

# Shell di UNIX

---

---

## Sommario

---

---

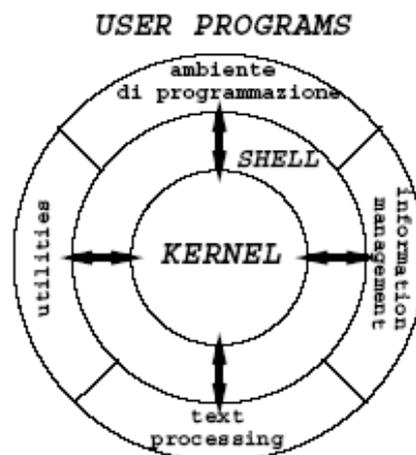
- Introduzione
- I comandi di base
- Il file system
- I processi
- La programmazione della shell

# Caratteristiche UNIX

- Caratteristiche principali
  - Multitasking & multiutente
  - Ottima integrazione in rete
  - Interfaccia utente modificabile
  - Modularità
  - File system gerarchico
  - Vari strumenti di ausilio alla programmazione

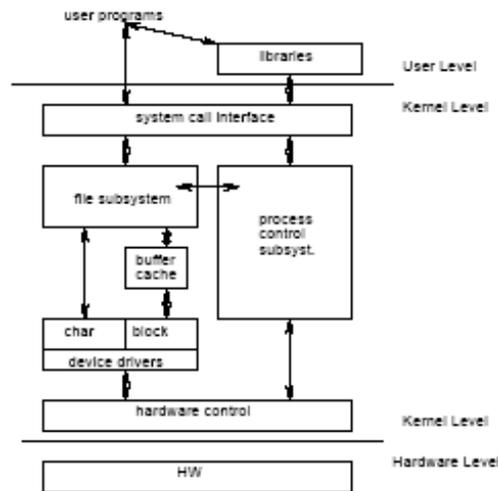
3

# La struttura



4

# La struttura



5

# Evoluzione

anno	UCB	Bell Labs	AT&T	MS	note
1969		PDP-7			K.Thompson
1971		PDP-11/20			B-language
1973		Ritchie			C-language
1976		V6			licenza
1977			PWB		AT&T internal
1978		V7			portabile
1979		32V			VAX
1979	BSD3				B.Joy
1980	BSD4.1				
1982			System III		in vendita
1983	BSD4.2	V8	System V		Ethernet
1984			SVR2	Xenix	PC
1986	BSD4.3	V9	SVR3		RFS, streams
1988	Tahoe		SVR3.2		80386
1989			SVR4		
1991					USL
1992					UNIVEL

BSD Berkeley Software Distribution

SVR System V Release

USL UNIX System Labs

6

## I dialetti

- **UNIX** è il nome di una famiglia di sistemi operativi, con diverse implementazioni per le varie architetture HW

<i>Nome</i>	<i>Produttore</i>
AIX	IBM
A/UX	Apple
BSD	Univ. California Berkeley
HP-UX	Hewlett-Packard
Linux	public-domain
OSF/1	DEC
SCO Unix	Santa Cruz Operation
SunOS	SUN
Solaris	SUN
Ultrix	DEC
System V	vari

7

## Standardizzazione

- Dalla fine degli anni 80 ci sono stati numerosi sforzi per “standardizzare” UNIX
- L’obiettivo è la portabilità delle applicazioni a livello sorgente
  - Programmi C
  - Script di shell
  - Programmi in altri linguaggi
- La competizione dei vari costruttori per il controllo dello Unix “Standard” ha creato una situazione piuttosto complessa

8

## Standardizzazione

---

- Standard principali
  - POSIX (IEEE dal 1988, poi ISO) “Portable Operating System Interface for Unix”
  - XGP (X/Open, dal 1989) “X/Open Portability Guide”
  - SVID (AT&T, 1989) “System V Interface Definition”
  - OSF (Open Software Foundation)

9

## La filosofia UNIX

---

- UNIX è più che una famiglia di sistemi operativi
  - Un insieme di programmi
  - Una filosofia basata su di essi
- Scopo di questa parte del corso è fornire una introduzione a questa filosofia
- Per una dettagliata descrizione dei comandi si rimanda ai manuali

10

## I Comandi di base

---

---

## Una sessione di lavoro

---

---

- Inizio di una sessione
  - **Login:**
  - **Passowd:**
- Fine di una sessione
  - CTRL-d, exit, logout (dipende dall'interprete dei comandi)
- **NOTA:** i caratteri maiuscoli sono diversi dai minuscoli!

