

Shell Bash

Università di Modena e Reggio Emilia

Prof. Nicola Bicocchi (nicola.bicocchi@unimore.it)



Utilità

Builtins

Bash, come **interprete dei comandi**, esegue il contenuto di file binari presenti nel filesystem (es. /bin/ls)

```
$ ls  
$ which ls  
/bin/ls  
$ which which  
/usr/bin/which
```

Builtins

Esistono particolari comandi, detti **bultins**, che non provengono dall'esecuzione di un binario esterno ma sono **implementati all'interno della shell**. Nel loro caso, *\$ which comando* non ritorna un percorso perchè il binario non esiste! Ad esempio:

```
$ cd  
$ history  
$ logout
```

https://www.gnu.org/software/bash/manual/html_node/Bash-Builtins.html



history

```
$ history
```

```
1 uname -a  
2 clear  
3 exit  
4 ls
```

```
$ !!      (esegue ultimo comando)
```

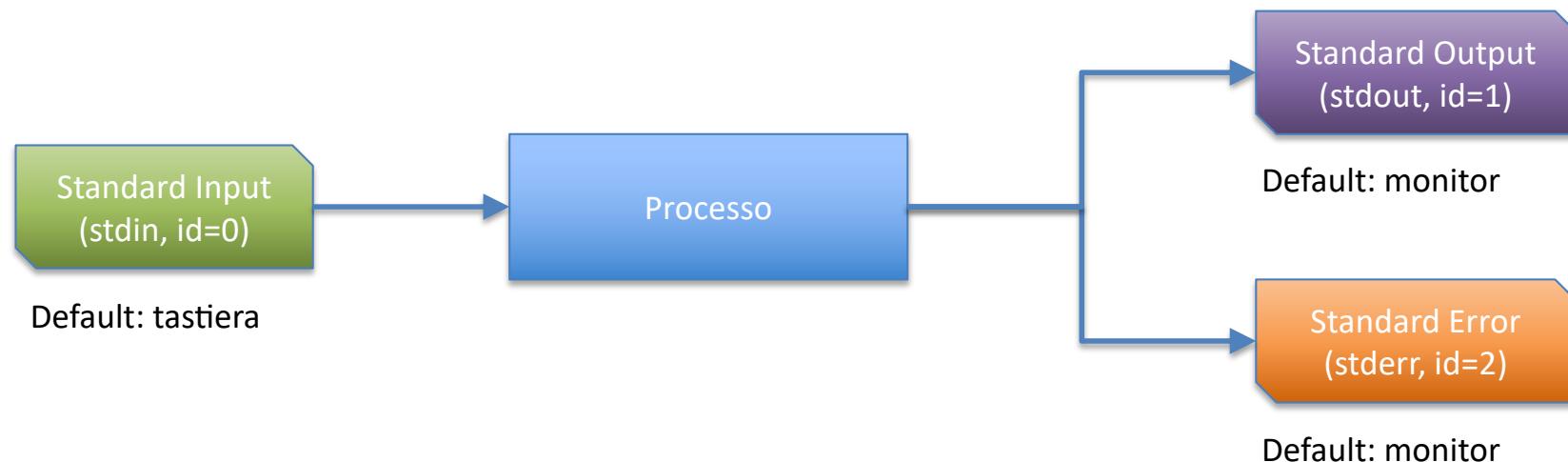
```
$ !2      (esegue comando #2)
```

Freccia su-giù, ctrl-r, tab

- Tasti freccia (su e giù) consentono di spostarsi all'interno della lista dei comandi precedenti (lo stesso elenco mostrato dal comando history)
- ctrl-r consente di inserire una stringa e selezionare tutti i comandi precedenti che la contengono. Ogni pressione successiva della combinazione ctrl-r accede agli altri comandi della stessa selezione
- tab auto-completa i nomi di file. Una doppia pressione (rapida) mostra l'elenco di tutte le possibilità

Ridirezione flussi dati

Flussi dati



Filtri Unix

- **cat [opzioni] [file...]**
 - legge da file o stdin, scrive su stdout
- **grep [opzioni] testo [file...]**
 - Legge da file o stdin, scrive su stdout le linee che contengono <testo>
- **head [opzioni] [file...]**
 - Legge da file o stdin, scrive su stdout un subset delle righe (prime n)
- **tail [opzioni] [file...]**
 - Legge da file o stdin, scrive su stdout un subset delle righe (ultime n)
- **cut [opzioni] [file...]**
 - Legge da file o stdin, scrive su stdout un subset delle colonne del file
- **sort [opzioni] [file...]**
 - Legge da file o stdin, scrive su stdout linee ordinate
- **tee [opzioni] file**
 - Legge da stdin, sdoppia il flusso in ingresso su stdout e <file>

Ridirezione

- E' possibile ridirigere input e/o output di un comando facendo sì che stdin/stdout/stderr siano sostituiti da file **in modo trasparente al comando**
- Ridirezione dell'input
 - comando < filein
- Ridirezione dell'output
 - comando > fileout (sovrascrive fileout)
 - comando >> fileout (aggiunge alla fine di fileout)

