
rende la partizione attiva (necessario per Windows);

[16] spesso si utilizza specificare il path tramite la sintassi GRUB, quindi per la prima immagine poteva essere espresso il percorso equivalentemente con `(hd0,1)/vmlinuz-2.6.20-15-generic`

[17] per motivi di sicurezza è consigliato impostare tale opzione, sarà successivamente rimontata in *read/write* in accordo con il contenuto del file `/etc/fstab`

[18] quindi, in condizioni di avvio ordinario, non saranno mostrati messaggi di debug

[19] è la contrazione di ***Init**ial **R**AM **D**isk*, i cui compiti principali sono:

- root filesystem temporaneo, fino al boot del kernel e al montaggio di quello che sarà poi effettivamente il root filesystem;
- contiene alcuni tool per il processo di boot e i driver necessari per l'accesso all'hardware, compreso le partizioni del disco.

Solitamente si adotta la soluzione del caricamento del `initrd`, ma è possibile evitare tale passaggio compilando autonomamente[20] il kernel

[21] consigliata per motivi di sicurezza

`chainloader`
: carica la prima traccia[22] della partizione la quale conterrà un secondo *boot loader* da avviare, quindi GRUB passerà il controllo a quest'ultimo;

Qualora non si avesse installato *GRUB* e si volesse installarlo, si utlizzerà `grub-install` *`device`*[23].

GRUB 2 L'evoluzione del software ha condotto alla versione 2, molto più flessibile e ovviamente meno leggibile. Processando con gli adeguati strumenti[24] il contenuto della directory `/etc/grub.d` e il file `/etc/default/grub`, si genererà il file di configurazione `/boot/grub/grub.cfg`. Dal contenuto del file si notano molti dettagli nuovi, in particolare:

- la possibilità di caricamento moduli con `insmod` per specifici filesystem o funzionalità aggiuntive;
- riorganizzazione del codice tramite l'utilizzo di `function ...`, con ovvi benefici per la leggibilità;
- il supporto della struttura logica condizionale `if/elif/else/fi`, permettendo il caricamento di funzioni, moduli o voci di menu solo per condizioni specifiche.

Per quanto riguarda le singole scelte di menu, di seguito un esempio:

```
1 menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu --
      class os $menuentry_id_option
      'gnulinux-simple-c6733fca-5211-4b85-ba28-a89d0da88bbf' {
2 ...
3     set root='hd0,gpt6'
4     linux   /boot/vmlinuz-5.15.0-116-generic root=UUID=c6733fca
      -5211-4b85-ba28-a89d0da88bbf ro
5     initrd  /boot/initrd.img-5.15.0-116-generic
6 }
```

Esempio 23.2: *Spezzone del file `/boot/grub/grub.cfg`*

in cui:

`menuentry`
: la *keyword* `title` è stata sostituita da `menuentry`, lo stesso titolo incluso in *quote*, e il blocco di riferimento dovrà essere tra parentesi graffe. In tale blocco si è riportato solo l'essenziale di un'entry generica;

`set root='hd0,gpt6'`
: è utilizzata la sintassi `set root=` per definire la *root partition*. Da notare che la numerazione delle partizioni non inizia più da 0 ma da 1;

`linux /boot/vmlinuz-*`
: indica il path completo dell'immagine kernel da avviare, la *partizione di root* tramite l'*UUID*[25] relativo e la modalità *readonly*;

[22] specificato dal `+1`
[23] ad esempio, `grub-install /dev/sda` per il primo settore del disco indicato
[24] v. `man grub-mkconfig`
[25] ***U**niversally **u**nique **id**entifier*, v. 23.6.2

`initrd /boot/initrd.img-*`
: indica il path completo dell'immagine *initrd*.

Per modifiche di configurazione, l'idea alla base è non modificare direttametne il file `/boot/grub/grub.cfg`, ma adoperare strumenti che validano e producano loro stessi un file di configurazione valido.
Ad esempio:

- per aggiungere o modificare entry, non si modificherà direttamente il file `/boot/grub/grub.cfg`, ma i file in `/etc/grub.d/` secondo la propria necessità, e si eseguirà il comando `grub-mkconfig`;
- per modificare variabili globali quali ad esempio:

```
1 GRUB_DEFAULT=0
2 GRUB_TIMEOUT=10
```

Esempio 23.3: *Esempio di variabili globali nel file `/boot/default/grub`*

  si modificherà il file `/etc/default/grub` e, successivamente alla modifica, si eseguirà il comando `update-grub`.

Alcune distribuzioni modificano direttamente tali file, ad esempio installando una nuova versione del kernel, ed eseguono automaticamente il relativo comando per generare la nuova configurazione finale.
In ultimo, qualora non si utilizzasse *GRUB* e si volesse installarlo, analogamente alla precedente versione, si utilizzerà `grub-install` *`device`*[26].

Kernel

Montato il filesystem di `/` con il valore indicato dall'opzione `root=`, se non esplicitato diversamente, terminato il caricamento lo stesso *kernel*[27] eseguirà il software `/sbin/init` a cui assegnerà il *PID* 1.

Init

Questo step è differente a seconda di quale software *init*[28] si utilizzi. Storicamente uno dei più diffusi è stato *System V*[29], ma spesso oggi si trovano software alternativi, ad oggi il più diffuso è `systemd`[30], ma non tutte le distribuzioni hanno optato per questa scelta[31].
In ogni caso *init* è il primo processo che il kernel eseguirà dopo aver terminato la fase di bootstrap. Esso ha il compito fondamentale di portare il sistema in uno stato operativo, avviando i programmi e servizi necessari.

Per quanto riguarda il sistema *System V*, il file `/etc/inittab` determina il **runlevel**[32].
Formalmente non esiste uno standard per ogni *runlevel*, di seguito lo schema di riferimento:

[26] ad esempio, `grub-install /dev/sda` per il primo settore del disco indicato
[27] l'argomento sarà esaminato in **??** a pag. **??**
[28] contrazione di *initialization*
[29] anche *SysV*, il nome deriva dalla versione di Unix *System V* che lo introdusse
[30] v. 23.5.1 a pag. 363
[31] v. **??** a pag. **??**
[32] rappresenta lo stato di attività del sistema relativamente al software in esecuzione e ai demoni attivi

runlevel 0, per l'arresto (*halt*) del sistema, corrisponde allo stato in cui nessun processo è in esecuzione, quindi il sistema sarà spento.

runlevel 1, denominato *single user mode*, corrisponde ad uno stato in cui non sono attivi software in background ed è identificato dalla sola presenza dell'amministratore. In altri termini, nessun altro utente ha accesso al sistema, per cui l'amministratore ha il pieno controllo del sistema. Questo runlevel è utilizzato per eseguire attività di manutenzione[33];

runlevel 2, il sistema sarà avviato in *multi user mode* e con il networking abilitato, ma non saranno attivati i demoni di rete;

runlevel 3, sarà attiva la modalità multiutente con funzionalità di networking (analogo al runlevel precedente) e saranno attivi i vari servizi;

runlevel 4, generalmente non è utilizzato;

runlevel 5, analogo al *runlevel 3*, in aggiunta sarà attivo il server grafico[34];

runlevel 6, utilizzato per effettuare il *reboot* del sistema. A differenza del *runlevel 0*, al termina il sistema non sarà spento ma riavviato.

Dalla breve descrizione proposta, si nota che Lo schema generale prevede che i runlevel 0, 1 e 6 sono riservati per uso del sistema, i rimanenti sono disponibili[35] per scopi propri e/o determinati dalla distribuzione.
Sintetizzando, lo schema potrebbe essere il seguente:

$0 \rightarrow$ *halt*

$1 \rightarrow$ *single user mode*

$2 \rightarrow$ *multi user mode* con limitazioni

$3 \rightarrow$ *full multi user mode*

$4 \rightarrow$ non utilizzato

$5 \rightarrow$ *graphical mode*

$6 \rightarrow$ *reboot*

in cui per questo testo si sta utilizzando un sistema con *runlevel 3*.
Si evidenzia nuovamente che la suddivisione a runlevel è un concetto centrale ma non rigido, e talune distribuzioni effettuano suddivisiono differenti[36].

[33] ad esempio, check dello storage, aggiornamento del sistema in maniera *protetta* (nel senso che il sistema è sotto controllo assoulto dell'amministratore)

[34] tendenzialmente *X Window System*

[35] come già accennato, possono essere utilizzate convenzioni diverse, ad esempio possono essere definiti ulteriori runlevel o utilizzati diversamente dalla propria distribuzione

[36] ad esempio, *Gentoo* utilizza runlevel nominali e quello di default si chiama banalmente `default`

Esecuzione del runlevel Al boot del sistema, saranno avviati diversi servizi a seconda del *runlevel* impostato. In particolare saranno eseguiti i programmi localizzati in una delle directory[37] di seguito:

`/etc/rc0.d/`

`/etc/rc1.d/`

`/etc/rc2.d/`

`/etc/rc3.d/`

`/etc/rc4.d/`

`/etc/rc5.d/`

`/etc/rc6.d/`

in cui `rcn.d` corrisponderà proprio al *runlevel*[38] definito.

Runlevel e relativi demoni All'interno delle directory tendenzialmente ci saranno link simbolici a script (solitamente contenuti nelle directory `/etc/init.d` oppure `/etc/rc.d` oppure `/etc/rc.d/init.d` a seconda della distribuzione) con il nome omologo ma con aggiunti:

una lettera `S` o `K` con:

`S` per *startup*, utilizzate durante la fase di startup;

`K` per *kill*, utilizzati durante la fase di *shutdown*.

un numero di due cifre il quale indicherà l'ordine[39] di esecuzione dello script medesimo.

Ovviamente gli script principali potranno essere avviati manualmente[40] con argomenti appropriati, i principali[41] sono: `status`, `start`, `stop` e `restart` le cui relative azioni sono autoesplicative.
Per cambiare la presenza (o assenza) del servizio su un runlevel si modificherà il link sulla directory del runlevel, ma è più curretto utilizzare stumenti appopriati, i più noti sono `chkconfig`[42] e `update-rc.d`[43].

---

[37] la struttura potrebbe variare, ad esempio la distribuzione *Slackware* opta per il metodo tradizionale *BSD* con le directory sostituite da script `/etc/rc.d/rc.n`) ciascuna con lo scopo di eseguire quanto necessario per portare il sistema al runlevel richiesto, oppure una generica directory `/etc/rc.n`) potrebbe essere un link alla directory `/etc/rc.d/rc.n`)

[38] talvolta si utilizza il runlevel `S` o `s`, in cui la lettera è per *single user mode*, utilizzato (quasi) come sinonimo di *runlevel 1*; tipicamente `S` o `s` proiettano in una shell root senza il montaggio del filesystem, `1` tenta il montaggio del filesystem e avvia taluni software di sistema

[39] l'esecuzione sarà con sequenza crescente, quindi un numero inferiore sarà eseguito prima del maggiore. Ad esempio `S10daemonA` avvierà il `daemonA` con numero di sequenza 10; `S81daemonB` ha numero di sequenza 81, quindi sarà avviato dopo `daemonA`

[40] ad esempio su questo sistema lo script `/etc/init.d/ssh` è per il servizio *ssh*

[41] quasi tutte sono supportate da ciascun script

[42] v. **??** a pag. **??**

[43] v. **??** a pag. **??**

Determinazione e modifica del proprio runlevel Come già accennato, nei sistemi con *System V* è possibile verificare il proprio runlevel consultando il file `/etc/inittab`, oppure semplicemente utilizzando il comando `grep` come segue:

```
1 $ grep :initdefault:  /etc/inittab
2 id:3:initdefault:
```

Esempio 23.4: *`grep` su `/etc/inittab`*

in cui è evidente il relativo runlevel.
Una manierà più elegante per ottenere tale informazione è l'utilizzo del comando `runlevel` come segue:

```
1 $ runlevel
2 N 5
```

Esempio 23.5: *Esecuzione di `runlevel`*

in cui:

- il primo carattere rappresenta il runlevel precedente, ma in questo caso è `N` poiché il sistema non ha avuto in precedente runlevel da quando è stato avviato[44];
- il secondo carattere mostra il corrente runlevel, ed è `5` poiché è stato eseguito su un sistema desktop.

Tornando al file `/etc/inittab`, tramite editazione sarà possibile modificare il runlevel con il numero desiderato ed esso sarà applicato al successivo avvio del sistema.

Switch di runlevel Su un sistema attivo è possibile switchare il proprio runlevel tramite il comando `init` *`n`* o `telinit` *`n`*, con *n* il nuovo runlevel. A differenza del primo comando che è proprio il primo processo eseguito dal kernel, tramite il comando `telinit` il runlevel può essere switchato in due modi:

- come già scritto, passando come opzione il numero che indentifica il nuovo livello;
- passando come opzione la lettera `Q` o `q` che indurranno la rilettura del file `/etc/initab` a cui si può aver precedentemente cambiato il runlevel.

È certamente utile verificare la sintassi tramite i rispettivi manuali[45], ma in molte distribuzioni il comando `telinit` è un link simbolico a `init`, per cui anche `init` potrà ricevere i medesimi parametri[46].

Un paio di esempi:

```
1 # init 0
```

Esempio 23.6: *Switch di runlevel tramite `init 0`*

switcherà il sistema al runlevel 0, ciò equivale allo spegnimento della macchina.
Un secondo esempio:

[44] su un sistema attivo è possibile *switchare* a runlevel differenti tramite le istruzioni mostrate nel prossimo paragrafo

[45] v. `man init` e `man telinit`

[46] oltre `Q` e `q` per la rilettura del runlevel nel file `/etc/inittab`, sicuramente saranno valide le opzioni `S` e `s` per il *rescue mode*, quindi utilizzate equivalentemente all'opzione `1`

```
1 # init 6
```

Esempio 23.7: *Switch di runlevel tramite* `init 6`

switcherà il sistema al runlevel 6, ciò equivale allo riavvio del sistema.
Metodi più eleganti per lo spegnimento e il riavvio del sistema saranno in 23.5 a pag. 360.

23.3 Notifiche di sistema

Alle volte può essere utile segnalare messaggi agli amministratori di sistemi[47] tra loro stessi o semplicemente un *reminder* o specificità del sistema da ricordare a se stessi.
Di seguito saranno illustrate alcune tipiche modalità.

Messaggio pre-login

Quando gli utenti tentano di loggarsi in modalità testuale, nella fase di login l'host mostrerà vari messaggi. Uno di questi è contenuto nel file `/etc/issue`[48] il quale solitamente identifica il sistema, ma può contenere una descrizione breve e, come mostrato in ***Tabella 23.1***, codici escape per mostrare il nome dell'host, il numero di versione del kernel, l'orario, ...

CODICE	DESCRIZIONE
`b`	baudrate della linea corrente
`d`	data corrente
`s`	nome del sistema
`l`	nome della linea tty corrente
`m`	identificatore dell'architettura hardware
`n`	hostname
`o`	nome di dominio
`r`	numero di release del kernel
`t`	ora corrente
`u`	numero degli utenti attualmente connessi
`U`	la stringa `1 user` o *`n`* `users` in cui *n* è il numero di utenti attualmente connessi
`v`	versione del sistema

Tabella 23.1: *Codici escape che possono essere utilizzati nel file* `/etc/issue`

I codici escape mostrati consistono nel carattere *backslash*[49] seguito immediatamente da una delle lettere indicate. Ad esempio, la stringa `\n \l` inserirà l'hostname, uno spazio, e il nome della linea tty corrente.

[47] ad esempio in caso di una moltitudine di sistemi amministrati da più sistemisti

[48] v. `man issue` e `man agetty`; per le connessione da rete, il file di riferimento è solitamente `/etc/issue.net`, ma il suo uso andrà configurato

[49] \

Message of the day

Spesso con l'aconimo **motd**, esso è un messaggio memorizzato nel file `/etc/motd`. Tale file, se esiste, avrà un contenuto che sarà visualizzato dopo un login testuale avvenuto con successo.
Per tale motivo, solitamente in tale file saranno memorizzate informazioni che si vorranno comunicare all'utenza, quali specificità del sistema o eventi prossimi (upgrade, downtime schedulati, ...).

Messaggi in tempo reale

Due strumenti permettono l'invio agli utenti di messaggi in real time:

`shutdown`: messaggi di importanza estrema sono sicuramente quelli di *shutdown*, per cui attraverso tale comando sarà possibile inviare alert tramite messaggio di testo a tutti gli utenti. Ad esempio, il comando

```
1 # shutdown -h +10 "System going down for maintenance"
```

Esempio 23.8: *Esempio con il comando* `shutdown`

invierà il suddetto messaggio a tutti gli utenti con frequenza crescente fino all'operazione di *shutdown* impostata. Per maggiori dettagli, tale comando sarà esaminato in 23.5 a pag. 361;

`wall`[50]: il comando permette di inviare del testo o il contenuto di un file a tutti i terminali attivi, può essere invocato da tutti gli utenti e i destinatari saranno tutti gli utenti del sistema (a meno che non si definisca come opzione un gruppo, nel qual caso solo i membri attivi con un terminale saranno i destinatari del messaggio).

