La shell di Unix

Uso interattivo e scripting

Lucidi per il corso di Laboratorio di Sistemi Operativi tenuto da Paolo Baldan presso l'Università Ca' Foscari di Venezia, anno accademico 2004/2005.

Obiettivi

L'obiettivo di questo ciclo di lezioni è quello di illustrare le funzionalità principali della shell di UNIX e di tools comunemente presenti nelle distribuzioni di UNIX.

- Uso interattivo della shell

  Concetti di base, interazione con la shell, comandi di base, personalizzazione.

- Struttura e funzionalità della shell di UNIX (con particolare riferimento a Linux ed alla Bash shell).

  Espansione della riga di comando, ridirezione, comandi composti (liste, pipe, sequenze condizionali).

- Programmazione della shell (shell scripting).

  Funzioni, pattern matching, costrutti di controllo, tipi, segnali, debugging.

Introduzione a UNIX

(Ripasso)

Cos'è UNIX

- UNIX è un sistema operativo multiutente e multitasking.
  Più utenti possono avere vari tasks/processi che sono eseguiti contemporaneamente.

- UNIX è anche un ambiente di sviluppo software.
  È nato ed è stato progettato per questo scopo. (Ad esempio fornisce utilità di sistema, editor, compilatori, assemblatori, interpreti, . . .)

- UNIX è stato appositamente progettato per essere generale e indipendente dall'hardware (facilmente portabile essendo scritto nel linguaggio ad alto livello C).

Breve storia di UNIX/A

- 1965: La AT&T, il MIT e la General Electric si uniscono nel progetto di un s.o. innovativo, che sia multiutente, multitasking, multiprocessore, con multilevel file system: MULTICS.
- 1969: Il progetto MULTICS “fallisce”.
- Presso i Bell Laboratories della AT&T, dalle ceneri di MULTICS nasce UNIX, sviluppato da Ken Thompson e Dennis Ritchie del gruppo Multics per un PDP-7. Anche sulla base dell’esperienza MULTICS, incorpora alcune caratteristiche e supera i limiti dei sistemi operativi preesistenti, supportando, in modo efficiente, multiutenza e multitasking.

  Il nome UNIX (inizialmente Unics) è scelto da Kernighan in opposizione a MULTICS.

Breve storia di UNIX / C

- 1973: La terza versione viene scritta nel linguaggio di programmazione ad alto livello C (anziché in assembler), sviluppato ai Bell Labs da Dennis Ritchie (come evoluzione di B) proprio per supportare UNIX.

  È un passo importante dato che esisteva la convinzione che solo il linguaggio Assembly permettesse di ottenere un livello di efficienza accettabile.

- Fine anni ’70: AT&T rende UNIX largamente disponibile, offrendolo alle Università a basso costo e distribuisce i sorgenti di varie versioni (costretta da un decreto dell’antitrust che le impone di scegliere tra telecomunicazioni e informatica).

Breve storia di UNIX / D

- 1984: presso l'Università di Berkeley (California) nasce la Berkeley Software Distribution (BSD).
  - versione 4BSD: sviluppata con una sovvenzione del DARPA al fine di progettare uno standard UNIX per uso governativo.
  - versione 4.3BSD: sviluppata originariamente per il VAX, è una delle versioni più influenti. Verrà "portata" su molte altre piattaforme.
- 1984: molte caratteristiche della BSD vengono incorporate nella nuova versione di AT&T: la System V.
- Esistono oggi diverse implementazioni di UNIX supportate da molte case costruttrici di computer: SunOS/Solaris (Sun Microsystems), Ultrix (DEC), XENIX (Microsoft), AIX (IBM), ...

Storia di “free” UNIX

- 1983: Richard Stallmann (MIT) inizia una reimplementazione “free” di UNIX, detta GNU (acronimo ricorsivo di “GNU is not UNIX”) che porterà a un editor (`emacs`), un compilatore (`gcc`), un debugger (`gdb`) e numerosi altri tools.

- 1991: Linus Torvalds (studente ad Helsinki) inizia l’implementazione di un kernel POSIX compatibile, che prenderà il nome di Linux. Arricchito con i tool GNU esistenti ed altri sviluppati da volontari, diviene un sostituto completo di UNIX.

- 1991: L’Università di Berkeley rilascia una versione “free” di UNIX, rimuovendo il codice proprietario rimanente della AT&T. Progetti volontari continueranno poi il suo sviluppo (FreeBSD, NetBSD, OpenBSD).

Copyleft

- Public domain
  - Il metodo più semplice per produrre “free software” è quello di renderlo di pubblico dominio, senza copyright.
  - In questo modo, però, qualcuno può modificare il software e distribuirlo come un prodotto proprietario.

