

SCHEMI BASH

I file sono riferiti con il PATHNAME:

- assoluto: `/home/bianchi/progetto/a`
- relativo: `progetto/a` ~pwd: `/home/bianchi`

NOTA: `dir1/*` rappresenta il nome di tutti i file dentro la cartella `dir1`

SHELL & SESSIONE

bash e altri tipi di SHELL

logout

uscire dalla shell: `$ exit`

pulire lo schermo: `$ clear`

HISTORY

Come vedere lista comandi: `$ history` → [num. evento] [evento = comando]

rieseguire ultimo comando: `$!!`

eseguire un comando con numero evento: `$!num.evento`

ricercare stringa a inizio comando: `$![evento]` → es `$!ls` cerca nella history ultimo `ls`

ricercare stringa in un punto qualsiasi: `$!?[stringa]?` → `$!?ls?` cerca `ls` in tutto il comando

ricerca di un evento e sostituzione argomenti

`$![evento]:s/[stringa1]/[stringa2]/` `$ [evento] [stringa1]` → `$ [evento] [stringa2]`

creare un alias: `$ alias nome_alias='[comando_alias]'`

rimuovere alias: `$ unalias nome_alias`

sostituzione eventi: `:s / [stringa da cercare] / [nuova stringa]`

METACARATTERI

`*` : stringa di 0 o più caratteri

`?` : singolo carattere

`[]` : singolo carattere tra quelli elencati (valgono intervalli es. `a-zA-Z`)

`{}` : sequenza di stringe

`>` : redirezione output (sovrascrive)

`>>` : redirezione output append (non sovrascrive)

`<` : redirezione input (input non da tastiera)

`<<` : redirezione input dalla linea di comando

`2>` : redirezione messaggi errore

`|` : pipe, compone comandi a cascata, eseguiti in parallelo

`;` : sequenza di comandi

`||` : esecuzione condizionale, esegue se il precedente fallisce

`&&` : esecuzione condizionale, esegue se il precedente termina con successo

`(...)` : raggruppamento comandi

`!` : ripetizione comandi memorizzati history list

QUOTING

`\` : inibisce il metacarattere successivo

`'` : inibisce metacaratteri racchiusi

`" "` : inibisce alcuni metacaratteri

CARATTERI SPECIALI

`\n` : newline

`\t` : tab

FILE E DIRECTORY

cartella corrente: `$ pwd`

spostarsi cartella: `$ cd [pathname]` (nota: per accedere serve permesso esecuzione)

spostarsi home: `$ cd` (senza argomenti)

spostarsi nella dir madre: `$ cd ..`

creare cartella: `$ mkdir [nome]`

creare più cartelle: `$ mkdir [path]/{dir1, dir2, dir3}` → crea 3 cartelle dentro nel percorso

rimuovere cartella: `$ rmdir [nome]`

rimuovere file: `$ rm [argomenti] f1`

visualizzare elenco file:

```
$ls [argomenti] [pathname]
```

argomenti:

`-l` :long

`-a` :nascosti

`-al` :nascosti & long

`-t` :in ordine di ultima modifica

`-S` :in ordine di dimensione decrescente

`-r` :in ordine inverso

`-R` :sottocartelle

`-d` :lista le cartelle in sé, non le sottocartelle

visualizzare elenco file per estensione: `$ ls [arg] *.estensione`

visualizzazione `$ ls -l`

[tipofile] [permessi] [num hardlink] [proprietario] [nomegruppo] [dim] [data ultima mod] [nomefile]

```
-rwxrwxrwx 1 root root 5395 Jul 13 1998 ciao.txt
```

tipofile: - file, d directory, l link, b block device, c character device

cambiare i permessi: `$ chmod [arg] [file]`

metodo ottale: `$ chmod 744 file` → `$ chmod 111 100 100 f1 [rwxr--r--]`

metodo classico: `$ chmod u=rwx go=r f1` u=owner, g=group, o=world

= imposta i comandi esattamente come seguono `$ chmod g=r` → r--

+ aggiunge permessi che seguono `$ chmod g+r f1` (-W- → rw-)