23.4 Evoluzioni nell'avvio

Nella precedente sezione si è esaminato il processo di bootstrap come in passato avveniva nella maggior parte dei sistemi. Tuttora in molti seguono il medesimo schema, ma nel tempo sempre più hanno adottate step alternativi.
Di seguito saranno presentate le principali evoluzioni.

23.4.1 Da BIOS a UEFI

Tradizionalmente si è utilizzato il *BIOS*, ma questo firmware ha iniziato a mostrare i limiti della sua vecchia concezione, per cui si è passati all'**UEFI**[51].
Per essere precisi,l'*UEFI* non ha sostituito il *BIOS* che è ancora il firmware incorporato nella scheda madre, ma l'*UEFI* ha aggiunto funzionalità rispetto la precedente, in particolare:

- strumenti per la diagnostica;
- servizi di crittografia;

[50] ***w****riting to* ***all*** *terminal*
[51] acronimo di ***U****Unified* ***E****xtensible* ***F****irmware* ***I****nterface*

- interfaccia grafica;
- gestione remota del sistema;
- supporto dello standard *GPT*[52].

Come appena accennato, per ottenere le funzionalità aggiuntive, l'*UEFI* ha esteso le funzionalità del *BIOS* attraverso del software contenuto nella partizione *EFI*, partizione[53] che costituisce l'interfaccia[54] tra firmware e SO.

23.4.2 Da MBR a GPT

In 23.2, al termine del paragrafo relativo all'**MBR**, sono state evidenziate alcune limitazioni di tale schema.
L'evoluzione è il *GUID*[55] *Partition Table*, noto come **GPT**, standard per la definizione della tabella delle partizioni ed è parte dello standard *UEFI*[56]. Lo schema di partizionamento è più flessibile, per la descrizione delle partizioni si utilizzano 64 bytes e quindi si supera la limitazione della partizione. Inoltre sono risolti altri problemi minori quali il massimo numero delle partizioni, il backup della tabella medesima, ... In particolare, l'utilizzo di 64 bytes consente di definire un disco fino a 9.4 ZB[57]

23.5 Spegnimento e riavvio

Da tener conto che le distribuzioni non si comportano in maniera completamente equivalente in relazione alla famiglia dei comandi che saranno esaminati, ma approssimando gli step che prevedono lo spegnimento sono i seguenti:

1. avviso dell'utenza;
2. esecuzione di eventuali task programmati;
3. sincronizzazione e smontaggio dei filesystem;
4. stop dei servizi di sistema;
5. spegnimento dei vari dispositivi collegati;
6. unload dei moduli del kernel;
7. stop delle sessioni utente;
8. disabilitiazione terminali;
9. disabilitazione logging;
10. passaggio del controllo all'hardware per l'arresto.

[52] v. prossima sezione
[53] è in formato `FAT32` ed è collocata all'inizio del disco o su eventuale storage esterno
[54] è possibile aggiungere ulteriore software scritto con strumenti di alto livello ed eseguito direttamente tramite interfaccia UEFI, quindi prima dell'avvio del SO
[55] contrazione di ***G**lobally **U**nique **Id**entifier*
[56] v. sezione precedente
[57] $\cong 9.4 \cdot 10^{21} bytes$, quindi 9.4 zettabytes

Come accennato precedentemente, tramite il comando `init` o `telinit` è possibile passare a nuovi runlevel, in particolare negli esempi 23.6 e 23.7 a pag. 357 si è rispettivamente spento e riavviato il sistema.
Questa procedura ha però degli svantaggi poiché:

- non è intuitiva in quanto il comando non è mnemonicamente associato all'azione che compirà;
- l'azione è immediata, quindi eventuali utenti collegati subiranno anch'essi immediatamente le conseguenze del cambio di runlevel.

Sono stati perciò previsti comandi alternativi[58] per lo spegnimento o il ravvio del sistema, i quali saranno esaminati nei prossimi paragrafi con le principali opzioni relative.

`shutdown` Nella classe di comandi per lo spegnimento o il riavvio, è da considerare quello più completo. La sintassi completa è la seguente:

```
1  shutdown [OPTIONS...]  [TIME] [WALL...]
```

Esempio 23.9: *Sintassi generica del comando* `shutdown`

in cui:

`OPTIONS` indica le opzioni, le più comuni sono:

`-H` o `--halt`
: per avviare la procedura di *halt* del sistema;

`-P` o `--poweroff`
: per avviare la procedura di *poweroff* del sistema;

`-r` o `--reboot`
: per avviare la procedura di *poweroff* della macchina.

`-c`
: per cancellare uno *shutdown pending*[59];

`TIME` è una stringa opzionale[60] la quale indica il tempo da attendere prima di eseguire il comando, i più comuni sono:

`now`, per forzare l'esecuzione immediata;

`hh:mm`, in cui `hh` indica l'ora e `mm` i minuti in un ciclo di 24 ore;

`+m`, con `m` che indica il numero di minuti dal momento dell'esecuzione[61];

nessun argomento implica `+1`;

`WALL`[62] è il messaggio opzionale inviato a tutti gli utenti collegati.

[58] si ricorda il comportamento variabile per distribuzione, in particolare ciascun sistema *init* esaminato in **??** a pag. **??** avrà le proprie implementazioni

[59] ovviamente sono da escludere gli `shutdown` con argomento `+0` o equivalentemente `now` poiché sono immediati (v. argomento `TIME`)

[60] se tale argomento è indicato, 5 minuti prima che il sistema effettui la procedura di *down* del sistema, sarà creato il file `/run/nologin` cosicché non sarà più possibile effettuare ulteriori login

[61] quindi la stringa `now` è un alias di `+0`

[62] l'argomento è omonimo del comando `wall` il quale ha lo scopo di inviare un messaggio a tutti gli utenti collegati. Per ulteriori, v. `man wall`

Alcuni esempi:

- `shutdown` senza alcun parametro:

```
1 # shutdown
2 Shutdown scheduled for Tue 2024-08-20 12:47:35 UTC, use 'shutdown -c'
      to cancel.
```

Esempio 23.10: *`shutdown` di default*

Gli altri utenti connessi saranno avvistati con una notifica:

```
1 Broadcast message from root@nb on pts/5 (Tue 2024-08-20 12:46:35 UTC)
     :
2
3 The system is going down for poweroff at Tue 2024-08-20 12:47:35 UTC!
```

Esempio 23.11: *Notifica di `shutdown`*

e non saranno possibili nuove sessioni da utenti non privilegiati:

```
1 System is going down. Unprivileged users are not permitted to log in
     anymore. For technical details, see pam_nologin(8).
```

Esempio 23.12: *Notifica di `shutdown`*

- cancellazione `shutdown` pendenti:

```
1 # shutdown -c
```

Esempio 23.13: *Cancellazione `shutdown` pendente*

Nonostante non sia mostrata nessuna notifica di cancellazione, gli utenti già loggati saranno notificati:

```
1 Broadcast message from root@nb on pts/5 (Tue 2024-08-20 12:46:43 UTC)
     :
2
3 The system shutdown has been cancelled
```

Esempio 23.14: *Avviso di cancellazione `shutdown` pendente*

- `shutdown` programmato con messaggio in broadcast:

```
1 # shutdown +10 Shutting down in 10 minutes!
2 Shutdown scheduled for Tue 2024-08-20 13:07:22 UTC, use 'shutdown -c'
      to cancel.
```

Esempio 23.15: *`shutdown` programmato con messaggio di broadcast*

`halt` Il comando `halt` equivale al comando `shutdown` con l'aggiunta dell'opzione `-H` o equivalentemente `--halt`. Esso terminerà i processi ma tendenzialmente[63] non disabiliterà l'alimentazione (manterrà lo stato di *power on*).

[63] talune implementazioni possono comportarsi in maniera differente

poweroff Il comando `poweroff` equivale al comando `shutdown` con l'aggiunta dell'opzione `-P` o equivalentemente `--poweroff`. Differentemente dall'implementazione del comando `halt`, esso terminerà i processi e successivamente disalimenterà la macchina.
Da segnalare che nei sistemi molto datati la *CPU* non aveva la gestione dell'alimentazione quindi, dopo lo spegnimento dei dispositivi collegati, era comunque necessaria una disalimentazione tramite il bottone di *power-off*.

reboot Il comando `reboot` equivale al comando `shutdown` con l'aggiunta dell'opzione `-r` o equivalentemente `--reboot`. Differentemente dall'implementazione del comando `halt`, esso terminerà i processi e successivamente avvierà nuovamente il sistema.

23.5.1 Da `System V` a `systemd`

Init alternativi

Come già accennato, storicamente uno dei più diffusi sistemi di *init* è stato *System V*, ma oggi il più adottato è `systemd`.

Configurazione di un servizio

`systemctl`

23.6 Approfondimenti

23.6.1 Bootloader

23.6.2 Identificazione dei dischi e delle partizioni

Ogni volta che il computer eseguirà il *POST*[64], ad ogni dispositivo sarà assegnato un identificativo in cui:

- la radice identificherà la tecnologia di interfacciamento[65], ad esempio sarà `hd` per l'*IDE*, `sd` per *SCSI*, *nvme* per i dispositivi *NVMe*;
- un numero *n* che identificherà l'ordine assegnato, quindi texttt0 per il primo dispositivo, `1` per il secondo, ...

Se la radice rimarrà sempre la stessa, non sarà garantita la coerenza della numerazione assegnata per successivi avvii[66], quindi per più dispositivi connessi alla stessa scheda l'identificativo assegnato potrebbe variare tra un avvio e il successivo. Si intuisce che tale problema è legato sia al bootloader sia al sistema in generale[67].
Per tale motivo, nelle sezioni successive si forniranno comandi utili per l'identificazione

[64] v. 23.2 a pag. 348
[65] ad esempio v. 21.3 a pag. 299 oppure **??** a pag. **??**
[66] per essere precisi, gli hard disk IDE sono numerati secondo un ordine predeterminato che sarà `hd0` per il master primario, `hd1` per il suo slave, `hd2` per il master secondario e `hd3` per lo slave di quest'ultimo. Ai dischi SCSI l'assegnazione numerica avviene non tenendo conto dell'identificativo SCSI
[67] si pensi al file `/etc/fstab`

univoca dei dispositivi e delle partizioni tramite l'*UUID*[68] a loro assegnati e, ovviamente, è consigliato utilizzare tali *UUID* per l'identificazione sia in GRUB sia in `/etc/fstab`, e dovunque sia richiesta un identificazione stabile.

`lsblk`

Per un elenco dei dispositivi e relativi partizioni, come già avvenuto a pag. 145 è possibile utilizzare il comando `lsblk` come di seguito:

```
1 $ lsblk
2 NAME                MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
3 nvme0n1             259:0    0 953,9G  0 disk
4 |-nvme0n1p1         259:1    0   512M  0 part /boot/efi
5 |-nvme0n1p2         259:2    0 465,7G  0 part /
6 |-nvme0n1p3         259:3    0 487,7G  0 part
7   |--cryptpart-crypt 252:0    0 487,7G  0 lvm
8 sr0                  11:0    1  1024M  0 rom
```

Esempio 23.16: *Esecuzione del comando* `lsblk`

in cui è mostrato il disco con le relative partizioni e un device ottico. Si consiglia di consultare il relativo manuale per le opzioni.

`blkid`

Il comando `blkid` mostra gli attributi dei device a blocchi. Di seguito un esempio:

```
1 $ blkid
2 /dev/sda1: LABEL="SYSTEM" UUID="0E04-DB5F" TYPE="vfat" PARTLABEL="EFI
      system partition" PARTUUID="f57e8508-69ce-42bf-b047-5efc868cbce0"
3 /dev/sda3: UUID="f75840f4-59f9-4fd3-9404-c90f6c81561f" TYPE="crypto_LUKS"
      PARTUUID="06f1fff2-fb43-494f-80ef-b582169272ce"
4 /dev/sda4: LABEL="KVM" UUID="94b6afff-a5af-40db-a2e6-b6ff38c11707" TYPE="
      ext4" PARTUUID="0ddb53c6-9e9d-4fb1-83a2-b49971c17629"
5 /dev/sda6: UUID="c6733fca-5211-4b85-ba28-a89d0da88bbf" TYPE="ext4"
      PARTUUID="128fa7e9-7ca0-4f9b-9d5d-9aaff5d289db"
6 /dev/sda7: UUID="793aed9f-b3d6-4523-b2fe-8069d603c8ae" TYPE="swap"
      PARTUUID="65886bcd-df05-4421-84a7-cd34fd7a0606"
```

Esempio 23.17: *Esecuzione del comando* `lsblk`

in cui per ogni partizione è mostrato il relativo *UUID*[69], la tipologia di partizione e ulteriori informazioni.
Si consiglia di consultare il relativo manuale per le opzioni.

[68] ***U**niversally **u**nique **id**entifier*
[69] ***u**niversally **u**nique **id**entifier*

23.7 Concetti chiave

BIOS: firmware che ha il compito di effettuare un check dello stato del sistema e passare il controllo dell'avvio al *boot loder*;

boot loader: software eseguito nella fase di avvio del sistema a cui è affidata l'esecuzione del kernel a cui passerà il controllo del sistema;

EFI: ***E**xtensible **F**irmware **I**nterface*, evoluzione del BIOS (v. emphUEFI);

GRUB: bootloader molto popolare;

MBR: acronimo di ***M**aster **B**oot **R**ecord*, sarà la sequenza che prenderà il controllo del sistema nel momento della sua esecuzione;

runlevel: rappresenta lo stato di attività del sistema relativamente al software in esecuzione e ai demoni attivi

systemd: una solizione *init* molto popolare;

UEFI: acronimo di ***U**nified **E**xtensible **F**irmware **I**nterface*, ingloba il BIOS estendendone la funzionalità;

23.8 Comandi introdotti nel capitolo

`blkid` ⇒ mostra gli attributi del device a blocchi
`chkconfig` ⇒ utility per l'attivazione/disattivazione di demoni su runlevel
`grub-install` ⇒ installa il bootloader *GRUB*
`grub-mkconfig` ⇒ genera il file di configurazione di *GRUB*
`halt` ⇒ halt del sistema
`init` ⇒ switch su un nuovo runlevel
`poweroff` ⇒ poweroff del sistema
`reboot` ⇒ reboot del sistema
`runlevel` ⇒ mostra il runlevel precedente e corrente
`shutdown` ⇒ shutdown del sistema
`telinit` ⇒ switch su un nuovo runlevel
`update-grub` ⇒ rigenera il file di configurazione di *GRUB*
`update-rc.d` ⇒ utility per l'attivazione/disattivazione di demoni su runlevel
`wall` ⇒ utilizzato per inviare un messaggio agli altri utenti del sistema

Capitolo 24

Logging

QUESTO capitolo costituisce una prosecuzione e ampliamento di quanto affrontato superficialmente in 12.3.11 a pag. 141 in relazione alla directory `/var/log`, brevemente approfondita in 12.6 a pag. 144.

24.1 Introduzione

Nei file di log sono riportate sequenze temporizzate degli eventi di sistema e delle applicazioni, in particolare il loro contenuto è tendenzialmente suddiviso per settori di interesse. I log sono estremamente utili per effettuare:

- verifiche del corretto funzionamento, ad esempio visualizzando messaggi ordinari;
- troubleshooting tramite messaggi che richiedono maggior attenzione quali *warning* ed *errori*;

Inoltre sono spesso utilizzati per attività di monitoraggio (ad esempio per valutazioni di performance o carichi di lavoro), eventualmente tramite un successivo processamento e talvolta aggregazione secondo la specifiche di interesse.

Nei sistemi GNU/Linux si utilizza il protocollo **syslog**[1], ma è necessaria una prima suddivisione tra i messaggi di logging in *kernel space* e *user space*.

24.2 Kernel space

Durante il processo di boot, in console sono visualizzati molti messaggi relativi alla fase dell'avvio del sistema. Per evitare che questi messaggi si perdano, il kernel utilizza una struttura dati ciclica denominata **ring buffer**[2] il cui compito è memorizzare tutti i messaggi inviati dal kernel medesimo.
In questo step di avvio, i messaggi saranno generati dalla funzione `printk()` definita ovviamente nello stesso codice. Tale funzione invierà i messaggi al *ring buffer* e questo, se riempito, sarà riscritto circolarmente. Successivamente i messaggi contenuti nel buffer

[1] v. [Ger09]

[2] anche *circular buffer*, *circular queue* o *cyclic buffer*, un buffer di dimensione fissa codificata nel codice; tale struttura permette la memorizzazione agevole di flussi di dati

saranno letti e memorizzati su disco tramite il servizio *syslog*.
Gli amministratori potranno consultare tali messaggi utilizzando il comando `dmesg` (o la consultazione dei file `/var/log/dmesg.log*`).

24.2.1 Kernel log levels

Analogamente alla tradizionale funzione `printf()` definita in *user space*, il kernel utilizza la propria funzione `printk()`[3] con una stringa e ulteriori parametri per definire il messaggio. In aggiunta, la funzione `printk()` può specificare un *log level* per definire la severità del messaggio in accordo a quanto mostrato in ***Tabella 24.1***.