- Copyleft: permette di distribuire un software come free, garantendo che eventuali versioni modificate ed estese saranno ancora free.
  - Ogni utente può avere accesso libero al software.
  - Può avere accesso ai sorgenti, modificare il programma ed estenderlo.
  - Può ridistribuirlo, ma con lo stesso tipo di licenza.

Filosofia di UNIX

Influenzata da problematiche relative all'HW disponibile all'epoca della progettazione di UNIX: i dispositivi di I/O erano teletype e più tardi terminali video con 80 caratteri per linea (ASCII fixed width).

- Comandi, sintassi di input e output succinti (per ridurre il tempo di battitura).
- Ogni componente/programma/tool realizza una sola funzione in modo semplice ed efficiente (UNIX vs. MULTICS). Es. `who` e `sort`.
- L'output di un programma può diventare l'input di un altro. Programmi elementari possono essere combinati per eseguire compiti complessi (Es. `who | sort`).
- Il principale software di interfaccia (la shell) è un normale programma, sostituibile, senza privilegi speciali.
- Supporto per automatizzare attività di routine.

Motivi del successo di UNIX

- Ha dimensioni relativamente ridotte, un progetto modulare e “pulito”
- Indipendenza dall’hardware
  - Il codice del SO è scritto in C anziché in uno specifico linguaggio assembler.
  - UNIX e le applicazioni UNIX possono essere “facilmente” portate da un sistema hardware ad un altro. Il porting usualmente consiste nel trasferire il codice sorgente e ricompilarlo.
- È un ambiente produttivo per lo sviluppo del software
  - Ricco insieme di tool disponibili.
  - Linguaggio di comandi versatile.

Principali caratteristiche di UNIX

- Gestione dei Processi
  - Sono basati sul modello fork / exec
  - Hanno un'identità ed un possessore, informazioni utilizzate per protezione e scheduling.
  - La CPU è gestita in time-sharing, suddividendo il tempo di CPU in slice (quanti di tempo), assegnate ai processi in base a priorità (e comportamento passato).
  - Supporto alla programmazione concorrente e distribuita, tramite meccanismi di IPC.

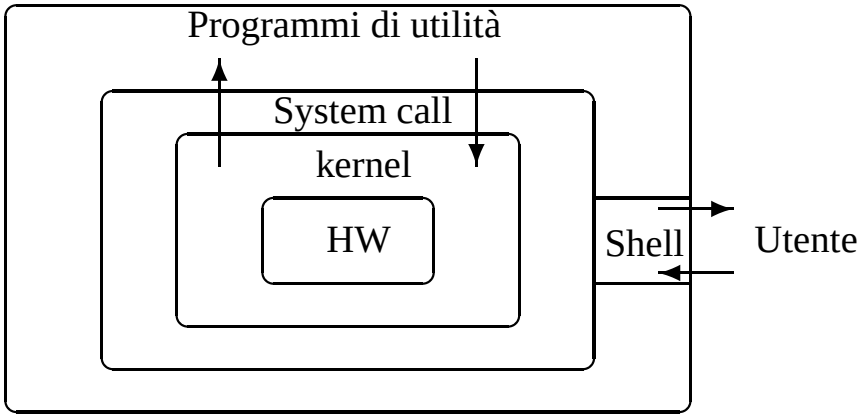
- Gestione della memoria
  - basata su paginazione
  - con varianti, es. buddy system

Principali caratteristiche di UNIX/A

- File System
  - Struttura gerarchica ad albero o grafo radicato, in cui file e processi hanno una propria locazione.
  - Strumenti per la creazione, modifica e distruzione e protezione (controllo degli accessi) ai file.
  - Unix è orientato ai file: Tratta i dispositivi periferici come se fossero dei file (speciali). Stampanti, terminali ed altri dispositivi sono accessibili nello stesso modo di file ordinari.

Componenti di UNIX

- Kernel: nucleo del sistema (gestore di risorse: CPU, memoria, dispositivi, . . .)
- System Call e routine di libreria: facilitano la programmazione, permettendo di accedere a funzionalità del kernel (manipolazione di file, gestione di processi concorrenti e comunicazione tra processi).
- Shell: interprete di comandi (interfaccia tra l'utente ed il kernel).
- Utilities: programmi di utilità (editor, operazioni su file, stampa, supporto alla programmazione, . . .)



Uso interattivo della shell

La Shell (informalmente . . .)

- Interfaccia (testuale) tramite la quale l'utente interagisce con il sistema dopo il login.
- È un interprete di comandi: legge ciascuna linea di comando, ne interpreta le diverse componenti e la esegue.
- Ogni sistema UNIX mette a disposizione vari tipi di shell. Le shell più diffuse sono:

  Bourne shell (sh), C shell (csh), Korn shell (ksh), TC shell (tcsh), Bourne Again shell (bash).