- toglie i permessi che seguono `$ chmod g-r f1` (rw- → -W-)

copiare un file f1 in f2: `$ cp [f1] [f2]` → f2 può non esistere

`-r` : ricorsivo

copiare: `$ cp [options]`

copiare più file in una dir: `$ cp [f1] ... [fn] [dir1]` → obbligo cartella ultimo argomento

spostare/rinominare file: `$ mv f1 f2`

spostare più file: `$ mv [f1] ... [f2] [dir1]`

aggiorna data ultima modifica: `$ touch f1` → se f1 non esiste viene creato (salvo arg `-c` o `-h`)

confronto file:

`$ cmp f1 f2` → primo byte e numero di linea in cui f1 e f2 differiscono

`$ diff f1 f2` → lista di cambiamenti da apportare in f1 per renderlo come f2

ricercare un file:

`$ find [pathnames] [expression]`

attraversa ricorsivamente le directory in [pathnames] applicando le regole [expression]

expression può essere: *opzione, condizione, azione*

esempi:

`$ find . -name '*.c' -print` , ricerca ricorsivamente a partire dalla directory corrente tutti i file .c e li stampa

`$ find . -name '*.bak' -ls -exec rm {} \;` cerca ricorsivamente .bak li stampa con attributi (-ls) e li cancella

`$ find /etc -type d -print`

file system in breve

L'informazione è memorizzata in *dischi fissi* suddiviso in una o più *partizioni*, ognuna delle quali può contenere un file system con una propria *top level directory*. In UNIX abbiamo un'unica struttura gerarchica.

Le opzioni sui filesystem da montare al boot sono in `/etc/fstab` . ,mentre il comando per montare è

`mount <file speciale><mount point>` . Solo root può usarlo, `mount` senza argomenti indica i filesystem in uso nel sistema.

Controllo della quantità di spazio: `df`

Controllo quantità di spazio: `du [cartella]` , con argomento `-s` per vedere soltanto il totale

PROCESSI E JOB

stampare informazioni riguardo user che sono loggati correntemente: `$ who [opzioni]`

vedere i processi dell'utente associati al terminale corrente:

```
$ ps [argomenti]
```

- argomenti:*
- a : tutti i processi di un terminale
 - f : full listing
 - e : anche processi non associati a un terminale
 - l : long listing
 - U :
 - no-header : stampa senza l'header

F	S	UID	PID	PPID	C	PRI	N	ADDR	SZ	WCHAN	TTY	TIME	CMD
8	S	0	1	0	0	41	20	?	100	?	?	0:03	init
8	S	140	12999	12997	0	56	20	?	278	?	pts/12	0:00	tcsh

- F: flag obsoleti
- S: stato del processo (T=stopped, R=ready, S=sleep)
- UID: utente
- PID: identificativo processo
- PPID:
- C:
- PRI: priorità
- NI: nice value
- ADDR: indirizzo in memoria
- SZ: memopria virtuale utilizzata
- WCHAN: evento su cui il processo è sleeping
- TTY: terminale

- terminare un processo: `$ kill PID_processo`
- processo sigkill: `$ kill -s kill PID_processo`
- aprire un processo in background: `$(comando) &`
- vedere i job in esecuzione: `$ jobs`
- resume del job in foreground: `$ fg`
- resume del job in background: `$ bg`
- terminare un job: `$ kill %numerojob`
- informazioni sulla memoria: `$ top`
- informazioni su spazio occupato nel disco: `$ df`
- blocchi memoria occupati da una cartella: `$ du [cartella]`

VISUALIZZAZIONE

visualizzare un file

`$ cat [f1]` → visualizzo l'intero file

`$ more [f1]` → scorro il testo

`$ tail [-n] [f1]` → ultime n righe [default n = 10]

`$ head [-n] [f1]` → prime n righe [default n = 10]

`$ echo` : ripete il segnale in input. Opzioni `-n` se non si vuole andare a capo, `-e` per abilitare backslash

`$ read` : legge una linea dallo standard input e la divide in campi, oppure da un file tramite `-u file`

`$ basename` : restituisce il nome di un file senza il path

INODE E LINK

creazione hardlink: `$ln [file1] [link1]`

creazione link simbolico `ln -s [file1] [link1]`

Posso creare link simbolici che puntano a link, creando catene fino a un massimo di 6 link simbolici.