LOG LEVEL	STRING	DESCRIZIONE
KERN_EMERG	"0"	Emergency messages
KERN_ALERT	"1"	errore che richiede immediata attenzione
KERN_CRIT	"2"	errore critico (hardware o software)
KERN_ERR	"3"	verificata condizione di *errore*
KERN_WARNING	"4"	verificata condizione di *warning*
KERN_NOTICE	"5"	non è un errore ma una condizione da notificare
KERN_INFO	"6"	messaggio informativo
KERN_DEBUG	"7"	messaggio di debug
KERN_DEFAULT	""	livello di log di default
KERN_CONT	"c"	*continuation log line*

Tabella 24.1: *I kernel log level. Se omesso sarà utilizzato KERN_DEFAULT*

Ricevuto il messaggio, il kernel deciderà se mostrarlo immediatamente alla console in relazione al livello di severità dello stesso e quanto impostato nella variabile kernel `console_loglevel`, in particolare solo se il messaggio avrà priorità maggiore o uguale (quindi *log level* minore o uguale) sarà mostrato in console.
Tali messaggi potranno essere successivamente visualizzati ricorrendo al comando `dmesg`[4] come di seguito:

```
1 $ dmesg
2 [    0.000000] kernel: microcode: microcode updated early to revision 0x26, date = 2019-11-12
3 [    0.000000] kernel: Linux version 5.15.0-105-generic (buildd@lcy02-amd64-054) (gcc (Ubuntu
      9.4.0-1ubuntu1~20.04.2) 9.4.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #115~20.04.1-Ubuntu SMP
      Mon Apr 15 17:33:04 UTC 2024 (Ubuntu 5.15.0-105.115~20.04.1-generic 5.15.148)
4 [    0.000000] kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-105-generic root=UUID=c6733fca
      -5211-4b85-ba28-a89d0da88bbf ro
5 ...
```

Esempio 24.1: `dmesg`

Ovviamente il buffer potrebbe essere sovrascritto da eventi più recenti, quindi è possibile che sia mostrato quanto segue:

```
1 $ dmesg
2 [1729926.330811] x86: Booting SMP configuration:
```

[3] anche chiamata *print kernel*. Per approfondire v. [coma]

[4] talvolta sarà necessario eseguire il comando da *superutente*. Per approfondire v. 24.6.1 a pag. 392

```
3 [1729926.330813] smpboot: Booting Node 0 Processor 1 APIC 0x2
4 [1729926.337324] CPU1 is up
5 [1729926.337403] smpboot: Booting Node 0 Processor 2 APIC 0x1
6 [1729926.339982] CPU2 is up
7 [1729926.340055] smpboot: Booting Node 0 Processor 3 APIC 0x3
8 [1729926.342377] CPU3 is up
9 ...
```

Esempio 24.2: *`dmesg` con buffer sovrascritto*

in cui si notano le marche temporali in secondi (come `[1729926.330811]`) molto spostate da quelle precedenti.

Tra i flag utili del comando ci sono:

`-H` o `--human`
: il quale mostra lo stesso output con la marca temporale leggibile e inviato automaticamente al pager `less`:

```
1 $ dmesg -H
2 [mag20 12:40] kernel: microcode: microcode updated early to revision 0x26, date = 2019-11-12
3 [  +0,000000] kernel: Linux version 5.15.0-105-generic (buildd@lcy02-amd64-054) (gcc (Ubuntu
        9.4.0-1ubuntu1~20.04.2) 9.4.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #115~20.04.1-
        Ubuntu SMP Mon Apr 15 17:33:04 UTC 2024 (Ubuntu 5.15.0-105.115~20.04.1-generic
        5.15.148)
4 ...
```

Esempio 24.3: *`dmesg -H`*

in cui dalla seconda linea sono mostrati i ritardi del relativo evento rispetto il precedente;

`-T` o `--ctime`

```
1 $ dmesg -T
2 [lun mag 20 12:40:56 2024] kernel: microcode: microcode updated early to revision 0x26, date
        = 2019-11-12
3 [lun mag 20 12:40:56 2024] kernel: Linux version 5.15.0-105-generic (buildd@lcy02-amd64-054)
        (gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #
        115~20.04.1-Ubuntu SMP Mon Apr 15 17:33:04 UTC 2024 (Ubuntu 5.15.0-105.115~20.04.1-
        generic 5.15.148)
4 ...
```

Esempio 24.4: *`dmesg -T`*

in cui ciascuna linea riporta li tempo assoluto dell'evento;

`--time-format` *`format`* in cui *format* va sostituito con `ctime`, `reltime`, `delta` p `iso` a seconda del formato voluto. Utilizzando `ctime` si otterrà lo stesso output mostrato nell'***Esempio 24.4***, scegliendo `iso` si otterrà la marca temporale secondo la specifica *ISO-8601*[5] quindi secondo lo schema `YYYY-MM-DD<T>HH:MM:SS,<microseconds><-+><timezone offset from UTC>`:

```
1 $ dmesg --time-format iso
2 2024-05-20T12:40:56,330813+02:00 kernel: microcode: microcode updated early to revision 0x26
        , date = 2019-11-12
3 2024-05-20T12:40:56,330813+02:00 kernel: Linux version 5.15.0-105-generic (buildd@lcy02-
        amd64-054) (gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0, GNU ld (GNU Binutils for Ubuntu
        ) 2.34) #115~20.04.1-Ubuntu SMP Mon Apr 15 17:33:04 UTC 2024 (Ubuntu
        5.15.0-105.115~20.04.1-generic 5.15.148)
4 ...
```

Esempio 24.5: *`dmesg --time-format iso`*

[5] il formato permette la comparazione dei timestamp riportati dallo stesso sistema o tra sistemi diversi, utile nel parsing e nell'ordinamento

`-x` o `--decode`
affinché ogni messaggio mostri le relative *facility*[6] e *priority*, come di seguito:

```
1 $ dmesg --decode
2 kern  :info  : [3881648.684436] usb 1-6: new full-speed USB device number 61 using xhci_hcd
3 kern  :info  : [3881648.833472] usb 1-6: New USB device found, idVendor=13d3, idProduct
      =3423, bcdDevice= 0.01
4 kern  :info  : [3881648.833485] usb 1-6: New USB device strings: Mfr=0, Product=0,
      SerialNumber=0
5 kern  :info  : [3881648.877528] usb 1-6: USB disconnect, device number 61
6 ...
```

Esempio 24.6: `dmesg --decode`

Nei prossimi due punti saranno argomentate rispettivamente le *facilities* e le *priorities*.

`-f` *list* o `--facility` *list*
in cui *list* sarà una lista contenente una o più *facilities*. Ad esempio di seguito sarà utilizzata la facility `kern`:

```
1 $ dmesg --facility kern
2 [3881648.684436] usb 1-6: new full-speed USB device number 61 using xhci_hcd
3 [3881648.833472] usb 1-6: New USB device found, idVendor=13d3, idProduct=3423, bcdDevice=
      0.01
4 [3881648.833485] usb 1-6: New USB device strings: Mfr=0, Product=0, SerialNumber=0
5 [3881648.877528] usb 1-6: USB disconnect, device number 61
6 ...
```

Esempio 24.7: `dmesg --facility kern`

L'elenco delle facilities è mostrato in ***Tabella 24.2*** e, qualora si volessero esprimerne più di una, nella lista fornita andranno separate dal carattere virgola.

LOG FACILITIES

`kern`	kernel messages
`user`	random user-level messages
`mail`	mail system
`daemon`	system daemons
`auth`	security/authorization messages
`syslog`	messages generated internally by syslogd
`lpr`	line printer subsystem
`news`	network news subsystem

Tabella 24.2: *Le log facilities gestite dalla* `printk()` *e* `dmesg`

`-l` *list* o `--levels` *list*
in cui *list* sarà una lista contenente uno o più *livelli*. Ad esempio di seguito sarà utilizzata la priorità `info`:

```
1 $ dmesg --level info
2 [3881648.684436] usb 1-6: new full-speed USB device number 61 using xhci_hcd
3 [3881648.833472] usb 1-6: New USB device found, idVendor=13d3, idProduct=3423, bcdDevice=
      0.01
4 [3881648.833485] usb 1-6: New USB device strings: Mfr=0, Product=0, SerialNumber=0
5 [3881648.877528] usb 1-6: USB disconnect, device number 61
6 ...
```

Esempio 24.8: `dmesg --level info`

[6] per comprendere la natura di questo campo nel *kernel ring buffer* v. 24.6.1 a pag. 392

Confrontandolo con l'***Esempio 24.8***, si nota che i messaggi mostrati sono stati filtrati sia con la facility `kern` sia con la priorità `info`, quindi si intuisce facilmente che sarebbero in output anche utilizzando contemporaneamente i due filtri `--facility kern --level info`.
L'elenco delle priorità è mostrato in ***Tabella 24.3*** e, qualora si volessero esprimerne più di una, nella lista fornita andranno separate dal carattere virgola.

LOG LEVELS (PRIORITIES)

`emerg`	il sistema non è usabile
`alert`	è necessaria un'azione immediata
`crit`	condizione *critica*
`err`	condizione di *errore*
`warn`	condizione di *warning*
`notice`	comportamento ordinario ma è una condizione significativa
`info`	messaggi *informativo*
`debug`	messaggi di *debug*

Tabella 24.3: *I log levels (priorità) gestiti dalla `printk()` e `dmesg`*

`-C` o `--clear`
questo flag è stato lasciato per ultimo poiché cancellerà il contenuto del buffer e solo eventi successivi torneranno a popolarlo:

```
1 $ sudo dmesg -C
2 $ dmesg
3 $
```

Esempio 24.9: *`dmesg -C`*

Il flag per il `clear` del buffer ha la necessità di essere digitato dal superutente, la seconda e terza linea per mostrare che il buffer si è effettivamente svuotato e sarà ripopolato da eventi successivi.

24.3 User space

Come accennato nell'introduzione, per lo *user space logging* si utilizza il protocollo *syslog*. Nell'architettura di network, il sistema di syslog prevede una comunicazione client/server formata da 3 componenti:

originator: il syslog client, creatore e mittente dei messaggi;

relay: inoltra i messaggi ricevuti dai client verso il *collector*;

collector: il syslog server, riceve i messaggi e li storicizza.

In una tipica configurazione standalone, il sistema ovviamente non utilizzerà un *relay*, ma funzionerà sempre in modalità client/server: i processi e i servizi genereranno log utilizzando la *system call* `syslog`[7], il server di sistema (tipicamente `rsyslogd`) intercetterà il messaggio e lo storicizzerà su filesystem.
In ogni caso, sia nell'architettura standalone sia di network, lo standard syslog prevede che

[7] v. `man3 syslog`

ogni messaggio di log sia etichettato con un prefisso il quale indicherà sia la *facility* sia la *priority* esaminate rispettivamente nelle due prossime sezioni.

24.3.1 Syslog facilities

Nel sistema molti programmi e servizi genereranno log, per cui il protocollo definisce un'etichetta definita *facility* per individuare la tipologia di messaggio[8] che potrà essere gestito differentemente in base a regole impostate nel relay o nel collector.
In ***Tabella 24.4*** sono mostrate le facilities definite dal protocollo.

FACILITY CODE	KEYWORD	DESCRIZIONE
0	kern	messaggi kernel
1	user	messaggi utente
2	mail	messaggi generati da un mail system
3	daemon	messaggi generati da tutti i demoni che non hanno assegnata una facility propria
4	auth	messaggi di sicurezza generati da un sistema di autenticazione (ad esempio login, su, sudo, imapd, ftpd, ...)
5	syslog	messaggi generati direttamente dal server syslogd o rsyslogd
6	lpr	sistema di stampa
7	news	sistema *news*
8	uucp	sistema *UUCP*
9	cron	sistema cron
10	authpriv	messaggi di sicurezza/autenticazione
11	ftp	demone FTP
12	ntp	sistema *Network Time Protocol*
13	security	log di verifica di sicurezza
14	console	log *alert*
15	solaris-cron	demone di schedulazione
da 16 a 23	da local0 a local7	riservati per uso locale

Tabella 24.4: *Le log facilities*

24.3.2 Syslog priorities

La *priority* definisce il livello di priorità di un messaggio che varierà tra il *debug* (bassissima priorità, livello 7) all'*emergenza* (altissima priorità, livello 0). In ***Tabella 24.5*** sono mostrate le priorities definite dal protocollo, tabella analogia alla 24.3 a pag. 371.

[8] indicativamente quale classe di programmi ha generato il messaggio e lo scopo dello stesso

VALORE	PRIORITÀ	KEYWORD	DESCRIZIONE
0	*emergency*	`emerg`	sistema non usabile
1	*alert*	`alert`	dev'essere prestata attenzione immediata
2	*critical*	`crit`	condizione critica
3	*error*	`err`	condizione di errore
4	*warning*	`warning`	condizione di *warning*
5	*notice*	`notice`	condizione ordinaria ma significativa
6	*informational*	`info`	messaggio informativo
7	*debug*	`debug`	messaggio di debug

Tabella 24.5: *Le log priority*

24.3.3 File di log

I file di log sono tradizionalmente gestiti da un servizio che implementa il protocollo *syslog*, in particolare in passato le distribuzioni installavano il demone `syslogd` ora rimpiazzato dal `rsyslogd`. Rispetto al precedente, quest'ultimo prevede un filtraggio più flessibile, molte più opzioni di configurazione, criptazione dei messaggi nella fase di relay e altre caratteristiche[9] riuscendo a mantenere retrocompatibilità con `syslogd`.
Per la consultazione dei file di log, sarà estremamente utile utilizzare gli strumenti presentati ai capitoli 5 a pag. 63, 9 a pag. 103 e 8 a pag. 91 e diffusamente utilizzati nel testo.

24.3.4 Configurazione `rsyslogd`

La configurazione è estremamente flessibile e può anche risultare complessa, ma per molti sistemi quella di default può essere sufficiente e, eventualmente, son già preimpostati in forma commentata opzioni di configurazione alternativa.
In generale il file principale è `/etc/rsyslog.conf`[10] che riporta al termine:

```
1 ....
2 # Include all config files in /etc/rsyslog.d/
3 #
4 $IncludeConfig /etc/rsyslog.d/*.conf
```

Esempio 24.10: *Spezzone del file `/etc/rsyslog.conf`*

e nella directory il file principale è `50-default.conf` in cui di seguito si riporta uno spezzone:

```
1 #
2 # First some standard log files.  Log by facility.
3 #
4 auth,authpriv.*                  /var/log/auth.log
5 *.*;auth,authpriv.none           -/var/log/syslog
6 #cron.*                          /var/log/cron.log
7 #daemon.*                        -/var/log/daemon.log
8 kern.*                           -/var/log/kern.log
```

[9] tra le quali moduli di input/output e nella comunicazione di rete il livello di trasporto può essere sia UDP sia TCP

[10] v. `man rsyslogd.conf`

```
 9 #lpr.*                        -/var/log/lpr.log
10 mail.*                       -/var/log/mail.log
11 #user.*                      -/var/log/user.log
```

Esempio 24.11: *Spezzone del file* `/etc/rsyslog.d/50-default.conf`

in cui molte linee sono commentate, ma suggeriscono eventuali personalizzazioni. Escludendo i commenti:

- la prima colonna è definita *selettore*, filtra un generico messaggio di log tramite *facility*`.`*priority*, mettendo il carattere `*` per indicare tutte le alternative[11];
- la seconda colonna è definita *action* e indica la destinazione dei messaggi di log, nel caso in esame indica tutti file localizzati nella directory `/var/log`.

Da notare il simbolo `-` prima della destinazione per disabilitare il *syncing* della risorsa: l'abilitazione del *syncing* degraderà le performance invece, procedendo tramite buffer delle operazioni di scrittura, la memorizzazione su storage avverrà in modalità asincrona e, ovviamente, le performance saranno migliori.

Selectors Per quanto riguarda le *facilities* e le *priorities*, la coppia è definita *selettore* e, come già indicato, sono separate dal carattere `.`, entrambe sono *case insensitive* e possono essere specificate nel corrispondente numero decimale (ma è sconsigliato).

Le keyword utilizzabili come facility sono le seguenti: `auth`, `authpriv`, `cron`, `daemon`, `kern`, `lpr`, `mail`, `mark`[12], `news`, `security`[13], `syslog`, `user`, `uucp`, da `local0` fino a `local7`.

Le keyword utilizzabili come priority ordinate secondo il livello di priorità discendente sono le seguenti: `debug`, `info`, `notice`, `warning`[14] `err` o equivalentemente `error`, `crit`, `alert`, `emerg` o equivalentemente `panic`.
Il comportamento di `syslogd` prevede che tutti i messaggi della priorità uguale o superiore a qualle indicata dalla keyword matchino la stessa. Il demone `rsyslogd` si comporta analogamente ma permette una flessibilità maggiore.