  Inizialmente, l'amministratore del sistema fornisce all'utente una shell di default che però può essere cambiata o personalizzata.
- Ogni shell fornisce un linguaggio di programmazione. I programmi di shell, chiamati shell script, sono sequenze di comandi, con costrutti di controllo, che automatizzano semplici task.

Perché usare una shell testuale?

Ormai tutti i sistemi UNIX hanno un'interfaccia grafica. Perché usare i comandi in linea di una shell testuale?

- Potenza e semplicità: I comandi UNIX sono progettati per risolvere problemi specifici. Sono semplici (senza menù e opzioni nascoste) e proprio per questo potenti (es. `grep parola filename`).
- Velocità e flessibilità: è più veloce scrivere pochi caratteri da tastiera piuttosto che cercare un programma opportuno e usare le operazioni che fornisce sulla base delle proprie specifiche esigenze.
- Accessibilità: permette di accedere efficientemente ad un sistema in remoto.

I comandi UNIX

La sintassi tipica dei comandi UNIX (builtin e file eseguibili) è la seguente

`comando` <opzioni> <argomenti>

Opzioni

- Sono opzionali e influiscono sul funzionamento del comando.
- Consistono generalmente di un hyphen (`-`) seguito da una sola lettera (Es. `ls -l`). Hanno anche una forma estesa (Es. `ls --format long`). Possono avere un argomento (Es. `dvips -o p.ps p.dvi`). Spesso più opzioni possono essere raggruppate insieme dopo un solo hyphen (Es. `ls -al`).

Argomenti

- Si possono avere più argomenti o nessuno.
- Alcuni argomenti sono opzionali. Se non specificati assumono valori di default.

I comandi di UNIX - Esempi

- nessun argomento: il comando `date` mostra data e ora corrente.

```
date
```

- un solo argomento: il comando `cd` cambia la directory di lavoro corrente in quella specificata dal suo argomento.

```
cd mydir
```

- un'opzione ed un argomento: il comando `wc` conta il numero di parole/caratteri/righe in un file, in base all'opzione specificata.

`wc -w file1`	*conta le parole in file1*
`wc -c file1`	*conta i caratteri in file1*
`wc -l file1`	*conta le linee in file1*

I comandi di UNIX - Esempi / A

- numero arbitrario di argomenti: il comando `cat` concatena e mostra sullo schermo il contenuto dei file argomento.

```
cat file1 file2 file3
```

- più opzioni ed un argomento di default:

```
ls -als
```

  lista dettagliata (`l`, long) di tutti (`a`, all) i file, con dimensione (`s`, size). L'argomento di default è la directory corrente.

- Più comandi, separati da "`;`", possono essere scritti sulla stessa linea.

```
cd newdir ; ls -l
```

- I comandi possono essere scritti su più linee usano il carattere backslash (\). Ad esempio,

```
cp /users/kelly/recipe kelly.recipe ; \
lpr -Pps3 kelly.recipe
```

Caratteri di controllo

Alcune combinazioni di tasti hanno un effetto particolare sul terminale. I caratteri di “controllo” sono ottenuti tenendo premuto il tasto Ctrl e quindi premendo un secondo tasto. Ecco i più comuni:

```
Ctrl-u - erase the command line
Ctrl-c - stop/kill a command
Ctrl-h - backspace (usually)
Ctrl-z - suspend a command
Ctrl-s - stop the screen from scrolling
Ctrl-q - continue scrolling
Ctrl-d - exit an interactive program
         (signals end of data)
```

Esempio - Ctrl-d

- Il programma `cat`, senza argomenti, copia l'input (battuto da tastiera) sull'output (schermo). Es.

```
/home/rossi$ cat
Questo l'ho scritto io! ← input
Questo l'ho scritto io! ← output
```

  La mancanza del prompt indica che siamo ancora in `cat`.

  Per segnalare a `cat` (ed a molti altri programmi interattivi) la fine dell'inserimento dei dati occorre digitare Ctrl-d, detto carattere di fine-file (End-Of-File o EOF).

- Esempio: Provate il programma `sort`: se inserite un paio di linee e poi premete Ctrl – d, `sort` le visualizzerà in ordine alfabetico.

Editing della linea di comando

Molte shell, tra cui `bash`, offrono funzioni di editing della linea di comando che sono “ereditate” dall’editor `emacs`. Ecco alcune delle funzioni più utili:

```
Ctrl-a - va a inizio riga
Ctrl-e - va a fine riga
Ctrl-k - cancella il resto della linea
Ctrl-y - reinserisce la stringa cancellata
Ctrl-d - cancella il carattere sul cursore
```

La shell, inoltre, registra i comandi inseriti dall’utente. Il comando `history` li elenca. Per richiamarli ...

```
!!  - richiama il comando precedente
!n  - richiama la n-ma entry
↑,↓ - naviga nella history
```

Completamento dei comandi

Un'altra caratteristica di `bash` è il completamento automatico della linea di comando con il tasto Tab.