FILTRO

numero di linee,parole,caratteri di un file:

```
$ wc [opzioni] [file]
```

opzioni:

`-c` : numero di byte

`-m` : numero di caratteri

`-l` : numero di linee (line)

`-w` : numero di parole (word)

[default] : linee, parole, byte

segnalare o cancellare linee ripetute in un file

```
$ uniq [file]
```

opzioni:

`-c` : precede ogni riga con un conteggio delle ripetizioni adiacenti

`-d` : visualizza righe ripetute

`-u` : visualizza righe non ripetute

restituire linee di un input che contengono un determinato pattern:

```
grep [options] pattern [filename] (General Regular Expression Parser)
```

```
fgrep [options] pattern [filename] (Fixed General Expression Parser)
```

```
egrep [options] pattern [filename] (Extended General Expression Parser)
```

opzioni:

- i : ignora maiuscole/minuscole
- l : lista dei file che contengono il pattern
- n : linee in output precedute dal numero di linea
- v : linee che NON contengono il pattern
- w : linee che contengono il pattern come parola completa
- x : linee che coincidono perfettamente con il pattern

esempi:

METACARATTERI DELLE ESPRESSIONI REGOLARI

Due tipi di sequenze di caratteri:

- B (basic) utilizzabili sia in `grep` che in `egrep`
- E (extended) utilizzabili solo in `egrep` oppure in `grep (-e)`

tipo B:

`^` : inizio della linea

`$` : fine della linea

`\<` : inizio di una parola

`\>` : fine di una parola

`.` : singolo carattere (qualsiasi)

`[str]` : un qualunque carattere in str

`[^str]` : un qualunque carattere non in str

`[a-z]` : un qualunque carattere tra a e z

`\` : inibisce carattere successivo

`*` : zero o più ripetizioni dell'elemento precedente

tipo E:

`+` : una o più ripetizioni dell'elemento precedente

`?` : zero o una ripetizioni dell'elemento precedente

`{j,k}` : un numero di ripetizioni compreso tra j e k dell'elemento precedente

`s|t` : l'elemento s oppure l'elemento t

`(exp)` : raggruppamento di exp come singolo elemento

ordinare linee input:

`$ sort` (default: alfabetico)

prende in input delle linee di testo, le ordina secondo le opzioni e le invia in output

opzioni:

- b ignora spazi chiavi di ordinamento
- f ignora distinzione maiuscole/minuscole
- n considera numerica la chiave di ordinamento
- r ordina in modo decrescente

- o [f1] invia output al file f1
- t [s] usa s come separatore di campo
- k s1,s2 usa i campi da s1 a s2 come chiavi di ordinamento, i successivi in caso di pareggio
- s rende stabile il confronto, senza passare ai successivi in caso di pareggio

conversione di carattere:

\$ tr (soltanto standard I/O, necessità di pipe/ridirezioni in caso contrario)

\$ tr [stringa1] [stringa2] → i caratteri in stringa1 vengono sostituiti con caratteri corrispondenti in stringa2

opzioni:

- c complemento
- s squeeze, compressione
- d cancella caratteri

estrarre colonne specifiche da linee di testo in input:

\$ cut

opzioni:

- d separatore (se non specificato <Tab>)
- f campo da estrarre (partono da 1)

es. \$ cut -d: -f2

combinare due righe corrispondenti di due file:

\$ paste f1 f2 (delimitatore default : <tab>)

editare testo passato da un comando all'altro in una pipeline:

\$ sed [actions] [files] nota: può prendere in input anche file

il comportamento standard è stampare in standard output le linee in input

salvo specifiche di indirizzo applica l'azione a tutte le linee in input

gli indirizzi di linea possono essere specificati come numeri o espressioni regolari.

opzioni:

- e in caso di più azioni, precede le azioni
- f specifica file da cui leggere le azioni da fare

sostituzione testo con sed:

\$ sed s/[expr]/[new]/[flags]

s: substitute, *expr*: stringa da cercare, *new*: stringa da sostituire

flag possibili:

- num tra 1 e 9 (quale occorrenza di *expr* sostituita, default 1 = prima)
- g : ogni occorrenza sostituita

p : linea corrente viene stampata a video in caso di sostituzione
n : silent mode, senza emettere output
w [**file1**] : in caso di sostituzione la linea corrente viene accodata in file1