Il demone `rsyslogd` introduce alcune estensioni rispetto il precedente permettendo l'uso:

`*`[15] per il matching di qualsiasi facility o qualsiasi priority a seconda che sia messa prima o dopo il `.` di separazione;

è possibile utilizzare più facilities della medesima priority in un'unica espressione separando le stesse tra loro con il carattere `,`[16];

[11] cioè non filtrare quell'etichetta
[12] solo per uso interno, non dovrebbe essere utilizzato per uso applicativo
[13] equivalente a `auth` ma l'utilizzo è deprecato)
[14] equivalente `warn` ma l'uso è deprecato
[15] asterisco
[16] virgola

è possibile utilizzare più selettori per una singola azione separandoli tra loro con il carattere `;`[17]. Ovviamente il selettore successivo si potrà sovrapporre al precedente, ed utilizzando tale peculiarità si potranno escludere talune piorità dal pattern;

`none` come keyword aggiuntiva per le priority, per indicare che la linea non dovrà matchare per nessuna priority di quella facility.

Oltre le precedenti, la sintassi delle priorities è stata estesa poiché è possibile aggiungere i caratteri:

=[18], per specificare solo quel livello di priorità, quindi escludendo i livelli con maggior priorità;

![19], per ignorare tutte le piorità indicate (quindi dal livello espresso ai livelli con maggior priorità);

!=, è la composizione dei due precedenti, per escludere esclusivamente il livello espresso.

Actions Le azioni mostrate nell'***Esempio 24.11*** sono tutte di scrittura sincrona o asincrona su file locale. Una tipica alternativa è inoltrare i messaggi su server remoto.
Ad esempio:

```
1 *.*      @192.168.0.1
```

Esempio 24.12: *Spezzone di configurazione per inotrare i messaggi di log via* UDP

inoltrerà tutti i messaggi verso l'host con indirizzo `192.168.0.1` utilizzando il metodo di default, quindi via *UDP* alla porta di destinazione 514.
Lato collettore si dovrà attivare il modulo `imudp` che semplicemente porrà il server in ascolto per la ricezione di messaggi *syslog* via *UDP*, di default sulla porta 514.

In situazioni di carico elevato o ulteriori criticità, si può optare per modalità di trasporto alternative:

- protocollo *TCP*, permette una comunicazione più robusta del protocollo *UDP*, quindi messaggi di log saranno persi solo eccezionalmente. Per utilizzare tale modalità sarà necessario:
  - lato client preporre `@@`[20] prima dell'host di destinazione nel file di configurazione;
  - attivare il modulo `imtcp` lato collettore.

  Ad esempio:

```
1 *.*      @@192.168.0.1
```

Esempio 24.13: *Spezzone di configurazione per inotrare i messaggi di log via* TCP

[17] punto e virgola
[18] uguale
[19] punto esclamativo
[20] quindi due chiocciole al posto di una

- protocollo *REPL* su *TCP*, modello di comunicazione tra client e server in cui è da escludere perdita di messaggi, ma prevede:
  - attivare il modulo `imrelp` lato collettore;
  - attivare il modulo `omrelp` lato client;
  - preporre `:omrelp:`[21] all'indirizzo di rete dell'host ricevente.

Ad esempio:

```
1 *.*      :omrelp:192.168.0.1
```

Esempio 24.14: *Spezzone di configurazione per inotrare i messaggi di log via* REPL

24.3.5 `logrotate`

Il sistema *logrotate* ha il compito di gestire i log di sistema, in particolare si occuperà della *rotazione*, *compressione* e *rimozione* dei file prodotti.
Spesso si trascura la sua configurazione poiché tipicamente è sufficiente quella impostata di default dalla distribuzione, e talvolta installando pacchetti aggiuntivi i cui software generano log, essi compieranno nella *postconfigurazione* una propria configurazione da inserire nel `logrotate`, configurazione adeguata[22] della gestione dei propri log.
Diverso è l'approccio sui server i cui log potrebbero essere massicci, in cui sono da tener in considerazione due aspetti principali:

- dimensione dello storage, in particolare non è salutare riempire il disco[23];
- per file di log particolarmente grandi, la consultazione dei contenuti non sarà agevole.

Supponendo il *logrotate* sia già disponibile nel sistema, si potranno immediatamente verificare gli effetti della sua gestione digitando:

```
1  $ ls -l /var/log
2  ...
3  -rw-r----- 1 syslog adm 473303 giu 25 12:05 /var/log/syslog
4  -rw-r----- 1 syslog adm 761727 giu 25 00:00 /var/log/syslog.1
5  -rw-r----- 1 syslog adm  46704 giu 24 00:00 /var/log/syslog.2.gz
6  -rw-r----- 1 syslog adm 143240 giu 23 00:00 /var/log/syslog.3.gz
7  -rw-r----- 1 syslog adm  60436 giu 22 00:00 /var/log/syslog.4.gz
8  -rw-r----- 1 syslog adm  62484 giu 21 00:00 /var/log/syslog.5.gz
9  -rw-r----- 1 syslog adm  71620 giu 20 00:00 /var/log/syslog.6.gz
10 ...
```

Esempio 24.15: *Effetto immediato del* logrotate *tramite* `ls -l /var/log`

Qui sono mostrati solo i file `syslog*`, ma si può notare uno schema comune: periodicamente i log sono ruotati e compressi in accordo ad alcune direttive che saranno di seguito argomentate.

[21] quindi `:omrelp:` al posto di `@`
[22] almeno secondo l'autore
[23] se previdentemente la directory `/var/log` è su una partizione differente dalla `/`, allora i danni saranno più contenuti

Installazione Il file principale della configurazione è `/etc/logrotate.conf` il cui contenuto è breve e commentato cosicché sia di facile comprensione e modifica. Di seguito è riportata la direttiva finale:

```
1 ....
2 # packages drop log rotation information into this directory
3 include /etc/logrotate.d
```

Esempio 24.16: *Spezzone del file `/etc/logrotate.conf`*

Nella directory indicata i pacchetti installati inseriranno il proprio file contenente le direttive da applicare ai propri log prodotti.

Funzionamento Semplificando il demone `cron`[24] giornalmente eseguirà il comando `/usr/sbin/logrotate /etc/logrotate.conf`, e la direttiva appena mostrata indicherà che anche i file contenuti nella directory dovranno essere considerati parte del file di configurazione.

Lo schema generale prevede una rotazione dei log periodica. Ad esempio, si consideri un file di log di seguito:

```
1 mylog
```

Esempio 24.17: *Generico file di log*

prodotto da un qualche software e, dopo un periodo prestabilito che possiamo ipotizzare di una settimana, il `logrotate` lo ruoterà rinominandolo come di seguito:

```
1 mylog → mylog.1
```

Esempio 24.18: *Generico file di log ruotato*

Dopo il passaggio di due periodi (nell'ipotesi due settimane) si procederà ad una nuova rotazione:

```
1 mylog.1 → mylog.2
2 mylog → mylog.1
```

Esempio 24.19: *Generico file di log ruotato due volte*

in cui il file:

`mylog.1` era stato prodotto dalla prima rotazione;

`mylog` è il nuovo contenuto prodotto, che questa rotazione lo rinominerà in `mylog.1`.

Dopo il passaggio di tre periodi (nell'ipotesi tre settimane) si procederà ad una nuova rotazione:

```
1 mylog.2 → mylog.3
2 mylog.1 → mylog.2
3 mylog → mylog.1
```

Esempio 24.20: *Generico file di log ruotato due volte*

[24] tramite l'inserimento del file `/etc/cron.daily/logrotate`

in cui non si esplicheranno le trasformazioni effettuate. Già dall'introduzione iniziale si è anticipato che la trasformazione dei log può non essere una semplice rinomina di file, ma ad esempio compressione o addirittura cancellazione qualora non si voglia occupare troppo storage.

Configurazione Partendo dal file principale `/etc/logrotate.conf`, esso conterrà *direttive globali* che si applicheranno a tutti i file di log a meno che singoli file in `/etc/logrotate.d` utilizzeranno la medesima direttiva, ma sarà una *direttiva locale* cioè sovrascriverà il valore solo nel proprio contesto.

Si riesamina il file `/etc/logrotate.conf` di seguito, di facile lettura

```
1 # rotate log files weekly
2 weekly
3
4 # use the adm group by default, since this is the owning group
5 # of /var/log/syslog.
6 su root adm
7
8 # keep 4 weeks worth of backlogs
9 rotate 4
10
11 # create new (empty) log files after rotating old ones
12 create
13
14 # use date as a suffix of the rotated file
15 #dateext
16
17 # uncomment this if you want your log files compressed
18 #compress
19
20 # packages drop log rotation information into this directory
21 include /etc/logrotate.d
```

Esempio 24.21: *File `/etc/logrotate.conf`*

in cui le direttive visibili sono:

`weekly`
: per ruotare i log settimanalmente;

`su root adm`
: esegue la rotazione con l'user `root` e group `adm`. In caso si voglia modificare quanto espresso, da verificare che la coppia `user/group` abbia sufficienti privilegi per gestire le operazioni;

`rotate 4`
: mantiene le 4 rotazioni più recenti, quindi (se più di 4) saranno cancellate le più vecchie;

`create`
: dopo aver effettuato la rotazione, si procederà alla creazione del file di log vuoto;

`dateext`
: è commentato, aggiungerebbe la data di archiviazione in formato `YYYYMMDD` ai log vecchi;

`compress`
: è commentato, effettuerebbe la compressione dei log vecchi;

`include /etc/logrotate.d`
: già esaminata, include i file della directory specificata il cui contenuto sarà inglobato nella configurazione principale.

Alcuni file contenuti nella directory `/etc/logrotate.d`:

```
1 alternatives  bootlog  cups-daemon  mysql-server  rsyslog
2 apt           btmp     dpkg         nginx         unattended-upgrades
```

Esempio 24.22: *Alcuni file nella directory* `/etc/logrotate.d`

Ad esempio, `apt` è installato dal pacchetto `apt` ed è il package manager *APT*[25], `cups-daemon` è installato dal pacchetto `cups-daemon` ed è un demone per la stampa, `dpkg` è il *Debian package management system*[26].
In ciascun file dei precedenti saranno contenuti i percorsi dei log su cui applicare le direttive locali.
Per esempio, consultando tra questi il file `/etc/logrotate.d/dpkg` si otterrà il seguente contenuto:

```
1 /var/log/dpkg.log {
2         monthly
3         rotate 12
4         compress
5         delaycompress
6         missingok
7         notifempty
8         create 644 root root
9 }
```

Esempio 24.23: *File* `/etc/logrotate.d/dpkg`

in cui si noteranno:

- il percorso dei log prodotti;
- l'elenco delle direttive locali da applicare, sostituiranno il valore di quanto espresso dall'omonima globale nel contesto[27] in cui sono espresse.

Le direttive saranno esaminate nel prossimo paragrafo.

Direttive Di seguito si elencano le più comuni[28] direttive utilizzate.

`compress` o `nocompress`
: indica se effettuare o non effettuare (comporramento di default) la compressione[29] del file appena ruotato.

[25] v. 19.2.2 a pag. 258
[26] v. 16.8 a pag. 204
[27] definito da `{ ... }`
[28] per un elenco completo si consulti il manuale `man logrotate`
[29] di default sarà `gzip` con estensione `gz`, in alternativa si utilizzerà `compresscmd` per indicare il pathname completo del compressore alternativo, `compressext` qualora si voglia una estensione dei file compressi differente, `compressoptions` per utilizzare opzioni di compressione differenti dal default (per `il default è -6`)

Nella configurazione proposta nel file `logrotate.conf` la direttiva è commentata, quindi i file non saranno compressi a meno che la direttiva locale omonima non sarà espressa.

`create` o `nocreate`
: per creare (o non creare) un nuovo file di log subito dopo la rotazione, che avrà gli stessi attributi del file ruotato a meno che non sia specificato differentemente negli argomenti della direttiva. Ad esempio `create 660 root:adm` creerà il file con permessi di lettura e scrittura per l'user `root` e gruppo `adm`.
Nella configurazione proposta la direttiva è espressa, quindi la rotazione prevede la creazione di un nuovo file di log a meno che una direttiva locale omonima non sarà espressa.

`daily` o `weekly` o `monthly` o `yearly`
: per definire il periodi di rotazione, rispettivamente giornalmente (il default) o settimanalmente o mensilmente o annualmente. `logrotate` si attiverà una volta al giorno[30] e, confrontando il timestamp di creazione del file potenzialmente da ruotare con la data attuale, provvederà alla rotazione qualora sia passato il tempo espresso dal periodo.
Nella configurazione proposta la direttiva è `weekly` per cui i log saranno ruotati settimanalmente a meno che non ci sia una direttiva locale differente per taluni log.

`dateext` o `nodateext`
: il file ha di default l'estensione di un numero per i log vecchi, con `dateext` l'estensione sarà in formato[31] `YYYYMMDD`. L'opzione `nodateext` ha senso in contesto locale qualora la direttiva globale sia stata espressa.
Nella configurazione proposta la direttiva non è espressa, per cui saranno utilizzate l'estensione del numero a meno che non ci sia una direttiva locale differente.

`delaycompress` o `nodelaycompress`
: sono considerate solo se è definita `compress`, sono per comprimere i log ritardando (o non ritardando) di una rotazione. Di default è `nodelaycompress`, quindi se è definita `compress` la rotazione comprimerà il log immediatamente, con `delaycompress` i log ruotato rimarrebbe in *plain text* fino alla successiva rotazione.

`ifempty` o `notifempty`
: con `ifempty` (default) la rotazione sarà effettuata anche se il file di log è vuoto, con `notifempty` la rotazione sarà effettuta solo se il file di log non sarà vuoto.

`mail` o `nomail`
: per inviare (o non inviare) tramite email i log ruotati. In particolare con `mail` si dovrà indicare l'indirizzo al quale inviare il log e si potrà aggiungere una seconda direttiva `mailfirst` o `maillast` rispettivamente per inviare il log appena ruotato o il log che starà per essere cancellato (default).

`size`
: se utilizzatp dovrà essere seguito dalla dimensione[32] ruota il file solo se raggiunta

[30] si veda l'orario espresso in `/etc/crontab`

[31] personalizzabile con `dateformat`, ma attenzione all'ordinamento lessicale che ne verrebbe fuori poiché la rimozione dei vecchi log rispetta tale ordinamento

[32] ad esempio `200` oppure `200k` oppure `200M` oppure `200G` rispettivamente per *200 bytes*, *200 kilobytes*, *200 megabytes* e *200 gigabytes*

tale dimensione. La direttiva lavora in *or logico* con il periodo quindi per la rotazione è sufficiente una delle due condizioni. Per variazioni su tale comportamente, consulta il manuale per `maxsize`[33] e `minsize`[34].

`missingok` o `nomissingok`
di default è impostato `nomissingok` per cui, qualora il file di log non esista, `logrotate` terminerà con un messaggio di errore. Se utilizzato `missingok`, nel caso manchi il file di log, `logrotate` continuerà senza segnalazioni.

`postrotate/endscript`
successivamente alla rotazione dei log, talvolta è opportuno eseguire dei comandi. Per inserire una sequenza di comandi, l'apertura dovrà essere `postrotate` e la chiusura `endscript` e tale sequenza di comandi sarà eseguita da `/bin/sh`.
Alcuni esempi dalla guida:

```
/var/log/messages {
    rotate 5
    weekly
    postrotate
        /usr/bin/killall -HUP syslogd
    endscript
}

"/var/log/httpd/access.log" /var/log/httpd/error.log {
    rotate 5
    mail recipient@example.org
    size 100k
    sharedscripts
    postrotate
        /usr/bin/killall -HUP httpd
    endscript
}

/var/log/news/* {
    monthly
    rotate 2
    olddir /var/log/news/old
    missingok
    postrotate
        kill -HUP $(cat /var/run/inn.pid)
    endscript
    nocompress
}
```

Esempio 24.24: *Esempi di* `postrotate/endscript`

in cui:

`/usr/bin/killall -HUP syslogd`
invia il segnale `SIGHUP` al processo con nome `syslogd`;

`/usr/bin/killall -HUP httpd`
invia il segnale `SIGHUP` al processo con nome `httpd`;

[33] non sarà modificato il timestamp del ruotato, quindi la rotazione prevista dal periodo indicato sarà comunque effettuata

[34] la direttiva è in *and logico* con il periodo, quindi affinché la rotazione sia effettuata dovranno essere rispettate entrambe le condizioni

`kill -HUP $(cat /var/run/inn.pid)`
invia il segnale `SIGHUP` al processo il cui *pid* è contenuto nel file indicato.

In particolare il segnale `SIGHUP` sarà intercettato dai relativi processi indicati, e ciascuno procederà a chiudere i file di log attivi e riaprirli[35].
Si può notare nel secondo blocco l'uso di `sharedscripts`: questa sezione si riferisce a due file di log, `/var/log/httpd/access.log` e `/var/log/httpd/error.log`. Inserendo `sharedscripts` la sequenza definita in `postrotate` sarà eseguita al termine della rotazione di tutti i log, quindi non sarà eseguita per ogni log ruotato.