- Vediamo un esempio di utilizzo. Supponiamo di essere in questa situazione

```
/home/rossi$ ls
un-file-con-un-nome-molto-lungo
```

- Per copiare il file su `corto`, anziché digitare per esteso il nome si può digitare `cp un`. Quindi premendo il tasto Tab il resto del nome viene mostrato nella riga di comando e si potrà a questo punto completare il comando.
- Se esistono più completamenti possibili, premendo Tab verrà emesso un suono. Premendo nuovamente Tab si ottiene la lista di tutti i file che iniziano in quel modo.

Gestione dei file

File system

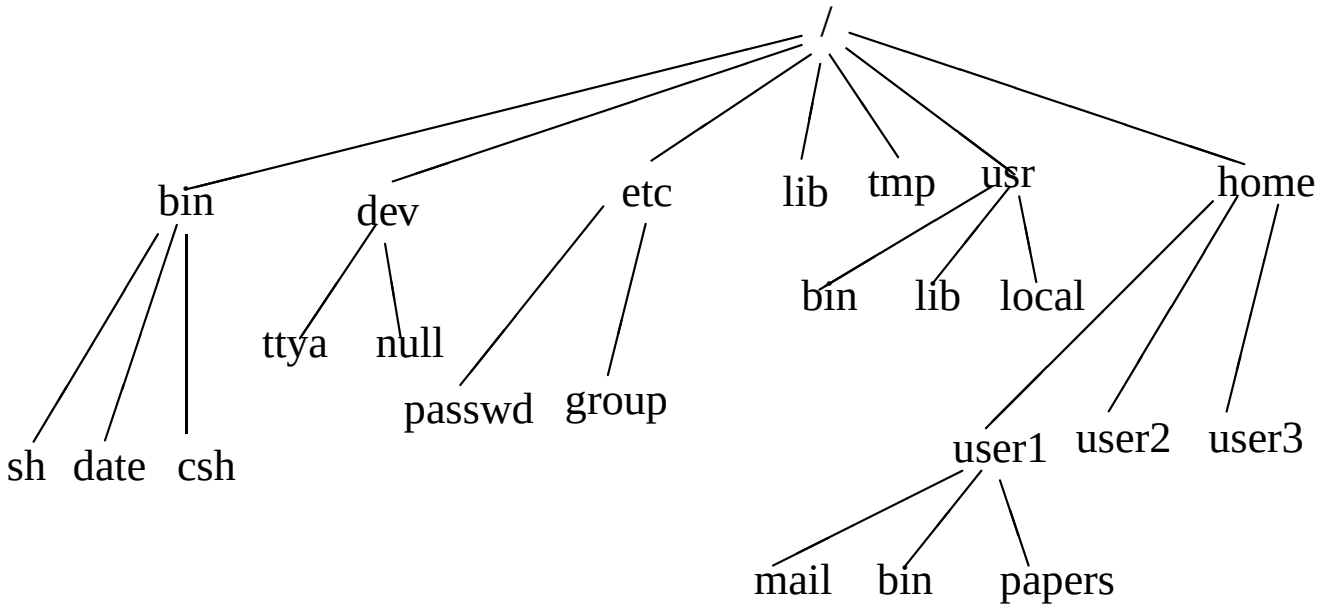
- File system: è la struttura che permette la memorizzazione e l'organizzazione dei dati nel sistema.

- È estremamente importante perché UNIX è un sistema orientato ai file nel senso che molte operazioni vengono effettuate tramite essi (accesso a periferiche e comunicazione tra processi).

- Un nome nel file system può riferirsi a:
  - file dati (che può anche essere una directory)
  - dispositivo di input/output (disco, nastro, stampante, . . .)
  - memoria interna
  - informazioni sul sistema hardware e software

File system / A

- Un file può essere visto come una sequenza di byte, indipendentemente dal tipo di file (file dati, dispositivo, . . .)
- Una directory può essere vista come un contenitore di file o di directory, si ha quindi una struttura gerarchica (ad albero. . . se si trascurano i link).



Organizzazione del file system

Per tradizione esistono alcune directory che nel file system assumono un ruolo particolare.

`\`	è la radice della gerarchia;
`bin`	comandi più importanti del sistema;
`sbin`	comandi che riguardano alcune importanti funzioni di gestione del sistema (`dump`, `shutdown`, ecc.);
`dev`	dispositivi di input/output che possono essere visti dalla macchina (dischi, nastri, ecc.);
`etc`	file di configurazione del sistema (`passwd`, `fstab`, `lilo.conf`, ecc.);
`lib`	librerie condivise per ridurre la dimensione dei file eseguibili;
`mnt`	mount point dei dispositivi esterni;
`var`	contiene file di dimensione variabile (es. in `log` ci sono i file di log, in `spool` ci sono file usati da altri programmi (posta in ingresso, stampe, `...`);
`usr`	contiene i file non essenziali ma utili (es: le sottodirectory `bin` con `emacs`, `gzip`, ... `X11R6` con il sistema XWindow, `man` con tutti i manuali, `src` con tutti i sorgenti);
`proc`	è un file system virtuale che contiene tutte le informazioni di sistema, dai processi alla cpu (`cpuinfo`, `version`, `...`).
`home`	contiene le directory personali di tutti gli utenti.