Ridirezione

```
$ cat /etc/passwd
```

cat legge da /etc/passwd e stampa il contenuto su stdout

```
$ cat < /etc/passwd
```

cat legge da stdin, ma il flusso proviene da /etc/passwd

```
$ sort < f > f2
```

sort legge da stdin, ma il flusso proviene da f

sort scrive su stdout, ma il flusso è ridiretto su f2

```
$ head f2 > f3
```

head legge da f2

head scrive su stdout, ma il flusso è ridiretto su f3

Separazione stdout/stderr

```
$ grep nicola /etc/passwd
```

- Seleziona tutte le righe che contengono la stringa *nicola* all'interno del file */etc/passwd* e le stampa sul terminale
 - Se il file viene trovato, stampa il risultato su stdout
 - Se il file non viene trovato, stampa un errore su stderr
- E' possibile separare i due flussi menzionandoli in modo esplicito con i loro valori numerici (0 = stdin, 1=stdout, 2=stderr)

```
$ grep nicola /etc/passwd 1>/dev/null (scarta stdout, mostra solo stderr)
```

```
$ grep nicola /etc/passwd 2>/dev/null (scarta stderr, mostra solo stdout)
```

Separazione stdout/stderr

- E' possibile ridirigere un flusso all'interno di un altro flusso

```
$ grep nicola /etc/passwd 1>/dev/null 2>&1
```

```
$ grep nicola /etc/passwd >/dev/null 2>&1
```

(Stesso significato, scrittura meno chiara)

- Il flusso 2 viene ridiretto all'interno del flusso 1. Il carattere & chiarisce che non si tratta di un file di nome 1, ma del flusso 1 (stdout).

Composizione comandi

Combinare comandi

- E' sempre possibile combinare comandi utilizzando il filesystem come strumento di mediazione

```
$ sort f > f2; head -n 10 f2
```

- sort legge f, lo ordina, stampa su stdout (ridiretto su file f2). head legge le prime 10 linee di f2. Il carattere ; è utilizzato per combinare comandi sulla stessa linea.

- Approccio estremamente inefficiente. La memoria secondaria (HDD) è molto meno performante della memoria primaria (RAM). HDD SSD: ~0.1GB/S, DDR4: ~5GB/S

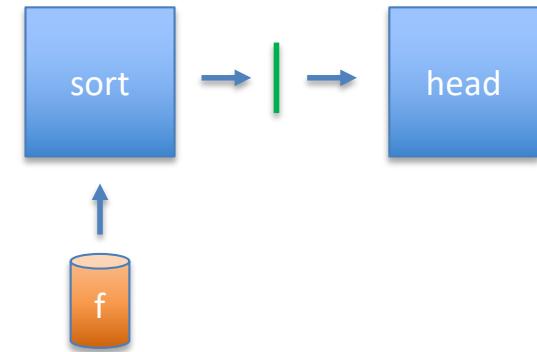
Combinare comandi

- L'output di un comando può esser diretto a diventare l'input di un altro comando (usando costrutto pipe '|')
- Pipe come costrutto parallelo (l'output del primo comando viene reso disponibile al secondo e consumato appena possibile, in assenza di file temporanei)

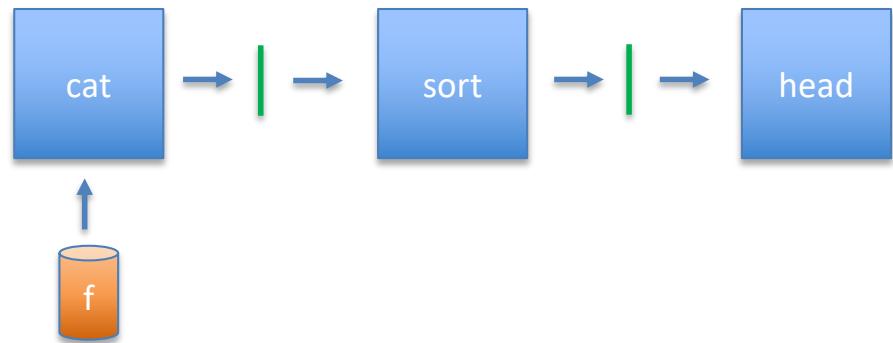
```
$ sort f | head
```

Combinare comandi

```
$ sort f | head
```



```
$ cat f | sort | head
```



Esempi

```
$ who | wc -l
```

Conta gli utenti collegati al sistema

```
$ ls -l | grep ^d | cut -d ' ' -f1 | sort
```

Stampa il contenuto della cartella corrente, seleziona le righe che iniziano per d, seleziona solo la prima colonna, ed ordina il risultato

Variabili

Variabili shell

- E' possibile definire variabili (trattate come stringhe) ed assegnare loro un valore con operatore =

```
$ VAR=3
```

- Si accede ai valori delle variabili con il carattere speciale \$

```
$ echo $VAR
```

```
$ A=1; B=nicola
```

```
$ echo $A
```

```
1
```

```
$ echo $B
```

```
Nicola
```



Visibilità variabili

- La visibilità delle variabili definite all'interno di una shell è limitata alla shell stessa. **Eventuali sotto-shell non ereditano le variabili.**

```
$ A=1  
$ echo $A  
1  
$ bash (sotto-shell)  
$ echo $A
```

```
$
```

Variabili d'ambiente

- Per propagare variabili anche alle sotto shell si utilizzano particolari variabili chiamate d'ambiente.
- Ogni comando esegue nell'ambiente associato alla shell che lo esegue. Ogni shell eredita l'ambiente dalla shell che l'ha creata
- La prima shell Bash ad eseguire dopo il login o dopo l'apertura di un terminale grafico, legge un file (.profile/.bashrc) che contiene fra le altre cose variabili di configurazione che vengono così propigate a tutte le shell eseguite in futuro.