## I comandi in UNIX

- Sintassi, in generale, di un comando UNIX  
`Comando [-opzioni] argomenti`
- I comandi troppo lunghi possono essere continuati sulla riga successiva battendo “\” come ultimo carattere della riga
- Si possono dare più comandi sulla stessa riga separandoli con “;” (saranno eseguiti in sequenza)  
`comando1 ; comando2 ; ...`

13

## Informazioni sul sistema

- Ogni utente è identificato dal suo login (UID) ed appartiene a uno o più gruppi (GID)
- Per avere informazioni sugli utenti o sul sistema:
  - `whoami`
  - `who am i`
  - `who`
  - `w`
  - `id`
  - `groups`
  - `finger`
  - `uname`
  - `passwd`
  - `su`
  - `date`

14

## Help in linea

---

- Tutti i comandi di UNIX sono documentati in linea
  - **man comando**

Organizzante in sezioni corrispondenti ad argomenti

1. Commands
2. System Calls
3. Library Functions
4. Administrative Files
5. Miscellaneous Information
6. Games
7. I/O and Special Files
8. Maintenance Commands

15

## Help in linea

---

- Oltre al man
  - apropos chiave**  
elenca le pagine del manuale contenente *chiave*
  - whatis comando**  
indica le sezioni del manuale in cui si trova *comando*

16

## Il File System

---

---

### I path

---

---

- . è la directory corrente
- .. è la directory padre di quella corrente
- I file che iniziano con . sono nascosti
- Path assoluto = /dir1/dir2/...
  - Parte dalla radice del file system
- Path relativo = dir1/dir2/...
  - Parte dalla cartella corrente

# I file

---

- Un solo tipo di file fisico: byte stream
- 4 tipi di file logici
  - Directory
    - Contiene nomi e indirizzi di altri file
  - Special file
    - Entry point per un dispositivo di I/O
  - Link
    - Collegamento ad un altro file
  - File ordinario
    - Tutti gli altri file

19

# Special file

---

- Ogni device di I/O visto come un file
- I programmi non sanno se operano su file o device di I/O
- Lettura/scrittura su special file causano operazioni di I/O sul relativo device
- Indipendenza dai dispositivi!

20

# Link

- Hard link
  - Un nome (in una directory) che punta a un i-node puntato anche da altri
- Soft link
  - Un file che contiene il nome di un altro file
- Particolarità
  - Non si può fare hard link di directory
  - Non si può fare hard link a file su altri file system
  - Un file viene rimosso quando tutti i suoi hard link sono stati rimossi

21

# Il comando `ls`

- Per visualizzare il contenuto di una directory  
**`ls [-opzioni] file ...`**  
Opzioni
  - a visualizza anche i file che iniziano con il punto
  - l output in formato esteso
  - g include/sopprime l'indicazione del proprietario
  - r ordine inverso (alfabetico o temporale)
  - F appende carattere per indicare i file particolari (/ \* @)
  - R elenca anche i file nelle sottodirectory

22

## Occupazione spazio su disco

---

- Per controllare l'occupazione dei dischi  
`df [-k -h]`  
Opzioni
  - k mostra l'occupazione in KByte
  - h mostra l'occupazione in formato "umano"

23

## Occupazione spazio su disco

---

- Per vedere lo spazio (in blocchi) occupato da una directory e tutte le sottodirectory  
`du [-opzioni] directory ...`  
Opzioni
  - a mostra l'occupazione di ciascun file
  - s mostra solo il totale complessivo

24

## Visualizzazione di file di testo

**cat file1 file2 ...**

concatena i file sullo std output

**head [-n] file1 file2**

visualizza le prima *n* righe

**tail [-+nrf] file1 file2 ...**

Visualizza le ultime (con + salta le prime) 10 righe

-r visualizza in ordine inverso

-f rilegge continuamente il file

-n visualizza (salta) le ultime (prime) *n* righe

25

## Visualizzazione per pagine

- Esistono tre comandi quasi equivalenti

**pg file1 file2 ...**

**more file1 file2 ...**

**less file1 file2 ...**

- Durante la visualizzazione è possibile dare dei comandi interattivi

- spazio prossima pagina

- CR prossima riga

- b pagina precedente

- */pattern* prossima pagina con *pattern*

- *?pattern* pagina precedente con *pattern*

- q termina programma

- v edita file corrente

26

## Manipolazione di file

---

**cp [-fir] src1 src2 ... dest**

copia uno o più file

**rm [-fir] file1 file2 ...**

cancella i file elencati

**mv [-fi] file1 file2 ... dest**

sposta uno o più file/cambia il nome di un file

-f non chiede mai conferma (attenzione!!!)