Working e Home directory

- In UNIX ogni processo ha associata una working directory (directory di lavoro).
  Per conoscere la directory corrente si può utilizzare il comando `pwd`

```
$ pwd
  /home/rossi
$
```

- Ogni utente “username” ha una home directory — la parte del file system riservata per suoi file. È indicata con

```
~username
```

- Dopo il login, l’utente si trova automaticamente nella propria home directory, da dove comincia a lavorare.

Navigare nel file system - cd

- Ci si può spostare all'interno del file system e quindi modificare la working directory con il comando `cd`.

  `cd [<dir>]`

  Il parametro `<dir>` è opzionale. Se non viene indicato, il comando porta nella home directory. Es:

```
/home/rossi$ cd /bin
/bin$ cd
/home/rossi$
```

- Per la navigazione risultano utili le directory: "." (working directory) e ".." (directory padre). Es:

```
/home/rossi$ cd ..
/home$ cd .
/home$
```

Pathname

- Specifica la locazione di un file all'interno di un file system.
- Pathname assoluto: dice come raggiungere un file partendo dalla radice del file system; comincia sempre con / (slash).

```
/usr/local/doc/training/sample
```

- Pathname relativo: dice come raggiungere un file a partire dalla working directory o dalla home directory di un utente.

```
training/sample
../bin
~/projects/report
~rossi/projects/report
```

- Esempio: Per spostarsi da `/home/rossi` a `/home/bianchi`:

```
cd ../bianchi          - relative pathname
cd /home/bianchi       - absolute pathname
```

Visualizzare una directory

Il comando `ls` permette di visualizzare il contenuto di una (o più) directory.

```
ls [-alsFR] [<dir1> ... <dirn>]
```

Alcune opzioni:

- `-a` visualizza anche i file nascosti (che iniziano con “.”)
- `-l` visualizza informazione estesa sui file
- `-s` visualizza la dimensione
- `-F` aggiunge un carattere finale al nome che ne denota il tipo:
  “/” per directory, “*” per eseguibile, “@” per link,
  “=” per socket, nessuno per i file ordinari).
- `-R` mostra ricorsivamente le sottodirectory

Se non viene specificata nessuna directory, `ls` si riferisce alla directory corrente.

Creare e rimuovere directory

Tramite i comandi `mkdir` e `rmdir` si possono creare e cancellare directory.

```
mkdir <dir1> [<dir2> ... <dirn>]
rmdir <dir1> [<dir2> ... <dirn>]
```

Ognuno degli argomenti è considerato una directory da creare/cancellare.
Es.:

```
/home/rossi$ mkdir report-2000 report-2001
/home/rossi$ ls -F
report-2000/  report-2001/
/home/rossi$ rmdir  report-2000 report-2002
rmdir:   report-2002:   No such file or directory
/home/rossi/$ ls -F
report-2001/
```

Una directory può essere rimossa solo se è vuota:

```
/home/rossi$ rmdir ..
rmdir:   '..': Directory not-empty
```

Copiare, spostare, rimuovere file

Tramite il comando `cp` si possono copiare file:

```
cp -i <oldname> <newname>
cp -ir <name1> [<name2> ... <namen>] <destdir>
```

Prima forma: crea una copia `<newname>` del file `<oldname>`. Con l'opzione "-i" il sistema chiede conferma prima di sovrascrivere un file. Es:

```
/home/rossi$ ls
chap1
/home/rossi$ cp chap1 chap2; ls
chap1  chap2
```

Seconda forma: il comando ha più di due argomenti. In questo caso l'ultimo `<destdir>` deve essere una directory, dove vengono inseriti i file `<name1> ... <namen>`. L'opzione "-r" permette di copiare ricorsivamente le sottodirectory.

Copiare, spostare, rimuovere file / A

- Il comando `mv` per spostare/rinominare file ha sintassi e funzionamento simile a `cp`, se non per il fatto che il file originale scompare

```
mv -i <oldname> <newname>
mv -i <name1> [<name2> ... <namen>] <dest>
```

  Copia + rimozione? Move tra fs diversi?

- Il comando per la cancellazione di file è `rm` ed ha sintassi

```
rm -ir <name1> <name2> ... <namen>
```

  Particolarmente interessante l'opzione "-r" che permette di rimuovere una directory con tutto il corrispondente "sottoalbero".