24.4 systemd-journald

Oltre al tradizionale `rsyslogd`, la suite `systemd`[36] ha introdotto un nuovo strumento per la gestione dei log denominato *systemd-journald*[37].
Tale componente della suite ha il compito di collezionare dati di log provenienti da diverse sorgenti:

- messaggi dal *kernel ring buffer*, via[38] `/dev/kmsg`;
- i tradizionali messaggi *syslog* tramite la system call `syslog`;
- messaggi via la nativa *API*;
- lo *standard output* e *standard error* di tutti i servizi gestiti via *systemd*;
- gli *audit record*[39] generati dal *kernel audit system*.

Successivamente li memorizza e indicizza, successivamente li rende disponibili all'utenza. La differenza sostanziale tra i file gestiti da `rsyslog` e `journald` è nella gestione e storicizzazione, in particolare i primi sono storicizzati gerarchicamente in `/var/log` in file testuali, gli ultimi sono memorizzati in file binari localizzati a seconda della preferenza definita in configurazione[40]. Le alternative principali sono:

volatile, gerarchicamente nella directory `/run/log/journal/`;

persistente, storicizzati gerarchicamente nella directory `/var/log/journal/`.

Per quanto riguarda la classificazione dei messaggi, rimane inalterata la trattazione precedente sulle *facilities*[41] e *priorities*[42].

---

[35] quest'attività è necessaria poiché, anche se le risorse sono state ruotate quindi sono state rinominate, eventuali processi che le usavano continueranno a puntare ciascuna alle medesime risorse. La strategia utilizzata è chiudere le risorse correnti e riaprile accedendo sempre nominalmente, ma ora alle nuove risorse

[36] v. 23.5.1 a pag. 363

[37] v. `man systemd-journald`

[38] v. 24.6.1

[39] sono informazioni per tener traccia di eventi rilevanti riguardanti la sicurezza del sistema

[40] v. `man journald.conf`, sezione `Storage`

[41] v. ***Tabella 24.4***

[42] v. ***Tabella 24.5***

24.4.1 Report

Di seguito saranno mostrati alcuni esempi che mostrano la potenza dello strumento e l'estrema flessibilità nella reportistica prodotta, ma per una documentazione esaustiva si consulti il manuale relativo[43].

Reportistica base

Per la consultazione dei file binari prodotti è necessario utilizzare il comando `journalctl` come di seguito:

```
$ journalctl
-- Logs begin at Thu 2023-08-31 09:10:29 CEST, end at Wed 2024-06-19 12:55:29 CEST. --
ago 31 09:10:29 nb-m systemd[4308]: Stopped target Bluetooth.
ago 31 09:10:29 nb-m systemd[1]: Stopped target Bluetooth.
ago 31 09:10:29 nb-m logrotate[685933]: [52B blob data]
...
ago 31 09:10:29 nb-m kernel: usb 1-6: new full-speed USB device number 64 using ...
ago 31 09:10:29 nb-m kernel: Generic FE-GE Realtek PHY r8169-0-200:00: attached PHY dri...
...
```

Esempio 24.25: *`journalctl`*

Il comando di default mostra i log paginati ma in ordine cronologico, per cui spesso sarà utile ordinarli dal più recente, quindi sarà utile il flag `-r` o equivalentemente `--reverse`:

```
$ journalctl -r
-- Logs begin at Thu 2023-08-31 09:10:29 CEST, end at Thu 2024-06-20 10:02:08 CEST. --
giu 20 10:02:08 nb-m systemd[12402]: Started Tracker metadata database store and lookup manager.
giu 20 10:02:08 nb-m dbus-daemon[12416]: [session uid=1000 pid=12416] Successfully activated
      service 'org.freedesktop>
giu 20 10:02:08 nb-m systemd[12402]: Starting Tracker metadata database store and lookup manager...
giu 20 10:02:08 nb-m dbus-daemon[12416]: [session uid=1000 pid=12416] Activating via systemd:
      service name='org.freed>
giu 20 10:01:33 nb-m rtkit-daemon[4227]: Supervising 0 threads of 0 processes of 0 users.
giu 20 10:01:33 nb-m rtkit-daemon[4227]: Supervising 0 threads of 0 processes of 0 users.
...
```

Esempio 24.26: *`journalctl -r`*

Alternativamente è possibile richiedere le 10 più recenti linee direttamente con il flag `-n` o equivalentemente `--lines`, con opzionalmente aggiungere il numero delle linee che si vogliono visualizzare oppure `--all`:

```
$ journalctl -n 5
-- Logs begin at Thu 2023-08-31 09:10:29 CEST, end at Thu 2024-06-20 10:02:08 CEST. --
giu 20 10:02:08 nb-m systemd[12402]: Started Tracker metadata database store and lookup manager.
giu 20 10:02:08 nb-m dbus-daemon[12416]: [session uid=1000 pid=12416] Successfully activated
      service 'org.freedesktop>
giu 20 10:02:08 nb-m systemd[12402]: Starting Tracker metadata database store and lookup manager...
giu 20 10:02:08 nb-m dbus-daemon[12416]: [session uid=1000 pid=12416] Activating via systemd:
      service name='org.freed>
giu 20 10:01:33 nb-m rtkit-daemon[4227]: Supervising 0 threads of 0 processes of 0 users.
```

Esempio 24.27: *`journalctl -n 5`*

Molto utile per il troubleshooting in tempo reale[44] avere anche il log in tempo reale tramite il flag `-f` o equivalentemente `--follow`:

```
$ journalctl -n 0 -f
-- Logs begin at Thu 2023-08-31 09:10:29 CEST. --
```

Esempio 24.28: *`journalctl -n 5`*

[43] `man journalctl` oppure il meno verboso `journalctl --help`

[44] sarà utilizzato in24.5 a pag. 391

in cui resterà in attesa[45] che appaia nuovo log da mostrare.

Per cercare tra i messaggi filtrando tra il contenuto del messaggio attraverso un'espressione regolare[46] si utilizza il flag `-g` o equivalentemente `--grep`, con aggiunta del flag `--case-sensitive=`*`bool`* per forzare il matching *case-sensitive* o *case-insensitive* sostituendo a *bool* il valore 1 o 0[47]:

```
1 $ journalctl -b --grep password --case-sensitive=1
2 mag 09 09:01:22 nb-m mysqld[3907]: 2024-05-09  9:01:22 220 [Warning] Access denied for user ...
3 giu 04 23:10:13 nb-m sudo[2301287]:  ms : 3 incorrect password attempts ; TTY=pts/12 ; PWD=/home/ms
        ; USER=root ; COMMAND=XXXXXXXXX
4 ...
```

Esempio 24.29: *`journalctl --grep`*

in cui l'autore ha sovrascritto il reale comando con la stringa `XXXXXXXXX` per anonimizzare il messaggio.

Report per boot

Molto utile il report ottenuto tramite il flag `--list-boots`:

```
1  $ journalctl --list-boots
2  -22 24dcc324323546559b536f9f0c911a62 Thu 2023-08-31 09:10:29 CEST-Fri 2023-09-08 10:07:48 CEST
3  -21 15316159ff2e45a98b7c0a038af6044f Fri 2023-09-08 10:08:56 CEST-Wed 2023-09-13 11:13:46 CEST
4  -20 daa06170c3cc46c5946eab55942e9e57 Wed 2023-09-13 11:16:06 CEST-Mon 2023-09-18 17:54:49 CEST
5  -19 2366a896d85948d3bc441f627a04c7cd Mon 2023-09-18 17:56:06 CEST-Wed 2023-09-20 16:58:54 CEST
6  ...
7  -5 542b04a994254e958c28b6fe806c62c9 Fri 2024-02-16 09:10:53 CET-Wed 2024-02-21 18:02:44 CET
8   -4 3503649767804cb597f33fc872e7ffaa Wed 2024-02-21 18:03:44 CET-Tue 2024-03-12 09:16:22 CET
9   -3 54131fad22bb4c8f91d20658336b5f0c Tue 2024-03-12 09:17:01 CET-Sun 2024-03-24 17:07:21 CET
10  -2 f25952ade4f3424ab84826618581725f Sun 2024-03-24 17:08:01 CET-Mon 2024-04-01 16:55:26 CEST
11  -1 baa343495be84ae281eb177a7c4be446 Tue 2024-04-02 09:08:11 CEST-Tue 2024-04-30 12:06:44 CEST
12   0 13574606af1d46bc97ba41662e6b8a7c Tue 2024-04-30 12:08:38 CEST-Thu 2024-06-20 11:18:17 CEST
```

Esempio 24.30: *`journalctl --list-boots`*

Possiamo ottenere i log solo di un particolare boot utilizzando l'opzione `-b` o equivalente `--boot` con opzionalmente l'aggiunta dell'*id*. Quindi i log dell'ultimo boot:

```
1  $ journalctl -b
2  apr 30 12:08:38 nb-m kernel: microcode: microcode updated early to revision 0x26, date = 2019-11-12
3  apr 30 12:08:38 nb-m kernel: Linux version 5.15.0-105-generic (buildd@lcy02-amd64-054) (gcc (Ubuntu
         9.4.0-1ubuntu1~20.04.2) 9.4.0, GNU ld (GNU B>
4  apr 30 12:08:38 nb-m kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-105-generic root=UUID=
        c6733fca-5211-4b85-ba28-a89d0da88bbf ro
5  apr 30 12:08:38 nb-m kernel: KERNEL supported cpus:
6  apr 30 12:08:38 nb-m kernel:   Intel GenuineIntel
7  apr 30 12:08:38 nb-m kernel:   AMD AuthenticAMD
8  apr 30 12:08:38 nb-m kernel:   Hygon HygonGenuine
9  apr 30 12:08:38 nb-m kernel:   Centaur CentaurHauls
10 apr 30 12:08:38 nb-m kernel:   zhaoxin   Shanghai
11 apr 30 12:08:38 nb-m kernel: BIOS-provided physical RAM map:
12 apr 30 12:08:38 nb-m kernel: BIOS-e820: [mem 0x0000000000000000-0x000000000057fff] usable
13 apr 30 12:08:38 nb-m kernel: BIOS-e820: [mem 0x0000000000058000-0x000000000058fff] reserved
14 ...
```

Esempio 24.31: *`journalctl --list-boots`*

oppure esplicitando l'*id* del boot con gli *id* mostrati nell'***Esempio 24.30***, quindi ottenere i log del precedente boot con:

[45] analogo a `tail -f` su file che attenderà nuovo contenuto immesso nel file che il `tail` mostrerà in output

[46] sarà una *Perl-compatible regular expression*, v. `man pcre2pattern`

[47] oppure le stringhe *true* o *false*

```
1  $ journalctl -b -1
2  apr 02 09:08:11 nb-m kernel: microcode: microcode updated early to revision 0x26, date = 2019-11-12
3  apr 02 09:08:11 nb-m kernel: Linux version 5.15.0-101-generic (buildd@lcy02-amd64-031) (gcc (Ubuntu
         9.4.0-1ubuntu1~20.04.2) 9.4.0, GNU ld (GNU B>
4  apr 02 09:08:11 nb-m kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-101-generic root=UUID=
       c6733fca-5211-4b85-ba28-a89d0da88bbf ro
5  apr 02 09:08:11 nb-m kernel: KERNEL supported cpus:
6  apr 02 09:08:11 nb-m kernel:   Intel GenuineIntel
7  apr 02 09:08:11 nb-m kernel:   AMD AuthenticAMD
8  apr 02 09:08:11 nb-m kernel:   Hygon HygonGenuine
9  apr 02 09:08:11 nb-m kernel:   Centaur CentaurHauls
10 apr 02 09:08:11 nb-m kernel:   zhaoxin   Shanghai
11 apr 02 09:08:11 nb-m kernel: BIOS-provided physical RAM map:
12 apr 02 09:08:11 nb-m kernel: BIOS-e820: [mem 0x0000000000000000-0x0000000000057fff] usable
13 apr 02 09:08:11 nb-m kernel: BIOS-e820: [mem 0x0000000000058000-0x0000000000058fff] reserved
14 ...
```

Esempio 24.32: *`journalctl --list-boots`*

Report per finestra temporale

Si possono utilizzare i flag `-S` o equivalente `--since` per la data di inizio e i flag `-U` o equivalente `--until` per la data di fine, con la data espressa in formato `YYYY-MM-DD hh:mm:ss`[48].
Ad esempio:

```
1  $ journalctl --since "2024-06-18 17:15:00"
2  giu 18 17:15:01 nb-m CRON[3014410]: pam_unix(cron:session): session opened for user root by (uid=0)
3  giu 18 17:15:01 nb-m CRON[3014411]: (root) CMD (command -v debian-sa1 > /dev/null && debian-sa1 1
       1)
4  giu 18 17:15:01 nb-m CRON[3014410]: pam_unix(cron:session): session closed for user root
5  giu 18 17:17:01 nb-m CRON[3014444]: pam_unix(cron:session): session opened for user root by (uid=0)
6  giu 18 17:17:01 nb-m CRON[3014445]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
7  giu 18 17:17:01 nb-m CRON[3014444]: pam_unix(cron:session): session closed for user root
8  giu 18 17:18:16 nb-m dnsmasq-dhcp[3737]: DHCPREQUEST(lxcbr0) 10.0.3.204 a2:ec:0c:61:85:e2
9  giu 18 17:18:16 nb-m dnsmasq-dhcp[3737]: Viene ignorato il dominio 04 per il nome host DHCP mylxc-
       ubuntu22
10 giu 18 17:18:16 nb-m dnsmasq-dhcp[3737]: DHCPACK(lxcbr0) 10.0.3.204 a2:ec:0c:61:85:e2 mylxc-
       ubuntu22
11 giu 18 17:19:05 nb-m rtkit-daemon[4227]: Supervising 0 threads of 0 processes of 0 users.
12 giu 18 17:19:05 nb-m rtkit-daemon[4227]: Supervising 0 threads of 0 processes of 0 users.
13 giu 18 17:21:35 nb-m rtkit-daemon[4227]: Supervising 0 threads of 0 processes of 0 users.
14 giu 18 17:21:35 nb-m rtkit-daemon[4227]: Supervising 0 threads of 0 processes of 0 users.
15 ...
```

Esempio 24.33: *`journalctl --since`*

Èchiaramente possibile esprimere solamente la fine della finestra, oppure un filtro con sia inizia sia fine della finestra temporale, ad esempio restringendo l'output appena mostrato

```
1  $ journalctl --since "2024-06-18 17:18:00" --until "2024-06-18 17:20:21"
2  giu 18 17:18:16 nb-m dnsmasq-dhcp[3737]: DHCPREQUEST(lxcbr0) 10.0.3.204 a2:ec:0c:61:85:e2
3  giu 18 17:18:16 nb-m dnsmasq-dhcp[3737]: Viene ignorato il dominio 04 per il nome host DHCP mylxc-
       ubuntu22
4  giu 18 17:18:16 nb-m dnsmasq-dhcp[3737]: DHCPACK(lxcbr0) 10.0.3.204 a2:ec:0c:61:85:e2 mylxc-
       ubuntu22
5  giu 18 17:19:05 nb-m rtkit-daemon[4227]: Supervising 0 threads of 0 processes of 0 users.
6  giu 18 17:19:05 nb-m rtkit-daemon[4227]: Supervising 0 threads of 0 processes of 0 users.
```

Esempio 24.34: *`journalctl --since --until`*

Report per unit

Per filtrare una particolare unit, si utilizza il flag `-u` o equivalentemente `--unit` con il nome dell'unità di interesse[49]:

[48] oppure `YYYY-MM-DD` in cui il filtro assumerà che la stringa `hh:mm:ss` sia `00:00:00`
[49] per ottenere la lista si digita `systemctl list-units`

```
$ journalctl -r -u ssh.service
giu 12 23:11:05 nb-m sshd[2747049]: Connection closed by authenticating user mariano 192.168.1.121
      port 48108 [pre>
giu 11 00:06:17 nb-m sshd[2588076]: pam_unix(sshd:session): session opened for user mariano by (uid
      =0)
giu 11 00:06:17 nb-m sshd[2588076]: Accepted keyboard-interactive/pam for mariano from
      192.168.1.121 port 57882 ss>
giu 07 23:15:42 nb-m sshd[2435207]: pam_unix(sshd:session): session opened for user mariano by (uid
      =0)
giu 07 23:15:42 nb-m sshd[2435207]: Accepted keyboard-interactive/pam for mariano from
      192.168.1.121 port 52802 ss>
giu 04 23:09:08 nb-m sshd[2300649]: pam_unix(sshd:session): session opened for user mariano by (uid
      =0)
giu 04 23:09:08 nb-m sshd[2300649]: Accepted keyboard-interactive/pam for mariano from
      192.168.1.121 port 35702 ss>
mag 30 23:47:36 nb-m sshd[2052163]: Connection closed by authenticating user mariano 192.168.1.121
      port 40642 [pre>
mag 30 23:47:35 nb-m sshd[2052163]: error: PAM: Authentication failure for mariano from
      192.168.1.121
mag 30 23:47:33 nb-m sshd[2052165]: pam_unix(sshd:auth): authentication failure; logname= uid=0
      euid=0 tty=ssh rus>
mag 30 23:07:30 nb-m sshd[2050373]: pam_unix(sshd:session): session opened for user mariano by (uid
      =0)
...
```

Esempio 24.35: `journalctl -r -u ssh`

Report per *pid* o *user*

Per filtrare un particolare *pid*, si utilizza l'espressione `_PID=`*n*, con *n* il *pid*. Ad esempio:

```
$ journalctl -b _PID=1
apr 30 12:08:38 nb-m systemd[1]: Mounted Kernel Trace File System.
apr 30 12:08:38 nb-m systemd[1]: Finished Availability of block devices.
apr 30 12:08:38 nb-m systemd[1]: Finished Set the console keyboard layout.
apr 30 12:08:38 nb-m systemd[1]: Finished Create list of static device nodes fo>
apr 30 12:08:38 nb-m systemd[1]: Finished Monitoring of LVM2 mirrors, snapshots>
apr 30 12:08:38 nb-m systemd[1]: modprobe@chromeos_pstore.service: Succeeded.
apr 30 12:08:38 nb-m systemd[1]: Finished Load Kernel Module chromeos_pstore.
...
```

Esempio 24.36: `journalctl -b _PID=1`

in cui il log è riportato del corrente boot (`-b`) e del *pid* 1[50].