-i chiede conferma per ciascun file

-r opera ricorsivamente nelle sottodirectory

27

## Manipolazione di directory

---

**cd *directory***

cambia la directory in quella indicata

**pwd**

mostra path directory corrente

**mkdir *directory***

crea la directory specificata

**rmdir *dir1 dir2 ...***

cancella una o più directory (devono essere vuote)

28

## Cambio di proprietario

**chgrp [-R] gruppo file**

cambia il gruppo del file

**chown [-R] utente[:gruppo] file**

cambia proprietario [e gruppo] del file

- In entrambi i casi l'opzione **-R** indica di propagare il comando alle sottodirectory

29

## Cambio protezione

**chmod [-R] protezione file**

Protezioni assolute: un numero di quattro cifre (il primo si può omettere)

	padrone	gruppo	altri
4 2 1	4 2 1	4 2 1	4 2 1
s S t	r w x	r w x	r w x

Protezioni simboliche: una stringa di tre caratteri

**u g o a + - = r w x s t**

30

## Cambio protezione

- Esempi

- **chmod 640 prova.txt**

- Lettura/scrittura per proprietario
    - Lettura per gruppo
    - Nessun permesso per altri

- **chmod 755 dir**

- Lettura/scrittura/esecuzione per proprietario
    - Lettura/esecuzione per gruppo
    - Lettura/esecuzione per altri

31

## Sticky bit

- Sticky bit (t)

- Non usato su file
  - Per directory, solo il proprietario del file o root possono cancellare o rinominare i file contenuti  
(es. directory /tmp)

```
$ ls -ld  
/tmp drwxrwxrwt 6 root root 1024 Aug 10 01:03 /tmp
```

32

## Setuid e setgid

- Setuid (s)
  - Per diventare temporaneamente il padrone del file
- Setgid (S)
  - Per diventare temporaneamente dello stesso gruppo del padrone del file

```
$ ls -l /usr/bin/passwd  
-r-s--x--x 1 root root 17700 Jun 25 2004 /usr/bin/passwd
```

33

## Protezioni standard

### **umask** *maschera*

Per definire la maschera delle protezioni

- Il comando **umask** senza argomento mostra i permessi che sono NEGATI quando si crea un file (la maschera delle protezioni)
- Esempio:  
*umask 027*  
Nega tutti i permessi agli "altri" e i permessi di scrittura al "gruppo"

34

## Ricerca di un file

### **find directory espressione**

Visita tutto l'albero sotto la directory specificata e ritorna i file che rendono vera l'espressione

- name pattern (usare gli apici se si usano espressioni regolari)
- type tipo (b c d l f)
- user utente
- group gruppo
- newer file
- atime, mtime, ctime [+/-] giorni
- print
- size [+/-] blocchi

35

## Confronto di file

```
diff [-opzioni] file1 file2
```

```
diff [-opzioni] dir1 dir2
```

mostra le righe diverse, indicando quelle da aggiungere (a), cancellare (d) e cambiare (c)

- b ignora gli spazi a fine riga, collassa gli altri
- i ignora la differenza tra maiuscolo e minuscolo
- w ignora completamente la spaziatura

36

## Confronto di file – Esempio

- Prova1  
ciao  
come va?  
bene  
grazie
- Prova 2  
ciao  
come?  
bene  
molto bene  
grazie
- Prova 3  
ciao

```
$ diff Prova1 Prova2
2c2
< come va?
---
> come?
4c4,5
< grazie
---
> molto bene
> grazie
```

```
$ diff Prova1 Prova3
2,4d1
< come va?
< bene
< grazie
```

```
$ diff Prova3 Prova1
1a2,4
> come va?
> bene
> grazie
```