I Metacaratteri

Accade spesso di voler operare su più file contemporaneamente. Ad es. supponiamo di voler copiare tutti i file che cominciano con `data` nella directory `~/backup`).

UNIX fornisce un modo molto efficiente per ottenere questo scopo:

```
/home/rossi/report$ ls -F
data-new   data1   data2   inittab2   pippo
/home/rossi/report$ mkdir ~/backup
/home/rossi/report$ cp data* ~/backup
/home/rossi/report$ ls -F ~/backup
data-new   data1   data2
/home/rossi/report$
```

(Cosa farebbe `cp d*2 ~/backup`?)

I Metacaratteri / A

Ecco un elenco dei metacaratteri:

`?`	ogni carattere
`*`	ogni stringa di caratteri
`[set]`	ogni carattere in set
`[!set]`	ogni carattere non in set

dove set può essere una lista di caratteri (es. `[abc]`) oppure un intervallo (es. `[a-c]`) o una combinazione dei due.

Se la shell trova un'espressione contenente metacaratteri in una linea di comando, la espande con tutti i nomi di file che "soddisfano" (match) l'espressione (espansione di percorso).

I Metacaratteri / B

Per capire meglio come funziona l'espansione di percorso introduciamo un nuovo comando, `echo`. È un comando estremamente semplice: rimanda sullo schermo ogni parametro.

```
/home/rossi$ echo Ciao!
Ciao!
/home/rossi$ cd report
/home/rossi/report$ ls -F
data-new   data1   data2   inittab   pippo   pippo2
/home/rossi/report$ echo data*
data-new   data1   data2
/home/rossi/report$ echo data?
data1   data2
/home/rossi/report$ echo [!id]*
pippo   pippo2
/home/rossi/report$
```

(Cosa accade se si esegue `cd *`?)

Caratteri speciali

```
NEWLINE - initiates command execution
      ; - separates commands on same line
    ( ) - groups commands or identifies a function
      & - executes a command in the background
      | - pipe
      > - redirects standard output
     >> - appends standard output
      < - redirects standard input
      * - wildcard for any number of characters in a file name
      ? - wildcard for a single character in a file name
       - quotes the following character
      ’ - quotes a string preventing all substitutions
      " - quotes a string allowing variable and command substitution
      ‘ - performs command substitution
    [ ] - denotes a character class in a file name
      $ - references a variable
      - command grouping within a function
      . - executes a command (if at beginning of line)
      # - begins a comment
      : - null command
```

Accesso ai file

Informazioni sui file

- Time stamp — Ad ogni file sono associate tre date: cambiamento attributi, la data di ultima modifica e ultimo accesso.
- User (Proprietario) — Ogni file in UNIX è di proprietà di un utente del sistema.
- Group (Gruppo) — Ogni file è associato ad un gruppo di utenti: quello più comune per i file utente è chiamato `users`, ed è generalmente condiviso da tutti gli account utente del sistema.
- Permessi — Ad ogni file sono associati dei permessi (chiamati anche "privilegi"), che specificano chi e in che modo (lettura, modifica, esecuzione) può accedere a quel file. Ognuno di questi permessi può essere impostato separatamente per il proprietario, il gruppo e tutti gli altri utenti (others).

Modificare le informazioni sui file

- Le informazioni sui file possono essere visualizzate tramite il comando `ls -l`. Per esempio

```
rossi@linuslab:/home/rossi/report > ls -l
total 2
-rw-r--r--  1 rossi users      15 Mar 9 12:40 pippo
-rw-r--r--  1 rossi users     155 Mar 9 12:41 pluto
drwxr-xr-x  2 rossi users    1024 Mar 7 17:33 OLD
```

- In particolare, ogni linea include:
  - il tipo del file (primo carattere)
  - i permessi di accesso (caratteri 2-10):

    2-4: owner 5-7: group 8-10: other
  - il proprietario
  - il gruppo

Permessi di accesso

- I permessi possono (sostanzialmente) essere:
  `r` = permesso di lettura
  `w` = permesso di scrittura
  `x` = permesso di esecuzione
  `-` = mancanza di permesso
- Alcuni esempi

```
-rw-------   1 rossi users    3287 Apr 8 12:10 file1
-rw-r--r--   1 rossi users   13297 Apr 8 12:11 file2
-rwxr-xr-x   1 rossi users    4133 Apr 8 12:10 myprog
drwxr-x---   1 rossi users    1024 Jun 7 10:00 SCCS
```

- Il permesso `x` su una directory consente di “attraversarla”.
- C’è una forma di dipendenza: per accedere ad un file, oltre ai diritti necessari per l’operazione desiderata, occorre il permesso `x` su ogni directory che compare nel pathname.