È possibile filtrare per utente specificando il parametro `_UID=`*n*, con *n* l'id dell'utente. Ad esempio:

```
$ journalctl -b _UID=id -u ms`- Logs begin at Thu 2023-08-31 09:10:29 CEST, end at Thu 2024-06-20
      12:26:10 C>apr 30 12:10:21 nb-m systemd[12402]:  Started Pending report trigger for Ubuntu
      >apr 30 12:10:21 nb-m systemd[12402]:  Reached target Paths.apr 30 12:10:21 nb-m
      systemd[12402]:  Reached target Timers.apr 30 12:10:21 nb-m systemd[12402]:  Starting D-Bus
      User Message Bus Socket.apr 30 12:10:21 nb-m systemd[12402]:  Listening on GnuPG network
      certificate man>apr 30 12:10:21 nb-m systemd[12402]:  Listening on GnuPG cryptographic agent
      and>apr 30 12:10:21 nb-m systemd[12402]:  Listening on GnuPG cryptographic agent and>apr 30
      12:10:21 nb-m systemd[12402]:  Listening on GnuPG cryptographic agent (ss>apr 30 12:10:21 nb-m
      systemd[12402]:  Listening on GnuPG cryptographic agent and>apr 30 12:10:21 nb-m
      systemd[12402]:  Listening on debconf communication socket.apr 30 12:10:21 nb-m
      systemd[12402]:  Listening on Sound System.apr 30 12:10:21 nb-m systemd[12402]:  Listening on
      REST API socket for snapd use>apr 30 12:10:21 nb-m systemd[12402]:  Listening on D-Bus User
      Message Bus Socket.apr 30 12:10:21 nb-m systemd[12402]:  Reached target Sockets.apr 30
      12:10:21 nb-m systemd[12402]:  Reached target Basic System.apr 30 12:10:21 nb-m
      systemd[12402]:  Starting Sound Service...apr 30 12:10:22 nb-m systemd[12402]:  Started D-Bus
      User Message Bus.apr 30 12:10:22 nb-m dbus-daemon[12416]:  [session uid=1000 pid=12416]
      AppArmor >apr 30 12:10:22 nb-m systemd[12402]:  Started Sound Service....apr 30 12:10:22 nb-m
      systemd[12402]:  Reached target Main User Target.apr 30 12:10:22 nb-m systemd[12402]:  Startup
      finished in 479ms....apr 30 12:10:23 nb-m systemd[12402]:  Started Path trigger for Avahi
      .local doma>apr 30 12:10:23 nb-m systemd[12402]:  Condition check resulted in Path trigger
      f>apr 30 12:10:23 nb-m systemd[12402]:  Started Path trigger for new release of Ub>apr 30
      12:10:23 nb-m systemd[12402]:  Condition check resulted in GNOME Initial >apr 30 12:10:23 nb-m
      systemd[12402]:  Starting Start gnome-keyring as SSH agent.>apr 30 12:10:23 nb-m
      systemd[12402]:  Starting Start gnome-keyring for the Secre>apr 30 12:10:23 nb-m
      systemd[12402]:  Starting Monitor Session leader for GNOME >...apr 30 12:10:23 nb-m sh[12465]:
      dbus-update-activation-environment:  setting SSH>apr 30 12:10:23 nb-m sh[12465]:
      dbus-update-activation-environment:  setting SSH>...apr 30 12:10:23 nb-m systemd[12402]:
      Started OpenSSH Agent....apr 30 12:10:23 nb-m dbus-daemon[12416]:  [session uid=1000
      pid=12416] Activatin>apr 30 12:10:23 nb-m systemd[12402]:  Starting Virtual filesystem
      service...apr 30 12:10:23 nb-m dbus-daemon[12416]:  [session uid=1000 pid=12416] Successfu>apr
      30 12:10:24 nb-m gnome-shell[12487]:  Some code accessed the property 'Crede>...apr 30
      12:10:24 nb-m gnome-shell[12487]:  Will monitor session 3apr 30 12:10:24 nb-m
      dbus-daemon[12416]:  [session uid=1000 pid=12416] Activatin>apr 30 12:10:24 nb-m
      dbus-daemon[12416]:  [session uid=1000 pid=12416] Successfu>apr 30 12:10:25 nb-m
      dbus-daemon[12416]:  [session uid=1000 pid=12416] Activatin>apr 30 12:10:25 nb-m
```

[50] processo `systemd`

```
systemd[12402]:  Starting sandboxed app permission store...apr 30 12:10:25 nb-m
dbus-daemon[12416]:  [session uid=1000 pid=12416] Activatin>apr 30 12:10:25 nb-m
dbus-daemon[12416]:  [session uid=1000 pid=12416] Successfu>...
```

Esempio 24.37: `journalctl -b _UID=1`

in cui nella digitazione è stata utilizzata l'interpolazione della bash tramite il comando tra i *backtick*.

Report per *facility* e *priority*

In accordo con le facilities e le piorities fin qui discusse (v. ***Tabella 24.4*** e ***Tabella 24.5***), possono essere specificate utilizzando rispettivamente il flag `--facility` e `-priority`. Per ottenere l'elenco delle *facilities* si digiterà:

```
$ journalctl --facility=help
Available facilities:
kern
user
mail
daemon
auth
syslog
lpr
news
uucp
cron
authpriv
ftp
12
13
14
15
local0
local1
local2
local3
local4
local5
local6
local7
```

Esempio 24.38: `journalctl --facility=help`

Per il filtro `facility` è possibile esprimere una lista in cui le facility dovranno essere separate da una virgola.

Per le *priorities* utilizzabili è utile consultare il manuale, ma per semplicità si fornirà di seguito la lista:

```
emerg   0
alert   1
crit    2
err     3
warning 4
notice  5
info    6
debug   7
```

Esempio 24.39: *Lista delle* priorities *utilizzabili con* `journalctl`

in cui sarà possibile utilizzare indifferentemente la stringa o il numero associato al livello.

Un esempio che utilizzi contemporaneamente entrambi i filtri è il seguente:

```
$ journalctl --facility=daemon --priority=warning
...
ago 31 09:10:29 nb-m systemd[1]: logrotate.service: Failed with result 'exit-code'.
...
set 01 14:37:47 nb-m mysqld[3810]: 2023-09-01 14:37:47 283 [Warning] Access denied for user...
...
```

Esempio 24.40: *journalctl --facility --priority*

Per il filtro sulle priorità è da ricordare che è possibile sia utilizzare una singola priorità sia un range[51] con l'espressione `da..a`. Nel caso ci si esprima con una singola priorità saranno mostrati anche i messaggi con priorità maggiore[52].
Ad esempio:

```
1 $ journalctl --priority=notice | wc -l
2 236544
3 $ journalctl --priority=notice..notice | wc -l
4 124691
```

Esempio 24.41: *journalctl --priority* range

in cui nel boot corrente ci sono 236544 messaggi con priorità da `notice`, 124691 con priorità uguale a `notice`.

Provando l'esempio appena proposto (ma l'effetto è anche visibile per gli esempi precedenti), inviando l'output su `tty`, le linee saranno colorate in accordo alla priorità, in particolare:

- dal livello `error` o priorità maggiore la linea sarà colorata di rosso;
- dal livello `notice` o priorità maggiore[53] la linea sarà evidenziata;
- le linee di livello `debug` sono colorate di *grigio chiaro*;
- il resto delle linee saranno mostrate normalmente.

Report con formattazione dell'output

La formattazione dell'output si controlla attivando il flag `-o=`*`formattazione`* o equivalentemente `--output=`*`formattazione`* sostituendo alla stringa una delle seguenti[54]:

`short`
: è la formattazione di default, per esempio `apr 30 12:08:38`, meno formale;

`short-full`
: molto simile al precedente ma formatta il timestamp come `Tue 2024-04-30 12:08:38 CEST`, quindi il giorno e l'orario nel formato `YYYY-MM-DD hh:mm:ss` aggiungendo il fuso orario;

`short-iso`
: molto simile al precedente ma formatta il timestamp come `2024-04-30T12:08:38+0200`, quindi il timestamp in *ISO 8601*;

`short-iso-precise`
: molto simile al precedente ma formatta il timestamp come `2024-04-30T12:08:38.845829+0200`, quindi aggiunge i microsecondi;

[51] in questo caso entrambi gli estremi saranno inclusi nei valori del filtro
[52] quindi il filtro è da considerare *dalla priorità* espressa
[53] ovviamente da escludere quelle del punto precedente
[54] per un elenco esaustivo si consulti il `man journalctl` sezione `-o, --output=`

`short-monotonic`
: simile al precedente ma al posto del timestamp è inserito un differenziale temporale come `[ 0.000000]`;

`1714471718.845410`
: simile al precedente ma al posto del timestamp è inserito lo *UNIX time*[55] in microsecondi;

`verbose`
: estremamente verboso, mostra la struttura complessiva, quindi tutti i campi di log;

`json`
: formatta il messaggio in oggetti JSON separati tra essi con un *newline*;

`json-pretty`
: simile al precedente, ma la struttura JSON è meno densa, quindi più leggibile.

Per alcuni esempi:

```
$ journalctl -b -o short-iso
2024-04-30T12:08:38+0200 nb-m kernel: microcode: microcode updated early to revision 0x26, date = 2019-11-12
2024-04-30T12:08:38+0200 nb-m kernel: Linux version 5.15.0-105-generic (buildd@lcy02-amd64-054) (gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0, GN>
2024-04-30T12:08:38+0200 nb-m kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-105-generic root=UUID=c6733fca-5211-4b85-ba28-a89d0da88bbf>
2024-04-30T12:08:38+0200 nb-m kernel: KERNEL supported cpus:
2024-04-30T12:08:38+0200 nb-m kernel:   Intel GenuineIntel
2024-04-30T12:08:38+0200 nb-m kernel:   AMD AuthenticAMD
2024-04-30T12:08:38+0200 nb-m kernel:   Hygon HygonGenuine
2024-04-30T12:08:38+0200 nb-m kernel:   Centaur CentaurHauls
2024-04-30T12:08:38+0200 nb-m kernel:   zhaoxin   Shanghai
2024-04-30T12:08:38+0200 nb-m kernel: BIOS-provided physical RAM map:
...
$ journalctl -b -u nginx -o json-pretty
{
        "PRIORITY" : "6",
        "CODE_LINE" : "574",
        "_CMDLINE" : "/sbin/init",
        "_TRANSPORT" : "journal",
        "_CAP_EFFECTIVE" : "1fffffffff",
        "_SOURCE_REALTIME_TIMESTAMP" : "1714658199644714",
        "MESSAGE_ID" : "7d4958e842da4a758f6c1cdc7b36dcc5",
        "__REALTIME_TIMESTAMP" : "1714658199644744",
        "_UID" : "0",
        "_SYSTEMD_CGROUP" : "/init.scope",
        "SYSLOG_IDENTIFIER" : "systemd",
        "_BOOT_ID" : "13574606af1d46bc97ba41662e6b8a7c",
        "_COMM" : "systemd",
        "_MACHINE_ID" : "9cf3d5e6f59046549e378b163d806c7d",
        "_SYSTEMD_UNIT" : "init.scope",
        "_GID" : "0",
        "_EXE" : "/usr/lib/systemd/systemd",
        "_SYSTEMD_SLICE" : "-.slice",
        "CODE_FUNC" : "job_log_begin_status_message",
        "_SELINUX_CONTEXT" : "unconfined\n",
        "_PID" : "1",
        "__MONOTONIC_TIMESTAMP" : "186488489606",
        "INVOCATION_ID" : "b3a43900d4e840e69dfd90902de50b93",
        "__CURSOR" : "s=07060a9b9aec487587872dea81b599d7;i=3e66d;b=13574606af1d4>
        "SYSLOG_FACILITY" : "3",
        "CODE_FILE" : "src/core/job.c",
        "UNIT" : "nginx.service",
        "JOB_ID" : "21964",
        "_HOSTNAME" : "nb-m",
        "MESSAGE" : "Starting A high performance web server and a reverse proxy >
        "JOB_TYPE" : "start"
}
...
```

Esempio 24.42: *`journalctl -b -o short-iso` e `journalctl -o json-pretty`*

[55] secondi passati dal `1° gennaio 1970 UTC`

in cui il primo output mostra il timestamp in *ISO 8601*, il secondo una sola entry in *JSON*.

24.4.2 Manutenzione dello spazio

Una questione da affrontare con i log è lo spazio necessario per l'immagazzinamento. In particolare, senza nessuna manutenzione, con il tempo l'accumulo dei log può essere eccessivo.
Un primo comando utile è il seguente;

```
1 $ journalctl --disk-usage
2 Archived and active journals take up 4.0G in the file system.
```

Esempio 24.43: *`journalctl --disk-usage` e `journalctl -o json-pretty`*

il quale riporta l'occupazione su storage.

Per liberarci dei log in eccesso, le due maggiori opzioni sono:

`--vacuum-size=BYTE`
: riduce lo spazio su disco al limite indicato;

`--vacuum-time=TIME`
: rimuove i *journal file* più vecchi del tempo specificato.

Ad esempio:

```
1 $ sudo journalctl --vacuum-size=1G
2 ...
3 Vacuuming done, freed 0B of archived journals from /var/log/journal.
4 Vacuuming done, freed 0B of archived journals from /run/log/journal.
5 Deleted archived journal /var/log/journal/9cf3d5e6f59046549e378b163d806c7d
      /system@000604d47d02c6d5-78ea1159164bbfe7.journal~ (48.0M).
6 ...
7 Vacuuming done, freed 3.0G of archived journals from /var/log/journal/9
      cf3d5e6f59046549e378b163d806c7d.
8 -- oppure --
9 $ journalctl --vacuum-time=1years
10 ...
```

Esempio 24.44: *`journalctl --vacuum-size` e `journalctl --vacuum-time`*

Più robusta l'alternativa di settare i limiti su file di configurazione[56], in particolare le principali opzioni sono:

`SystemMaxUse`
: specifica la massima dimensione utilizzabile su storage;

`SystemKeepFree`
: specifica quanto storage dovrà almeno rimanere libero.

I valori specificati dovranno essere in byte o multipli utilizzando i simboli `K`, `M`, `G`, `T`, `P` e `E` per indicare un moltiplicatore rispettivamente di 1024, 1024^2, 1024^3, 1024^4, 1024^5 e 1024^6.

[56] `/etc/systemd/journald.conf`, v. `man journald.conf`

24.5 Logger

Solitamente si utilizza una via programmatica per inviare messaggi al syslog ed è quasi sempre la via da percorrere. In ogni caso di seguito un esempio con il software `logger` per inviare un messaggio al syslog e recepirlo sia i due sistemi di logging presenti su questo sistema: `rsyslogd` e `systemd-journald`.

Per verificare assieme l'invio, si aprano tre terminali:

terminale 0: è il terminale dal quale invieremo istruzioni;

terminale 1: digitare `tail -f /var/log/syslog` e premere Enter cosicché su tale terminale saranno mostrati i log ricevuti da `rsyslogd`

terminale 2: digitare `journalctl -f` e premere Enter cosicché su tale terminale saranno mostrati i log ricevuti da `systemd-journald`;

Il consiglio è battere un paio di Enter su entrambi terminali per spaziare verticalmente eventuali nuovi messaggi.
Sul **terminale 0** digitare:

```
1 $ logger -p syslog.info 'Hello world!!'
```

Esempio 24.45: *Invio di un messaggio di log tramite* `logger`

e, dopo aver premuto Enter su questo terminale, sarà inviato al sistema di logging il messaggio, in particolare specificando `-p syslog.info`[57] sarà utilizzata la priority `info` e la facility `syslog`. In output su entrambi i terminali avremo[58] ricevuto il messaggio.