37

## Modifica di attributi di file

**touch [-opzioni] [data] file ...**

aggiorna data e ora dell'ultimo  
accesso/modifica di un file

- se data non è specificata, usa data e ora corrente
- se il file non esiste lo crea vuoto

-a modifica accesso

-m ultima modifica

38

# Processi

---

---

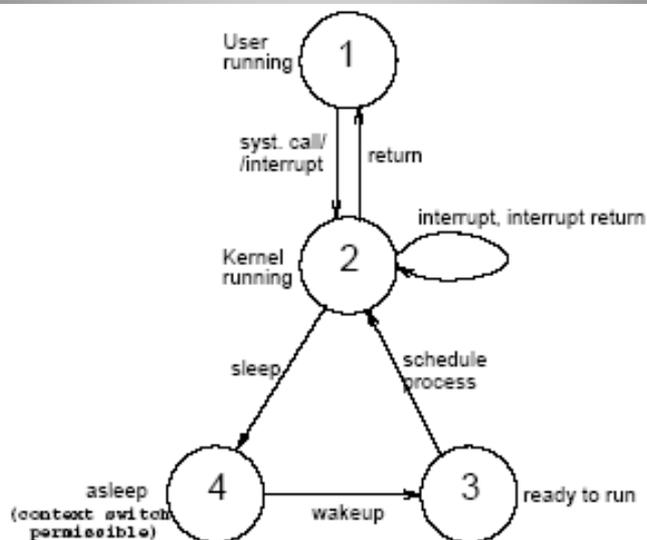
## I processi

---

---

- Un processo è un programma in esecuzione
- Un processo è una sequenza di byte che la CPU interpreta come istruzioni (text e dati)
- Caratteristiche
  - Organizzazione gerarchica
  - PID (assegnato dal sistema)
  - Priorità (assegnata dal sistema)
- Evolve attraverso un certo numero di stati

## Lo stato dei processi



41

## Lo stato dei processi

- Il comando `ps` permette di analizzare lo stato di un processo
- Numerose opzioni con vari livelli di informazioni
- Output base

PID	TTY	TIME	CMD
3490	pts/3	00:00:00	bash
3497	pts/3	00:00:00	ps

PID Process Identifier

TTY terminale da cui il processo è eseguito

TIME tempo totale di esecuzione

CMD comando eseguito corrispondente

42

## Lo stato dei processi

- Opzioni principali
  - e: visualizza tutti i processi (tutti gli utenti)
  - x: visualizza anche i processi in background
  - u: visualizza info sulle risorse
- Stati di un processo:

<b>R</b>	In esecuzione/esequibile
<b>T</b>	Stoppato (es. ^Z)
<b>S</b>	Addormentato
<b>Z</b>	Zombie
<b>D</b>	In attesa I/O non interrompibile

43

## Zombie e daemon

- Zombie: processo che ha terminato o è stato ucciso, ma non riesce a segnalare l'evento al padre
- Daemon (demone): processi che girano persistentemente in background e forniscono servizi al sistema (es: la posta elettronica o la gestione delle risorse)
  - Sono disponibili in qualunque momento per servire più task o utenti

44

## Gestione dei processi

---

- I processi normalmente eseguono in foreground e hanno tre canali standard connessi al terminale
- I processi attivati con & eseguono in background e sono provi di stdin
- Un processo in foreground può essere sospeso con ^Z

45

## Gestione dei processi

---

- I processi sospesi possono essere continuati sia in foreground che in background
- I processi in background possono essere riportati in foreground
- Il comando `at` permette di lanciare e controllare processi batch

46

## Gestione dei processi – Comandi

---

- **jobs [-l]**  
elena i job in background o sospesi
- **bg [job-id]**  
esegue i job specificati in background
- **fg [job-id]**  
esegue i job indicati in foreground
- **kill [-signal] process-id**
- **kill [-signal] %job-id**  
manda un segnale al processo/job indicato (i più comuni sono  
1 HUP e 9 KILL)
- **kill -l**  
elena tutti i segnali disponibili

47

La programmazione  
della shell

---

# Shell

- E' lo strato più esterno del sistema operativo
- Offre due vie di comunicazione con il SO
  - interattivo
  - shell script
- Script di shell
  - è un file (di testo) costituito da una sequenza di comandi
- La shell non è parte del kernel del SO, ma è un normale processo utente
  - Ciò permette di poter modificare agevolmente l'interfaccia verso il sistema operativo