Cambiare i permessi con `chmod`

```
chmod <triads> <filename>*
```

dove `triads` sono della forma who action permission

who			action			permission		
u	=	user	+	=	aggiunge	r	=	read
g	=	group	-	=	rimuove	w	=	write
o	=	other	=	=	assegna	x	=	execute
a	=	all						
	=	user						

Cambiare i permessi con `chmod` / A

- Alcuni esempi:

```
chmod a+r sample.c
chmod o-r sample.c
chmod og+rx prog*
chmod g=rw rep*
chmod +w *
```

- L'opzione `-R` cambia (ricorsivamente) i permessi di una directory e di tutti i file e sottodirectory di quella directory.

chmod “numerico”

- I permessi di accesso `rwx` possono essere rappresentati come un numero ottale. Es. `r-x` diviene 101 e quindi 5_8.
- Il permesso `r` è rappresentato da 4, `w` da 2 e `x` da 1. Per ciascuna categoria di utenti (proprietario, gruppo, altri) questi valori possono essere sommati. Quindi, si ha

```
0 = ---              4 = r--
1 = --x              5 = r-x (4+1)
2 = -w-              6 = rw- (4+2)
3 = -wx (1+2)        7 = rwx (4+2+1)
```

- I permessi di accesso possono allora essere espressi con 3 cifre ottali.

```
                     user   group   others
chmod 640 file1      rw-    r--     ---
chmod 754 file1      rwx    r-x     r--
chmod 664 file1      rw-    rw-     r--
```

Stabilire i permessi con `umask`

```
umask <mask>
```

- Può essere utilizzato per assegnare i permessi di default.
- Tipicamente è usato in uno dei file di startup (per esempio, `.login`, `.cshrc`, `.profile`) ma può essere usato anche in modo interattivo.
- Specifiche ottali (anche simboliche in bash). Diversamente da `chmod`, rappresentano i permessi da "mascherare" (cioè, da rimuovere).

```
Octal number    Access permissions given
   0            rwx    read, write and execute
   1            rw-    read and write
   2            r-x    read and execute
   3            r--    read only
   4            -wx    write and execute
   5            -w-    write only
   6            --x    execute only
   7            ---    no permissions
```

Stabilire i permessi con `umask` / A

- Il sistema assegna di default i permessi `666` per i file e `777` per le directory.
- Alcuni esempi:
  - `umask 077`
    Sottrae 077 dai permessi di default del sistema per i file e le directory. Il risultato è che i permessi di default diventano 600 per i file (`rw-------`) e 700 (`rwx------`) per le directory.
  - `umask 002`
    Sottrae 002 dai permessi di default del sistema per i file e le directory. Il risultato è che i permessi di default diventano 664 per i file (`rw-rw-r--`) e 775 (`rwxrwxr-x`) per le directory.
- `!` `umask` sottrae i valori passati come parametri sempre ai valori di default (`666` e `777`), anche se usato più volte consecutivamente.

Cambiare proprietario e gruppo

- Per cambiare il proprietario (owner) di un file

```
chown <new_owner> <filename>
```

  Nella maggior parte dei sistemi UNIX, per modificare il proprietario di un file bisogna essere un super-user (ad es. l'amministratore del sistema).

- Il proprietario di un file può cambiare il gruppo (group) del file con un qualsiasi altro gruppo a cui l'utente stesso appartenga.

```
chgrp <new_group> <filename>
```

- Per entrambi i comandi al'opzione `-R` permette di propagare ricorsivamente il cambiamento su tutta una directory.

Visualizzare i file

Mostrare il contenuto di un file

- Il comando `cat` concatena e visualizza sullo schermo (*standard output*) il contenuto di uno o più file.

```
cat [<filename1> ...<filenamen>]
```

- Il comando `echo` visualizza sullo schermo l'argomento che gli viene passato (a cui aggiunge un carattere newline, a meno che non si specifichi l'opzione `-n`).

```
echo [-n] <string>
```

- I comandi `more`, `less` e `pg` (potrebbero non essere tutti disponibili) mostrano il contenuto di un file una schermata alla volta. Inoltre, permettono di tornare a pagine precedenti, di effettuare ricerche, ...

Il comando `less`

Una volta digitato

```
less <filename>
```

si può navigare nel file. Ecco alcune opzioni

- ↑, ↓ scorre su e giù di una riga;
- ←, → scorre a dx e sx di una schermata;
- `/<pattern>` cerca `<pattern>` in avanti;
- `?<pattern>` cerca <pattern> indietro;
- `<n>g` va alla riga `<n>`;
- `g`, `G` va ad inizio/fine file;
- `q` esce e torna al prompt;
- e molte altre (`h` dà l'help in linea).