Sul **terminale 1** si digiti Ctrl + C per uscire dal `tail`, e dal **terminale 0** aprire il file `/etc/rsyslog.d/50-default.conf` e aggiungere in fondo queste due righe:

```
1 local0.crit                                          /var/log/
      local0crit.log
2 local0.=info                                         /var/log/
      local0info.log
```

Esempio 24.46: *Aggiunta nel file* `/etc/rsyslog.d/50-default.conf`

in cui la cifra `0` può essere sostituita da una cifra compresa tra 0 e 7[59]. Riavviando il servizio `rsyslogd`, si scriva sul **terminale 0**:

```
1 $ logger -p local0.info "Questo è un messaggio info a local0"
```

Esempio 24.47: *Invio di un messaggio di log su una facility appena configurata*

ricordando di sostituire `local0` con quanto indicato in ***Esempio 24.46***.
Successivamente al tasto Enter nel **terminale 0**, sul **terminale 1** si esegua un `ls -l /var/log/local0info.log` e sarà appena comparso tale file, quindi digitare:

[57] equivalentemente `--priority syslog.info`

[58] nell'ipotesi verosimile che il sistema dello sperimentatore abbia una configurazione simile al presente sistema – v. 24.6.2 a pag. 393

[59] v. ***Tabella 24.4***. Ha senso la sostituzione qualora la facility sia già utilizzata

```
1 $ tail -f /var/log/local0info.log
2 Jun 26 12:02:52 nb-m ms: Questo è un messaggio info a local0
```

Esempio 24.48: *Invio di un messaggio di log su una facility appena configurata*

e al `terminale 2` sarà apparso un analogo messaggio.

Ora si digiti sul **terminale 0**

```
1 $ logger -p local0.notice "Questo è un messaggio notice a local0"
```

Esempio 24.49: *Invio di un un messaggio di log su una facility non configurata*

e sugli altri terminali in `follow` si noterà un comportamento differente: sul **terminale 1** non si osserverà alcun messaggio, sul **terminale 2** si leggerà il messaggio inviato.
Questo comportamento differente lo si potrà imputare al simbolo = inserito prima della priority `info` nell'***Esempio 24.46***.

Ovviamente l'invito e riprovare il software `logger` variando le opzioni in accordo al manuale[60] relativo.

24.6 Approfondimenti

24.6.1 Kernel ring buffer

L'accesso al *ring buffer* del kernel avviene attraverso il device `/dev/kmsg` e, usualmente, come avvenuto più volte in questo capitolo, per la lettura si utilizzerà il comando `dmesg`.
Per accedere al device è necessario l'esecuzione del comando con i privilegi del *superutente*, ma è possibile rimuovere tale protezione digitando:

```
1 $ sudo sysctl -w kernel.dmesg_restrict=0
2 kernel.dmesg_restrict = 0
```

Esempio 24.50: *`sysctl` per utilizzare `dmesg` senza essere superutente*

e, dopo aver settato[61] tale variabile a `0`, tutti gli utenti potranno utilizzare il comando `dmesg` per la lettura del buffer.
Per quanto riguarda la scrittura sul kernel ring buffer, i messaggi saranno inviati, oltre che dal kernel, dai processi di sistema accedendo in scrittura al citato device `/dev/kmsg`. Questi accederanno tramite la chiamata esplicita della *system call* `syslog`[62].
Come già trattato nel capitolo e in particolare mostrato nella ***Tabella 24.2***, i messaggi generati dal kernel avranno assegnata la facility `kern`, i messaggi degli utenti `user` e così di seguito.

[60] `man logger`

[61] per rendere persistente il settaggio, digitare `sudo echo "kernel.dmesg_restrict=0" >> /etc/sysctl.conf`

[62] da non confondere con l'omonimo servizio di sistema *syslog* deputato alla gestione dei messaggi dei processi e servizi di sistema

24.6.2 `rsyslog` o `systemd-journal`

Il logger invia dati al device `/dev/log` via *Unix socket*. Storicamente era gestito dal servizio syslog, poi sostituito da rsyslog. Ma per molte distribuzioni è avvenuta l'introduzione di `systemd` che ha preso il controllo del sistema di logging. Per verificare quale sistema di logging ha il proprio sistema sarà sufficiente digitare:

```
1 $ ls -l /dev/log
2 lrwxrwxrwx 1 root root 28 Sep 17  2021 /dev/log -> /run/systemd/journal/
      dev-log
3 $ ls -l /run/systemd/journal/dev-log
4 srw-rw-rw- 1 root root 0 Sep 17  2021 /run/systemd/journal/dev-log
```

Esempio 24.51: *Verifica su `/dev/log` tramite `ls`*

e dalla prima replica si intuisce che il log è gestito via *journal* di `systemd`, la seconda replica conferma che il file è un socket.
Una verifica più rigorosa è tramite il comando `fuser`[63]:

```
1 $ sudo fuser -v /dev/log
2                      UTENTE      PID ACCESS COMMAND
3 /run/systemd/journal/dev-log:
4                      root          1 F.... systemd
5                      root        456 F.... systemd-journal
```

Esempio 24.52: *Verifica su `/dev/log` tramite `fuser`*

in cui è esplicito il nome del processo che utilizza la risorsa.
Come argomentato in 24.4, una seconda prova deriva dalla verifica sul socket `/dev/kmsg`:

```
1 $ sudo fuser -v /dev/kmsg
2                      UTENTE      PID ACCESSO COMANDO
3 /dev/kmsg:           root          1 F.... systemd
4                      root        368 F.... systemd-journal
```

Esempio 24.53: *Verifica su `/dev/kmsg` tramite `fuser`*

Per mantenere funzionante il sistema rsyslog, il demone `systemd-journald` dovrà essere configurato come *relay*[64].

24.6.3 `printk`^{P}

Come accennato in 24.2.1 a pag. 368, differentemente dalla funzione `printf()`, la `printk` permette di assegnare il livello di priorità a ciascun messaggio.
Una tipica chiamata della funzione può essere:

```
1 printk(KERN_INFO "Messaggio: %s\n", arg);
```

Esempio 24.54: *Tipica chiamata della funzione `printk()`*

in cui:

`KERN_INFO` è una delle possibili alternative per l'assegnazione del livello di priorità;

`args` è l'argomento della funzione (nel caso specifico una stringa poiché c'è il `%s`), che possiamo trascurare poiché la trattazione è analoga alla funzione `printf()`.

[63] v. **??** a pag. **??**
[64] v. `man journald.conf` opzione `ForwardToSyslog`

È utile notare che i livelli di priorità da scegliere tra quelli già mostrati in ***Tabella 24.1***, sono concatenati[65] alla stringa che forma il messaggio, in altri termini non è un argomento in più della funzione.

[65] v. colonna **STRING** della medesima tabella

24.7 Concetti chiave

logging: è suddiviso in *kernel space logging* e *user space logging*

`printk()` funzione kernel per il popolamento dei messaggi nel ring buffer

ring buffer: struttura dati circolare, nel contesto è utilizzata dal kernel per l'invio di messaggi

syslog: protocollo per la gestione dei messaggi di log

systemd-journal: componente della suite `systemd` deputato al logging tramite la componente `journald`

24.8 Comandi introdotti nel capitolo

`dmesg` ⇒ mostra e gestisce il *ring buffer* del kernel

`logger` ⇒ software per inviare messaggi al system log

`journalctl` ⇒ esegue query al giornale di `systemd`

Parte V

Complementi

Tabella ASCII

ASCII base character codes 0 – 127

DEC	HEX	GLIFO	ABBR	DEC	HEX	GLIFO	DEC	HEX	GLIFO
000_{10}	00_{16}	␀	nul	043_{10}	$2B_{16}$	+	086_{10}	56_{16}	V
001_{10}	01_{16}	☺	soh	044_{10}	$2C_{16}$	’	087_{10}	57_{16}	W
002_{10}	02_{16}	☻	stx	045_{10}	$2D_{16}$	-	088_{10}	58_{16}	X
003_{10}	03_{16}	♥	etx	046_{10}	$2E_{16}$	.	089_{10}	59_{16}	Y
004_{10}	04_{16}	♦	eot	047_{10}	$2F_{16}$	/	090_{10}	$5A_{16}$	Z
005_{10}	05_{16}	♣	enq	048_{10}	30_{16}	0	091_{10}	$5B_{16}$	[
006_{10}	06_{16}	♠	ack	049_{10}	31_{16}	1	092_{10}	$5C_{16}$	\
007_{10}	07_{16}	•	bel	050_{10}	32_{16}	2	093_{10}	$5D_{16}$	]
008_{10}	08_{16}	◘	bs	051_{10}	33_{16}	3	094_{10}	$5E_{16}$	^
009_{10}	09_{16}		tab	052_{10}	34_{16}	4	095_{10}	$5F_{16}$	_
010_{10}	$0A_{16}$	◙	lf	053_{10}	35_{16}	5	096_{10}	60_{16}	‘
011_{10}	$0B_{16}$	♂	vt	054_{10}	36_{16}	6	097_{10}	61_{16}	a
012_{10}	$0C_{16}$		np	055_{10}	37_{16}	7	098_{10}	62_{16}	b
013_{10}	$0D_{16}$	♪	cr	056_{10}	38_{16}	8	099_{10}	63_{16}	c
014_{10}	$0E_{16}$	♫	so	057_{10}	39_{16}	9	100_{10}	64_{16}	d
015_{10}	$0F_{16}$	☼	si	058_{10}	$3A_{16}$	:	101_{10}	65_{16}	e
016_{10}	10_{16}	►	dle	059_{10}	$3B_{16}$	;	102_{10}	66_{16}	f
017_{10}	11_{16}	◄	dc1	060_{10}	$3C_{16}$	<	103_{10}	67_{16}	g
018_{10}	12_{16}	↕	dc2	061_{10}	$3D_{16}$	=	104_{10}	68_{16}	h
019_{10}	13_{16}	‼	dc3	062_{10}	$3E_{16}$	>	105_{10}	69_{16}	i
020_{10}	14_{16}	¶	dc4	063_{10}	$3F_{16}$	?	106_{10}	$6A_{16}$	j
021_{10}	15_{16}	§	nak	064_{10}	40_{16}	@	107_{10}	$6B_{16}$	k
022_{10}	16_{16}	▬	syn	065_{10}	41_{16}	A	108_{10}	$6C_{16}$	l
023_{10}	17_{16}	↨	etb	066_{10}	42_{16}	B	109_{10}	$6D_{16}$	m
024_{10}	18_{16}	↑	can	067_{10}	43_{16}	C	110_{10}	$6E_{16}$	n
025_{10}	19_{16}	↓	em	068_{10}	44_{16}	D	111_{10}	$6F_{16}$	o
026_{10}	$1A_{16}$		eof	069_{10}	45_{16}	E	112_{10}	70_{16}	p
027_{10}	$1B_{16}$	←	esc	070_{10}	46_{16}	F	113_{10}	71_{16}	q
028_{10}	$1C_{16}$	└	fs	071_{10}	47_{16}	G	114_{10}	72_{16}	r
029_{10}	$1D_{16}$	↔	gs	072_{10}	48_{16}	H	115_{10}	73_{16}	s
030_{10}	$1E_{16}$	▲	rs	073_{10}	49_{16}	I	116_{10}	74_{16}	t
031_{10}	$1F_{16}$	▼	us	074_{10}	$4A_{16}$	J	117_{10}	75_{16}	u
032_{10}	20_{16}	␣		075_{10}	$4B_{16}$	K	118_{10}	76_{16}	v
033_{10}	21_{16}	!		076_{10}	$4C_{16}$	L	119_{10}	77_{16}	w
034_{10}	22_{16}	"		077_{10}	$4D_{16}$	M	120_{10}	78_{16}	x
035_{10}	23_{16}	#		078_{10}	$4E_{16}$	N	121_{10}	79_{16}	y

036_{10}	24_{16}	$	079_{10}	$4F_{16}$	O	122_{10}	$7A_{16}$	z
037_{10}	25_{16}	%	080_{10}	50_{16}	P	123_{10}	$7B_{16}$	{
038_{10}	26_{16}	&	081_{10}	51_{16}	Q	124_{10}	$7C_{16}$	|
039_{10}	27_{16}	'	082_{10}	52_{16}	R	125_{10}	$7D_{16}$	}
040_{10}	28_{16}	(	083_{10}	53_{16}	S	126_{10}	$7E_{16}$	~
041_{10}	29_{16}	)	084_{10}	54_{16}	T	127_{10}	$7F_{16}$	⌂
042_{10}	$2A_{16}$	*	085_{10}	55_{16}	U			

extended ASCII - character codes 128 – 255

128_{10}	80_{16}	€	171_{10}	AB_{16}	«	214_{10}	$D6_{16}$	Ö
129_{10}	81_{16}		172_{10}	AC_{16}	¬	215_{10}	$D7_{16}$	×
130_{10}	82_{16}	‚	173_{10}	AD_{16}		216_{10}	$D8_{16}$	Ø
131_{10}	83_{16}	*f*	174_{10}	AE_{16}	®	217_{10}	$D9_{16}$	Ù
132_{10}	84_{16}	„	175_{10}	AF_{16}	¯	218_{10}	DA_{16}	Ú
133_{10}	85_{16}	…	176_{10}	$B0_{16}$	°	219_{10}	DB_{16}	Û
134_{10}	86_{16}	†	177_{10}	$B1_{16}$	±	220_{10}	DC_{16}	Ü
135_{10}	87_{16}	‡	178_{10}	$B2_{16}$	²	221_{10}	DD_{16}	Ý
136_{10}	88_{16}	^	179_{10}	$B3_{16}$	³	222_{10}	DE_{16}	Þ
137_{10}	89_{16}	‰	180_{10}	$B4_{16}$	´	223_{10}	DF_{16}	ß
138_{10}	$8A_{16}$	Š	181_{10}	$B5_{16}$	µ	224_{10}	$E0_{16}$	à
139_{10}	$8B_{16}$	‹	182_{10}	$B6_{16}$	¶	225_{10}	$E1_{16}$	á
140_{10}	$8C_{16}$	Œ	183_{10}	$B7_{16}$	·	226_{10}	$E2_{16}$	â
141_{10}	$8D_{16}$		184_{10}	$B8_{16}$	¸	227_{10}	$E3_{16}$	ã
142_{10}	$8E_{16}$	Ž	185_{10}	$B9_{16}$	¹	228_{10}	$E4_{16}$	ä
143_{10}	$8F_{16}$		186_{10}	BA_{16}	º	229_{10}	$E5_{16}$	å
144_{10}	90_{16}		187_{10}	BB_{16}	»	230_{10}	$E6_{16}$	æ
145_{10}	91_{16}	‘	188_{10}	BC_{16}	¼	231_{10}	$E7_{16}$	ç
146_{10}	92_{16}	’	189_{10}	BD_{16}	½	232_{10}	$E8_{16}$	è
147_{10}	93_{16}	“	190_{10}	BE_{16}	¾	233_{10}	$E9_{16}$	é
148_{10}	94_{16}	”	191_{10}	BF_{16}	¿	234_{10}	EA_{16}	ê
149_{10}	95_{16}	•	192_{10}	$C0_{16}$	À	235_{10}	EB_{16}	ë
150_{10}	96_{16}	–	193_{10}	$C1_{16}$	Á	236_{10}	EC_{16}	ì
151_{10}	97_{16}	—	194_{10}	$C2_{16}$	Â	237_{10}	ED_{16}	í
152_{10}	98_{16}	~	195_{10}	$C3_{16}$	Ã	238_{10}	EE_{16}	î
153_{10}	99_{16}	™	196_{10}	$C4_{16}$	Ä	239_{10}	EF_{16}	ï
154_{10}	$9A_{16}$	š	197_{10}	$C5_{16}$	Å	240_{10}	$F0_{16}$	ð
155_{10}	$9B_{16}$	›	198_{10}	$C6_{16}$	Æ	241_{10}	$F1_{16}$	ñ
156_{10}	$9C_{16}$	œ	199_{10}	$C7_{16}$	Ç	242_{10}	$F2_{16}$	ò
157_{10}	$9D_{16}$		200_{10}	$C8_{16}$	È	243_{10}	$F3_{16}$	ó
158_{10}	$9E_{16}$	ž	201_{10}	$C9_{16}$	É	244_{10}	$F4_{16}$	ô
159_{10}	$9F_{16}$	Ÿ	202_{10}	CA_{16}	Ê	245_{10}	$F5_{16}$	õ
160_{10}	$A0_{16}$		203_{10}	CB_{16}	Ë	246_{10}	$F6_{16}$	ö
161_{10}	$A1_{16}$	¡	204_{10}	CC_{16}	Ì	247_{10}	$F7_{16}$	÷
162_{10}	$A2_{16}$	¢	205_{10}	CD_{16}	Í	248_{10}	$F8_{16}$	ø
163_{10}	$A3_{16}$	£	206_{10}	CE_{16}	Î	249_{10}	$F9_{16}$	ù
164_{10}	$A4_{16}$	¤	207_{10}	CF_{16}	Ï	250_{10}	FA_{16}	ú
165_{10}	$A5_{16}$	¥	208_{10}	$D0_{16}$	Đ	251_{10}	FB_{16}	û
166_{10}	$A6_{16}$	¦	209_{10}	$D1_{16}$	Ñ	252_{10}	FC_{16}	ü

167_{10}	$A7_{16}$	§	210_{10}	$D2_{16}$	Ò	253_{10}	FD_{16}	ý
168_{10}	$A8_{16}$	¨	211_{10}	$D3_{16}$	Ó	254_{10}	FE_{16}	þ
169_{10}	$A9_{16}$	©	212_{10}	$D4_{16}$	Ô	255_{10}	FF_{16}	ÿ
170*d*	AA*h*	ª	213*d*	D5*h*	Õ			

Tabella 24.6: *Extended ASCII - character codes 0 – 255* LATIN1/CP1252

Tavola di conversione esadecimale, decimale, ottale e binario

Di seguito una tavola per il confronto di valori da 0 a 15:

HEX *esadecimale*, in base 16;

DEC *decimale*, in base 10;

OCT *ottale*, in base 8;

BIN *binario*, in base 2.