49

# Shell – Caratteristiche

- Espansione/completamento dei nomi dei file
- Ri-direzione dell'I/O (stdin, stdout, stderr)
- Pipeline dei comandi
- Editing e history dei comandi
- Aliasing
- Gestione dei processi (foreground, background sospensione e continuazione)
- Linguaggio di comandi
- Sostituzione delle variabili di shell

50

## Le shell disponibili

---

- Bourne shell (sh)
  - La shell originaria, preferita nella programmazione sistemistica
- C-shell (csh)
  - La shell di Berkeley, ottima per l'uso interattivo e per gli script non di sistema
- Korn shell (ksh)
  - La Bourne sh riscritta dall'AT&T per assomigliare alla C-shell
- Tahoe (tcsh)
  - Dal progetto Tahoe, una C-shell migliorata

51

## Le shell disponibili

---

- All'interno del corso useremo la *bash*
  - Bourne again shell (bash)
    - Tipica shell di Linux

<http://www.linuxdoc.org/HOWTO/Bash-Prompt-HOWTO/index.html>

<http://www.linuxdoc.org/HOWTO/Bash-Prog-Intro-HOWTO.html>

man bash

52

## Le shell a confronto

Shell	Chi	Complessità relativa (in linee di codice)
sh	S.R. Bourne	1.00
csh	UCB	1.73
bash	GNU, LINUX	2.87
ksh	David Korn (AT&T)	3.19
tcsh	Tahoe	4.54

53

## Esecuzione della shell

- `/etc/passwd` contiene info relative al login
  - tra cui quale programma viene automaticamente eseguito al login (in genere sempre una shell)
- Durante l'esecuzione, la shell cerca nella directory corrente, nell'ordine, i seguenti file di configurazione
  - `.bash_profile`
  - `.bash_login`
  - `.profile`
    - contengono i comandi che vengono eseguiti al login

54

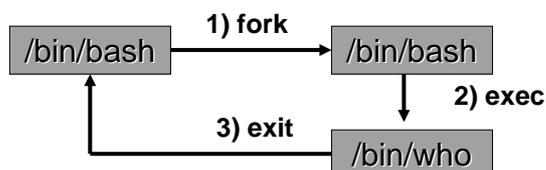
## Esecuzione della shell

- Se la shell non è di tipo “login” viene eseguito il file `.bashrc`
- Se non li trova, vengono usati quelli di sistema nella directory `/etc`
- E' previsto anche un file `.bash_logout` che viene eseguito alla sconnessione

55

## Funzionamento della shell

- Esempio: esecuzione del comando `who`



- System call coinvolte
  - `fork()`  
crea un nuovo processo (figlio) che esegue il medesimo codice del padre
  - `exec()`  
carica un nuovo codice nell'ambito del processo corrente
  - `exit()`  
termina il processo corrente

56

## Bash – Variabili

---

- La shell mantiene un insieme di variabili per la personalizzazione dell'ambiente
- Assegnazione: **variabile=valore**
- Variabili di shell più importanti
  - PWD la directory corrente
  - PATH elenco di directory in cui cercare comandi
  - HOME directory di login
  - PS1, PS4 stringhe di prompt
    - (vedere PROMPTING su `man bash`)
- Le assegnazioni vengono in genere aggiunte all'interno del **.bash\_profile**

57

## Bash – Variabili

---

- Per accedere al valore di una variabile, si usa l'operatore **\$**
  - Esempio: se **x** vale 123, si può usarne il valore tramite **\$x**
- Per visualizzare il valore di una variabile, si usa il comando **echo**
- NOTA
  - I valori delle variabili sono sempre STRINGHE
  - Per valutazioni aritmetiche si può usare l'operatore **\$( )**, oppure il comando **let**

58

## Bash – Variabili

- Esempio

```
# x=0
# echo $x+1
0+1
# echo $((x+1))
1
# let "x+=1"
# echo $x
1
```

59

## Bash – Storia dei comandi

- La bash mantiene una storia dei “comandi precedenti” dentro un buffer circolare memorizzato nel file indicato dalle variabili **HISTFILE** (default **.bash\_history**)
- Utile per chiamare comandi o correggerli