Variabili d'ambiente

```
$ env
SHELL=/bin/bash
TERM=xterm-256color
USER=nicola
HOME=/home/nicola
LOGNAME=nicola
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

Variabili d'ambiente

- Il comando `builtin set` è un'alternativa al comando `env`.
- Differenze principali:
 - `set` è un comando builtin: vede variabili sia locali che d'ambiente della shell corrente
 - `env` è un comando esterno: vede solo variabili, per ovvi motivi, solo variabili d'ambiente

Variabili d'ambiente

- E' possibile aggiungere variabili all'ambiente utilizzando il comando **export**. Le variabili esportate si comportano come variabili locali ma sono visibili anche dalle sotto-shell

```
$ export A=1
$ echo $A
1
$ bash (sotto-shell)
$ echo $A
1
$
```



Variabile PATH

- I binari di sistema sono posizionati all'interno di varie directory. Le principali sono /bin, /usr/bin ma possono esisterne altre anche definite dall'utente.
- La variabile d'ambiente PATH tiene traccia dell'elenco delle cartelle che contengono binari all'interno del sistema

```
$ echo $PATH
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b  
in:/sbin:/bin:/usr/games:/usr/local/games:/snap  
/bin
```

Variabile PATH

- E' possibile aggiungere un nuovo percorso alla variabile PATH utilizzando l'apposita variabile

```
$ export PATH="$PATH":/opt/mybin (alla fine)  
$ export PATH= /opt/mybin:$PATH (all'inizio)
```

```
$ export PATH="" (svuota PATH)  
$ ls  
ls: No such file or directory
```

Espansione ed inibizione



Metacaratteri

- La shell riconosce caratteri speciali (wild card)
 - * una qualunque stringa di zero o più caratteri in un nome di file
 - ? un qualunque carattere in un nome di file
 - [abc] un qualunque carattere, in un nome di file, compreso tra quelli nell'insieme. Anche range di valori: [a-g]. Per esempio ls [q-s]* stampa tutti i file con nomi che iniziano con un carattere compreso tra q e s
 - \ segnala di non interpretare il carattere successivo come speciale

Metacaratteri

```
$ ls [a-p,1-7]*[c,f,d]?
```

Elenca i file i cui nomi hanno come iniziale un carattere compreso tra ‘a’ e ‘p’ oppure tra ‘1’ e ‘7’, e il cui penultimo carattere sia ‘c’, ‘f’, o ‘d’

```
$ ls *\**
```

Elenca i file che contengono, in qualunque posizione, il carattere ‘*’

Esecuzione in-line

E' possibile eseguire un comando ed utilizzarne l'output all'interno di un altro comando

```
$ echo $(pwd)  
/home/nicola  
$ echo `pwd`  
/home/nicola  
$ echo $(expr 2 + 3)  
5
```

Espansione

- Comandi contenuti tra `$()` o `` (backquote) sono eseguiti e sostituiti dal risultato prodotto
- Nomi delle variabili (`$A`) sono espansi nei valori corrispondenti
- Metacaratteri `* ? []` sono espansi nei nomi di file secondo un meccanismo di pattern matching

Inibizione espansione

- In alcuni casi è necessario privare i caratteri speciali del loro significato, considerandoli come caratteri normali
 - \ carattere successivo è considerato come un normale carattere
 - '' (singoli apici): proteggono da qualsiasi tipo di espansione
 - ""(doppi apici) proteggono dalle espansioni con l'eccezione di \$ \ ` (backquote)

Inibizione espansione

```
$ rm *$var *
```

Rimuove i file che cominciano con *\$var

```
$ rm *$var*
```

Rimuove i file che cominciano con *<contenuto della variabile var>

```
$ echo $(pwd)
```

/home/nicola

```
$ echo <$(pwd)>
```

```
<$(pwd)>
```

```
$ A=1+2; B=$(expr 1 + 2)
```

In A viene memorizzata la stringa 1+2, in B la stringa 3

Riassumendo

```
$ cp -r $(pwd)/ssh* "$HOME"/config >  
/tmp/service.log
```

- Ridirezione dell'input/output
 - Esecuzione e sostituzione dei comandi \$()
- $\$(pwd)$ → /etc
- Sostituzione di variabili e parametri
- $\$HOME$ → /home/nicola
- Sostituzione di metacaratteri $plu?o^*$ plutone
- ss^* → /ssh/