SCRIPT

programma interpretato dalla shell scritto in comandi UNIX. Viene eseguito in una sottoshell della shell corrente.

set -x : visualizza i comandi nel momento in cui li esegue

set -v : visualizza i comandi nel momento in cui li legge

set - : annulla gli effetti di **set -x**, **set -v**

assegnamento: **variabile=valore** (nota, = senza spazi è assegnazione)

accesso a una variabile **\$variabile**

le variabili sono locali alla shell o allo script in cui sono definite

promuovere a variabile globale (*variabile d'ambiente*): **export**

variabili d'ambiente:

PS1 : prompt primario

PS2 : prompt secondario

PWD : pathname assoluto directory corrente

UID : ID user corrente

PATH : lista di pathname in cui la shell cerca i comandi

HOME : pathname assoluto della home directory

variabili speciali (parametro): **\$1,\$2,...,\$9** associate al primo, ... , nono parametro *passati su linea di comando*

aumentare numero parametri con shift a sx: **shift [n]**

variabili di stato automatiche, gestiscono lo stato

variabili di stato:

\$? : exit status ultimo comando (0 successo, errore altrimenti)

\$\$: PID shell corrente (uso: nomi file temporanei unici tra utenti e shell diverse)

\$! : PID ultimo comando in background

\$- : opzioni della shell corrente

\$# : numero parametri forniti allo script su linea di comando

\$* : lista dei parametri passati allo script su linea di comando

\$@ : lista dei parametri passati allo script su linea di comando

CONTROLLO DI FLUSSO

COSTRUTTO IF


```
if condition_command (condition exit status)
then (exit 0)
    true_commands
else (exit != 0)
    false_commands
fi
```

in caso non sia valutabile l'exit status

```
test [expression] (se vera exit = 0, altrimenti 1 )
```

- espressioni che controllano attributi di un file *f*

```
-e [f] : 0 se f esiste
-f [f] : 0 se f esiste ed è un file ordinario
-d [f] : 0 se f esiste ed è una dir
-r [f] : 0 se f esiste ed è leggibile
-w [f] : 0 se f esiste ed è scrivibile
-x [f] : 0 se f esiste ed è eseguibile
```

- espressioni su *stringhe*

```
-z str : 0 se str è lunga 0
-n str : 0 se str non è lunga 0
str1 = str2 : 0 se str1 è uguale a str2 (nota = con spazi è confronto)
str1 != str2 : 0 se str1 è diversa da str2
```

- espressioni su *valori numerici*:

```
num1 -eq num2 : 0 se uguali
num1 -ne num2 : 0 se diversi
num1 -lt num2 : 0 se num1 < num2
num1 -gt num2 : 0 se num1 > num2
num1 -le num2 : 0 se num1 <= num2
num1 -ge num2 : 0 se num1 >= num2
```

- *espressioni composte*:

```
exp1 -a exp2 : 0 se entrambe vedere (and)
exp1 -o exp2 : 0 se è vera exp1 o exp2 (or)
! exp : 0 se exp non è vera
( exp ) : per cambiare ordine valutazione degli operatori (è necessario quoting)
```

costruzioni numeri complesse: `$(espressione)` Nota: le quadre vanno messe

CICLO WHILE (finché vero continuo)

```
while condition_command
do
    commands
done
```

finché la condition_command è vera vengono eseguiti i comandi

CICLO UNTIL (finché falso continuo)

```
until condition_command
do
    commands
done
```

finché la condition_command è falsa vengono eseguiti i comandi

CICLO FOR

```
for var in wordlist
do
    commands
done
```

CASE SELECTION

```
case string in
expression_1)
    commands_1
    ;;
expression_2)
    commands_2
    ;;
...
*)
default_commands
;;
esac
```

COMMAND SUBSTITUTION

sostituire a un comando o pipeline quanto stampato sullo standard output dal comando stesso prevede uso dei `` backquote

esempio :

```
> date
Tue Nov 19 17:50:10 2002
> vardata=`date`
> echo $vardata
Tue Nov 19 17:51:28 2002
```