60

## Bash – Storia dei comandi

- Per accedere ai comandi
  - !n esegue il comando n del buffer (potrebbe non esserci)
  - !! esegue l'ultimo comando
  - !-n esegue l'n-ultimo comando
  - !\$ l'ultimo parametro del comando precedente
  - !^ il primo parametro del comando precedente
  - !\* tutti i parametri del comando precedente
  - !stringa l'ultimo comando che inizia con stringa
  - ^stringa1^stringa2 sostituisce stringa1 nell'ultimo comando con stringa 2

61

## Bash – Storia dei comandi

- Esempio

```
#cc -g prog.c
#vi iop.c
#cc prog.c iop.c
#a.out
```
- Dopo l'ultimo comando si ha

```
#!$ esegue a.out
#!-1 idem
#!c esegue cc prog.c iop.c
#!v esegue vi iop.c
#rm !* esegue rm a.out
#rm !$ esegue rm a.out
```

62

## Bash – Globbing

- Espansione dei nomi dei file (e comandi) con il tasto TAB (o ESC)
  - Per i nomi di file eseguibili la shell cerca nelle directory del PATH
  - Per i file generici, la shell espande i nomi di file nella directory corrente

63

## Bash – Wildcard

- Caratteri speciali
  - / separa i nomi delle directory in un path
  - ? un qualunque carattere (ma solo uno)
  - \* una qualunque sequenza di caratteri
  - ~ la directory di login
  - ~**user** la directory di login dello **user** specificato
  - [ ] un carattere tra quelli in parentesi
  - { , } una parola tra quelle in parentesi (separate da , )
- Esempio

```
cp ~/.[azX]* ~/rap{1,2,20}.doc ~/man.wk? ~bos
```

64

## Bash – Aliasing

- E' possibile definire dei comandi con nuovi nomi (alias), tipicamente più semplici

**alias**

elenca gli alias definiti

**alias nome='valore'**

definisce un alias (no spazi prima/dopo = )

**unalias nome**

cancella un alias

- Esempio

- **alias ll="ls -la"**

65

## Bash – Ambiente

- Le variabili sono di norma locali alla shell
  - Occorre un meccanismo che consenta di passare i valori delle variabili ai processi creati dalla shell (in particolare alle sub-shell)
- L'ambiente della shell è una lista di coppie **nome=valore** trasmessa ad ogni processo creato

- **export variabile [ = valore ]**

assegna un valore a una variabile di ambiente

Esempio:

```
export PS1='\h_mionome_\w>'
```

- **printenv [variabile]**

stampa il valore di una o tutte le variabili d'ambiente

- **env**

stampa il valore di tutte le variabili d'ambiente

66

## Bash – Variabili di Ambiente

- Le principali variabili d'ambiente

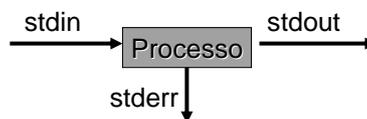
PWD	SHELL
PATH	HOME
HOST	HOSTTYPE
USER	GROUP
MAIL	MAILPATH
OSTYPE	MACHTYPE

- Alcune variabili di ambiente sono legate al valore delle corrispondenti variabili di shell (per es. PATH)

67

## Ri-direzione dell'I/O

- Ogni processo ha tre canali associati



- Ogni canale può essere ri-diretto
  - su file
  - su altro canale
- Il collegamento stdout → stdin si chiama **pipe** e crea in memoria un canale di comunicazione diretto tra due processi

68

## Ri-direzione dell'I/O

---

**comando < file**  
    stdin da file

**comando > file**  
    stdout in file (cancellato se esiste)

**comando >> file**  
    stdout aggiunto in coda al file

**comando >& file**  
    stderr e stdout in file

**comando 2> file**  
    stderr in file (cancellato se esiste)

69

## Ri-direzione dell'I/O

---

**comando > file 2>&1**

**comando &> file**  
    stdout e stderr sullo stesso file  
    descriptor

**comando1 | comando2**  
    pipe tra comando1 e comando2 (stdout  
    di comando1 in stdin di comando2)

70

## File di comandi (script)

---

- E' possibile memorizzare in un file una serie di comandi, eseguibili richiamando il file stesso
- Esecuzione
  - Eseguendo `bash script argomenti` sulla linea di comando
  - Eseguendo direttamente `script`
    - E' necessario che il file abbia il permesso di esecuzione, ossia, dopo averlo creato si esegue: `chmod +rx file`
    - Per convenzione, la prima riga del file inizia con `#!`, seguita dal nome della shell entro cui eseguire i comandi (`#!/bin/bash`)