Mostrare parte di un file

- Il comando `head` mostra le prime 10 linee di un file. Se si usa l'opzione `-n`, `head` mostra le prime n linee.

```
head [-n] file
```

- Il comando `tail` mostra le ultime 10 linee di un file. Se si usa l'opzione `-n`, `tail` mostra le ultime n linee.

```
tail [-n] file
```

Cercare file e comandi

Cercare un file

Il comando `find` permette di cercare un file che soddisfi un dato pattern.

```
find <pathname> -name <filename> -type <t> -print
```

dove

- pathname indica la directory da cui deve iniziare la ricerca. La ricerca continuerà in ogni sottodirectory.
- `<filename>` è il nome del file da cercare; può essere anche un pattern costruito con metacaratteri.
- L'opzione `-print` viene usata per mostrare i risultati della ricerca.
- Vediamo alcuni esempi d'uso del comando (con ulteriori opzioni):

```
find .  -name mtg_jan92 -print
find ~/ -name README -print
find .  -name '*.ps' -print
find /usr/local -name gnu -type d -exec ls {} \;
```

Cercare un file / A

Un problema di `find` è la sua pesantezza computazionale.

Una soluzione è data dal comando `locate`, che basandosi su di un database, periodicamente aggiornato (con `updatedb`), risulta molto più efficiente

```
locate <modello>
```

Elenca i file che nelle basi di dati corrispondono a `<modello>`.

Esempio:

```
/home/rossi$ locate astri
/usr/include/g++-3/std/bastring.cc
/usr/include/g++-3/std/bastring.h
/home/rossi$
```

Cercare i programmi

Per cercare la locazione di un programma (tra i sorgenti, tra i file eseguibili e nelle pagine del manuale in linea):

```
whereis [-bms] <command>
```

- Alcune opzioni utili sono:
  - `-b` solo file binari
  - `-m` solo pagine dei manuali
  - `-s` solo file sorgenti
- Esempio

```
/home/rossi$ whereis -bms mail
mail:/bin/mail /usr/bin/mail /etc/mail.rc /etc/mail
/home/rossi$
```

Cercare un comando

Accade talvolta di essere a conoscenza dell'esistenza di un comando e di voler sapere quale file sarà eseguito quando il comando viene invocato.

```
which <command>
```

fornisce

- il pathname completo del file eseguibile
- l'alias corrispondente a `<command>`

Un comando (builtin nella bash) simile, ma più completo è `type`

```
type [-all -path] <command>
```

indica come la shell interpreta `<command>`, specificandone la natura (alias, funzione, file eseguibile, builtin, parola chiave della shell) e la definizione. Con l'opzione `-all` vengono fornite tutte le definizioni di `<command>`, mentre l'opzione `-path` restringe la ricerca ai file eseguibili.

Gestire archivi

Il comando `tar` permette di archiviare insiemi di file (in realtà parti del filesystem, con informazioni sulla gerarchia delle directory) in un unico file.

```
tar [-ctvx] [-f <file>.tar] [<directory/file>]
```

- Alcune opzioni comuni sono

`-c`	crea un archivio
`-t`	mostra il contenuto di un archivio
`-x`	estrae da un archivio
`-f <file>.tar`	specifica il nome del file di archivio
`-v`	fornisce informazioni durante l'esecuzione

- Alcuni esempi

```
tar -cf logfile.tar mylogs/logs.*
tar -tf logfile.tar
tar -xf logfile.tar
```

Comprimere file

Sono presenti alcuni tool che permettono di ridurre la dimensione di file con algoritmi di compressione (che sfruttando la ridondanza dei dati danno risultati migliori su file di testo):

```
compress [opt] <file>
uncompress [opt] <file>.Z
```

Ad esempio

```
compress logs.*
uncompress logfile.tar.Z
```

Oppure si può usare `gzip` (Lempel-Ziv coding LZ77) o `bzip2` (Burrows-Wheeler block sorting text compression algorithm):

```
gzip [opt] <file>   gunzip [opt] <file>.gz
```

Ad esempio

```
gunzip logfile.tar.gz
gzip logs.*
```

Esercizio

Scrivere un comando che crea una sottodirectory della directory corrente, con nome `text-files`. Quindi copia in questa directory tutti i file con estensione `txt` contenuti nella propria home directory. Se la directory `text-files` è già presente rimuove da questa i file il cui nome non inizia con almeno due caratteri alfanumerico o che non hanno estensione `txt`.

[Sugg.: Usare la concatenazione di comandi, il comando `find` e i metacaratteri.]

Comandi di stampa

Alcuni comandi per la stampa dei file e la gestione della coda di stampa

- UNIX BSD e Linux

```
lpr [opt] <filename>
lpq [opt] [<job#>]  [<username>]
lprm [opt] [<job#>] [<username>]
```

  Un'opzione usata frequentemente è `-P <printername>`.

- UNIX System V

```
lp [opt] <filename>
lpstat [opt]
cancel [<requestID>] <printer>
```