HEX	DEC	OCT	BIN	HEX	DEC	OCT	BIN
0	00	00	0000	8	08	10	1000
1	01	01	0001	9	09	11	1001
2	02	02	0010	A	10	12	1010
3	03	03	0011	B	11	13	1011
4	04	04	0100	C	12	14	1100
5	05	05	0101	D	13	15	1101
6	06	06	0110	E	14	16	1110
7	07	07	0111	F	15	17	1111

Tabella 24.7: *Hexadecimal, Decimal, Octal and Binary*

Multipli del byte

Esistono due scale concorrenti per i multipli del byte basate rispettivamente sulla potenza di:

10: è la storica scala del **SI**[66]

2: è definito dallo standard **IEEE**[67] **1541**[68].

In tabella sono mostrate entrambe le scale dei multipli del byte.

SI			IEEE 1541		
VALORE	SIGLA	NOME	VALORE	SIGLA	NOME
1.000^1	kB	kilobyte	1.024^1	KiB	kibibyte
1.000^2	MB	megabyte	1.024^2	MiB	mebibyte
1.000^3	GB	gigabyte	1.024^3	GiB	gibibyte
1.000^4	TB	terabyte	1.024^4	TiB	tebibyte
1.000^5	PB	petabyte	1.024^5	PiB	pebibyte
1.000^6	EB	exabyte	1.024^6	EiB	exbibyte
1.000^7	ZB	zettabyte	1.024^7	ZiB	zebibyte
1.000^8	YB	yottabyte	1.024^8	YiB	yobibyte
1.000^9	RB	ronnabyte			
1.000^{10}	QB	quettabyte			

Tabella 24.8: *Grandezza numerica, sigla e nome delle scale*

Nell'***Esempio 24.55*** un semplice script per apprezzare lo scarto nel confondere i prefissi tra i due sistemi relativi alla medesima grandezza. L'output dello script è mostrato nell'***Esempio 24.56***, in cui è evidente che per grandezze omogenee, salendo nella scala, il rapporto reciproco mostra uno scarto sempre maggiore.

[66] la *Convenzione del Metro* del 1875 ha posto le basi per un *sistema di unità di misura* comune chiamato comunemente ***S**istema **I**nternazionale*, denominato per esteso il **sistema internazionale di unità di misura** (in francese: *Système international d'unités*), ed è il più diffuso sistema di unità di misura

[67] L'***I**nstitute of **E**lectrical and **E**lectronics **E**ngineers* è un'associazione internazionale di scienziati con l'obiettivo della promozione delle scienze tecnologiche

[68] è uno standard emanato dall'IEEE riguardante l'uso di prefissi per multipli binari delle unità di misura relative all'elettronica digitale e all'informatica

```
 1 use strict;
 2 use warnings;
 3 use bignum;
 4
 5 my $base_binaria = 2**10;
 6 my $base_decimale = 10**3;
 7
 8 foreach my $exp (0..8) {
 9     my $value_binario = $base_binaria**$exp;
10     my $value_decimale = $base_decimale**$exp;
11     print "exp $exp",
12         "\n  b ->   ", $value_binario,
13         "\n  d ->   ", $value_decimale,
14         (sprintf "\n  b/d -> %.2f", $value_binario/$value_decimale)
       ,
15         sprintf "\n  d/b -> %.2f\n", $value_decimale/$value_binario
       ;
16 }
```

Esempio 24.55: *Script Perl per la valutazione degli scarti tra prefissi omogenei*

```
$ perl byte_scale.pl
exp 0
  b ->   1
  d ->   1
  b/d -> 1.00
  d/b -> 1.00
exp 1
  b ->   1,024
  d ->   1,000
  b/d -> 1.02
  d/b -> 0.98
exp 2
  b ->   1,048,576
  d ->   1,000,000
  b/d -> 1.05
  d/b -> 0.95
exp 3
  b ->   1,073,741,824
  d ->   1,000,000,000
  b/d -> 1.07
  d/b -> 0.93
exp 4
  b ->   1,099,511,627,776
  d ->   1,000,000,000,000
  b/d -> 1.10
  d/b -> 0.91
exp 5
  b ->   1,125,899,906,842,624
  d ->   1,000,000,000,000,000
  b/d -> 1.13
  d/b -> 0.89
exp 6
  b ->   1,152,921,504,606,846,976
  d ->   1,000,000,000,000,000,000
  b/d -> 1.15
  d/b -> 0.87
exp 7
  b ->   1,180,591,620,717,411,303,424
  d ->   1,000,000,000,000,000,000,000
  b/d -> 1.18
  d/b -> 0.85
exp 8
  b ->   1,208,925,819,614,629,174,706,176
  d ->   1,000,000,000,000,000,000,000,000
  b/d -> 1.21
  d/b -> 0.83
```

Esempio 24.56: *Output dello script dell'***Esempio 24.55**

Indice analitico

<, 92
>, 92
>>, 92
[. . .], 333
Ctrl + C , 27, 71
Ctrl + D , 16, 27, 63, 65, 188
Ctrl + U , 188
Ctrl + d , 182
Ctrl + r , 159
Ctrl + C , 65
/, 33
$?, 332
bunzip2, 113
bzip2, 113
cd, 19
chfn, 216
convmv, 32
gunzip, 112
gzip, 112
halt, 362
iwconfig, 285
netcat, 149
poweroff, 363
reboot, 363
shutdown, 361
wall, 361

, 21, 160, 326
--help, 23
.bashrc, 165
/etc/passwd, 109, 215
/etc/shadow, 216
0<, 93
1>, 93
2>, 93
<<, 94
>&, 94
FD<..., 151
FD<>..., 151
FD>..., 151
FD>>..., 151
LC_ALL, 105
[[...]], 334
$(...), 97
$?, 156
/, 31
\, 37
`...`, 97
addgroup, 223
adduser, 219
alias, 157
apropos, 27
bash, 17
bc, 96, 176
blkid, 364
bzcat, 114
cat, 63
cd, 19, 35
checkinstall, 255
chgrp, 223, 240
chkconfig, 356
chmod, 235
chown, 240
chsh, 216
convmv, 177
cp, 40
csh, 17
cut, 107
date, 19, 20
dig, 291
dirs, 161
dmesg, 142, 368
dmidecode, 301
doas, 197
echo, 20, 46
exit, 16, 23
export, 124
fdisk, 146
fg, 175
file, 56, 131
find, 50
free, 141, 201
fsck, 143
gdebi, 264
git, 254
groff, 24
groupadd, 223
groupdel, 223
groupmod, 222, 223
groups, 214
hash, 127
hdparm, 305
head, 68
help, 20, 23
history, 159
hwinfo, 297
ifconfig, 285, 287, 314
init, 357
inxi, 295
ip, 285, 313
jobs, 175
journalctl, 383
ksh, 17
less, 67
ln, 43
locale, 179
locate, 49
login, 15

logout, 16
lsb_release, 139
lsblk, 135, 364
lscpu, 300
lshw, 299
lsof, 149
lspci, 303
lsusb, 309
ls, 36
man, 24
mkdir, 38
mkfifo, 133, 148
mknod, 133
mv, 41
ncal, 156
nc, 149
netstat, 315
newgrp, 217
nmcli, 314
ping6, 290
ping, 288
plocate, 49
popd, 161
pr, 178
ps, 198
pushd, 161
pwd, 35
rev, 65
rmdir, 39
rm, 43, 57
route, 285
runlevel, 357
shasum, 121
shutdown, 359
sh, 17
sleep, 171
sl, 37
smartctl, 307
sort, 103
source, 166
split, 119
stat, 234
stty, 188
sudo, 196
su, 195
systemctl, 363
tac, 63
tail, 69
tar, 115
tcpdump, 315
tee, 96
telinit, 357
test, 333
timeout, 172
time, 172
top, 202
touch, 31
tree, 33
tr, 179
type, 19, 23
umask, 246
umount, 147
unalias, 159
uniq, 107
unset, 123
update-rc.d, 356
uptime, 140
userdel, 221
usermod, 222
wait, 175
wall, 359
watch, 169
wget, 254
whatis, 27
which, 19
xargs, 170
zcat, 113

alias, 36

backtick, 97
BIOS, 348
boot, 134
boot loader, 134, 347
BRE, 80
buffering, 135

caratteri speciali, 20
case
    insensitive, 52
cat, 119
checksum
    SHA, 121
chunk, 119
classi di carattere, 46
command substitution, 97

device, 132
    block, 135
    character, 135
device driver, 135
digest, 121
directory, 33
    current, 34, 35
    home, 33, 34
    parent, 35
    root, 33, 133
    working, 34, 35, 198

editor
    modale, 74
egrep, 84
emacs, 76
ERE, 81
escaping, 20, 32
espressione regolare, 79
exec, 151
exit status, 156, 332

FD, 92, 150
fgrep, 84
FIFO, 132
file, 31
    special, 135
    symbolic link, 132
file descriptor, 92, 150
filesystem, 131
fork bomb, 343
FOSS, 10
free software, 6

GID, 214
globbing, 45
GNU, 6
GNU/Linux, 9
GPT, 360
greedy, 83
grep, 84
GRUB, 134, 349

here
    document, 94
    strings, 96

ICMP, 288
indirizzo
    IP, 278

logico, 278
init, 354
inode, 132
interattivo, 321
interpolazione, 21
interprete, 322
IPC
pipeline, 96
Unix domain
socket, 132, 148
IPv4, 278

kernel, 134, 354

LAN, 278
LILO, 134
link
hard, 43, 54
soft, 44
Linux, 7
login, 15

magic number, 342
MBR, 348
metacaratteri, 45
mount point, 147

nano, 73
newgrp, 222

overwrite, 41

path, 33
assoluto, 33
relativo, 34
PCRE, 83
pico, 73
pipeline, 96
process
table, 199
processo, 198
child, 156
init, 210
parent, 156
prompt, 18

rgrep, 84
root, 193
runlevel, 354

set-GID bit, 242
set-UID bit, 242
SGID bit, 242
shell
Bourne, 17
Bourne again, 17
C, 17
Korn, 17
shutdown, 134
software libero, 6
sort, 98

standard
channels, 91
error, 91
input, 91
output, 91
standard output, 51
subnet mask, 279
SUID bit, 242
syntax highlighting, 73
system call, 151

TCP/IP, 273
type, 126

UEFI, 359
UID, 213
Unix, 5
Unix domain socket, 132

variabile
di ambiente, 124
di environment, 124
di shell, 123, 124
vi, 74
vim, 74
vimtutor, 75

wc, 93
wildcard, 45

Bibliografia e Webografia

[81a] *Internet Control Message Protocol*. RFC 792. Set. 1981. DOI: 10.17487/RFC0792. URL: https://www.rfc-editor.org/info/rfc792.

[81b] *Internet Protocol*. RFC 791. Set. 1981. DOI: 10.17487/RFC0791. URL: https://www.rfc-editor.org/info/rfc791.

[87a] *Domain names - concepts and facilities*. RFC 1034. Nov. 1987. DOI: 10.17487/RFC1034. URL: https://www.rfc-editor.org/info/rfc1034.

[87b] *Domain names - implementation and specification*. RFC 1035. Nov. 1987. DOI: 10.17487/RFC1035. URL: https://www.rfc-editor.org/info/rfc1035.

[Ful+93] Vince Fuller et al. *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*. RFC 1519. Set. 1993. DOI: 10.17487/RFC1519. URL: https://www.rfc-editor.org/info/rfc1519.

[RL93] Yakov Rekhter e Tony Li. *An Architecture for IP Address Allocation with CIDR*. RFC 1518. Set. 1993. DOI: 10.17487/RFC1518. URL: https://www.rfc-editor.org/info/rfc1518.

[Deu96] L. Peter Deutsch. *DEFLATE Compressed Data Format Specification version 1.3*. RFC 1951. Mag. 1996. DOI: 10.17487/RFC1951. URL: https://www.rfc-editor.org/info/rfc1951.

[Mos+96] Robert Moskowitz et al. *Address Allocation for Private Internets*. RFC 1918. Feb. 1996. DOI: 10.17487/RFC1918. URL: https://www.rfc-editor.org/info/rfc1918.

[DC98] Dr. Steve E. Deering e Alex Conta. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 2463. Dic. 1998. DOI: 10.17487/RFC2463. URL: https://www.rfc-editor.org/info/rfc2463.

[DM99] Ockman S. DiBona C. e Stone M. *Open Sources. Voci dalla rivoluzione Open Source*. Apogeo, gen. 1999, p. 250.

[Pee+02] Jerry Peek et al. *Unix Power Tools*. 3ª ed. O'Reilly Media, Inc., 2002.

[TD02] Linus Torvalds e David Diamond. *Just for Fun: The Story of an Accidental Revolutionary*. Harper Business, giu. 2002, p. 288.

[Les05] Lawrence Lessig. *Cultura libera*. Apogeo, feb. 2005, p. 256.

[Cev06] Gustavo Cevolani. "Norme tipografiche per l'italiano in LATEX". In: *ArsTeXnica* 1 (2006), pp. 29–42.

[Ger09] Rainer Gerhards. *The Syslog Protocol*. RFC 5424. Mar. 2009. DOI: 10.17487/RFC5424. URL: https://www.rfc-editor.org/info/rfc5424.

[Edd22] Wesley Eddy. *Transmission Control Protocol (TCP)*. RFC 9293. Ago. 2022. DOI: 10.17487/RFC9293. URL: https://www.rfc-editor.org/info/rfc9293.

[SK] Gilles 'SO e Stephen Kitt. *Allow setuid on shell scripts*. URL: https://unix.stackexchange.com/questions/364/allow-setuid-on-shell-scripts.

[aar] aaronryank. *fork-bomb*. URL: https://github.com/aaronryank/fork-bomb/tree/master.

[Com] Debian Community. *Debian wiki*. URL: https://wiki.debian.org/.

[coma] kernel development community. *Message logging with printk*. URL: https://www.kernel.org/doc/html/next/core-api/printk-basics.html.

[comb] Ubuntu community. *Smartmontools*. URL: https://help.ubuntu.com/community/Smartmontools.

[comc] Wikipedia community. *Processor Info and Feature Bits*. URL: https://en.wikipedia.org/wiki/CPUID#EAX.3D1:_Processor_Info_and_Feature_Bits.

[Coo] Mendel Cooper. *Advanced Bash-Scripting Guide*. URL: https://tldp.org/LDP/abs/abs-guide.pdf.

[dev] GNU developers. *GNU GRUB Manual 0.97*. URL: https://www.gnu.org/software/grub/manual/legacy/grub.html.

[DM] Ockman S. DiBona C. e Stone M. *The Tanenbaum-Torvalds Debate*. URL: https://www.oreilly.com/openbook/opensources/book/appa.html.

[FAQ] Unix FAQ. *Unix - Frequently Asked Questions*. URL: http://www.faqs.org/faqs/unix-faq/faq/part4/section-7.html.

[Foua] Free Software Foundation. *Bash manual*. URL: https://www.gnu.org/software/bash/manual/html_node/index.html.

[Foub] The Free Software Foundation. *GNU/Linux FAQ*. URL: https://www.gnu.org/gnu/gnu-linux-faq.html.en.

[Fouc] The Free Software Foundation. *What is Free Software?* URL: https://www.gnu.org/philosophy/free-sw.html.en.

[ita] Comunità Debian italiana. *https://guide.debianizzati.org/*. URL: https://guide.debianizzati.org.

[Mas] Sven Mascheck. *The #! magic, details about the shebang/hash-bang mechanism on various Unix flavours*. URL: https://www.in-ulm.de/~mascheck/various/shebang/.

[Ove] Duncan Overbruck. *OpenDoas: a portable version of OpenBSD's doas command*. URL: https://github.com/Duncaen/OpenDoas/tree/master.

[sig] sigoden. *UPT — Universal Package-management Tool*. URL: https://github.com/sigoden/upt.

[Ste] Steinar. *plocate, a much faster locate*. URL: https://plocate.sesse.net/.

[Vin] Lyonel Vincent. *Hardware Lister (lshw)*. URL: https://ezix.org/project/wiki/HardwareLiSter.