71

## Esempio

---

```
#!/bin/bash
date    #restituisce la data
who     #restituisce chi è connesso
```

72

## Variabili speciali

---

- La bash memorizza gli argomenti della linea di comando dentro una serie di variabili **\$1, ... \$n**
- Alcune variabili speciali
  - \$\$ PID del processo shell
  - \$0 Il programma corrispondente al processo corrente
  - \$# il numero di argomenti
  - \$? se esistono argomenti (no=0, si=1)
  - \*, @\$ tutti gli argomenti

73

## Variabili vettore

---

- Definizione
  - enumerando i valori tra parentesi tonde
- Accesso ai campi
  - con la notazione del C usando le parentesi quadre
  - La valutazione dell'espressione richiede gli operatori { }
- NOTA: gli indici partono da 0!

74

## Variabili vettore

- Esempio

```
# v=(1 2 3)
# echo $v
1
# echo $v[1]
1[1]
# echo ${v[2]}
3
```

75

## Bash – Input/Output

- Per stampare un valore su standard output  
`echo espressione`
- Nel caso si tratti di variabili, per stampare il valore, usare `$`
- Esempio

```
# x=1
# echo x
x
# echo $x
1
```

76

## Bash – Input/Output

- Per acquisire un valore da standard input

```
read variabile
```

- Esempio

```
# read x
pippo
# echo $x
pippo
```

77

## Bash – Strutture di controllo

- Strutture condizionali

```
if [ condizione ];
then azioni;
fi
```

```
if [ condizione ];
then azioni;
elif [condizione ];
then azioni;
...
else
azioni;
fi
```

78

## Bash – Strutture di controllo

- Le **parentesi [ ]** che racchiudono la condizione sono in realtà un'abbreviazione del comando `test`, che può essere usato al loro posto

- Esempio

```
if [ a=0 ]; then # =senza spazi!  
    echo $a;  
fi
```

```
if test a=0; then # =senza spazi!  
    echo $a;  
fi
```

79

## Bash – Test e condizioni

- Per specificare condizione in un `if` è necessario conoscere il comando `test`

```
test operando1 operatore operando2
```

80

## Bash – Test e condizioni

### Operatori principali (`man test` per altri)

Operatore	Vero se ...	# di operandi
<code>-n</code>	operando ha lunghezza $\neq$ 0	1
<code>-z</code>	operando ha lunghezza = 0	1
<code>-d</code>	esiste una directory con nome = operando	1
<code>-f</code>	esiste un file regolare con nome = operando	1
<code>-e</code>	esiste un file con nome = operando	1
<code>-r, -w, -x</code>	esiste un file leggibile/scrivibile/eseguibile	1
<code>-eq, -neq</code>	gli operandi sono interi e sono uguali/diversi	2
<code>=, !=</code>	gli operandi sono stringhe e sono uguali/diversi	2
<code>-lt, -gt</code>	operando1 <, > operando2	2
<code>-le, -ge</code>	operando1 $\leq$ , $\geq$ operando2	2

81

## Bash – Strutture di controllo

```
case selettore
case1):      azioni;;
case2):      azioni;;
...
):           azioni;;
esac
```

82

## Bash – Strutture di controllo

- Esempio

```
if [ -e "$HOME/.bash_profile" ]; then
    echo "you have a .bash_profile file";
else
    echo "you have no .bash_profile file";
fi
```

83

## Bash – Strutture di controllo

- Esempio

```
echo "Hit a key, then hit return."
read Keypress
case "$Keypress" in
    [a-z]) echo "Lowercase letter";;
    [A-Z]) echo "Upper letter";;
    [0-9]) echo "Digit";;
    * ) echo "other";;
esac
```

84

## Bash – Strutture di controllo

- Ciclo for

```
for arg in [lista]
do
    comandi
done
```
- `lista` può essere
  - un elenco di valori
  - una variabile (corrispondente ad una lista di valori)
  - un meta-carattere che può espandersi in una lista di valori
- In assenza della clausola `in`, il `for` opera su `$@`, cioè la lista degli argomenti
- E' previsto anche un ciclo `for` che utilizza la stessa sintassi del `for` C/Java

85

## Bash – Strutture di controllo

- Esempi

```
for file in *.c
do
    ls -l "$file"
done
```

```
*****
LIMIT=10
for ((a=1;a <= LIMIT; a++))
# Doppie parentesi e "LIMIT"senza "$"
do
    echo -n "$a "
done
```

86

## Bash – Strutture di controllo

- Ciclo while

```
while [ condizione ]
do
  comandi
done
```
- La parte tra [ ] indica l'utilizzo del comando test (come per `if`)
- E' previsto anche un ciclo `while` che utilizza la stessa sintassi C/Java

87

## Bash – Strutture di controllo

- Esempio

```
LIMIT=10
a=1
while [ $a -le $LIMIT ]
# oppure
while ((a <= LIMIT))
do
  echo -n "$a "
  let a+=1
done
```

88

## Bash – Strutture di controllo

- Ciclo until

```
until [ condizione vera ]
do
    comandi
done
```

- La parte tra [ ] indica l'utilizzo del comando test (come per `if`)

89

## Bash – Strutture di controllo

- Esempio

```
LIMIT=10
a=1
until [ $a -gt $LIMIT ]
do
    echo -n "$a "
    let a+=1 #oppure a=$(( a+1 ))
done
```

90

## Bash – Funzioni

- E' possibile usare sottoprogrammi (funzioni)
- Sintassi della definizione

```
function nome {
  comandi
}
```
- La funzione vede quali parametri \$1, ...\$n, come fosse uno script indipendentemente dal resto
- Valore di ritorno tramite il comando `return valore`
- Codice di uscita tramite il comando `exit(valore)`

91

## Bash – Funzioni

- Esempio

```
function quit {
  exit
}
```

```
function e {
  echo $1
}
```

```
e "Hello World" #"main" dello script
quit
```

92

## Bash – Funzioni

```
function func2 {
  if [ -z "$1" ] ; then
    echo "Parametro 1 ha lunghezza 0";
  else
    echo "Parametro 1 e' $1";
  fi
  return 0
}
func2 "$1"
```

93

## Bash – Uso output di un comando

- E' possibile utilizzare l'output di un comando come "dati" all'interno di un altro comando
- Tramite l'operatore "` `"
- Sintassi
  - `comando` (``` = ALT+96 su tastiera italiana)
  - `$(comando)`
- Esempio
  - Cancellazione di tutti i file con il nome `test.log` contenuti nell'albero delle directory `/home/joe`

```
rm `find /home/joe -name test.log`
```

94

## Bash – Filtri

- Programmi che ricevono dati di ingresso da stdin e generano risultati su stdout
- Molto utili assieme alla ri-direzione dell'I/O
- Alcuni dei filtri più usati sono
  - `more`
  - `sort`
  - `grep, fgrep, egrep`
  - `cut`
  - `head, tail`
  - `uniq`
  - `wc`
  - `awk (sed)`

95

## Bash – grep

- Per cercare se una stringa compare all'interno di un file

```
grep [-opzioni] pattern file
```

### Opzioni

- c conta le righe che contengono il pattern
- i ignora la differenza maiuscolo/minuscolo
- l elenca solo i nomi dei file contenenti il pattern
- n indica il numero d'ordine delle righe
- v considera solo righe che non contengono il pattern

96

## Bash – Espressioni regolari

- I pattern di ricerca in grep possono essere normali stringhe di caratteri o espressioni regolari. In questo caso, alcuni caratteri hanno un significato speciale (a meno che siano preceduti da \)

.	un carattere qualunque
^	inizio riga
\$	fine riga
*	ripetizione (zero o più volte)
+	ripetizione (una o più volte)
[ ]	un carattere tra quelli in parentesi
[ ^ ]	un carattere esclusi quelli in parentesi
\<	inizio parola
\>	fine parola

97

## Bash – Varianti di grep

**fgrep** [option] [string] [file] ...

- I pattern di ricerca sono stringhe
- E' veloce e compatto

**egrep** [option] [string] [file] ...

- I pattern di ricerca sono delle espressioni regolari estese
- E' potente ma lento
- Richiede molta memoria

98

## Bash – Ordinamento di dati

```
sort [-opzioni] file ...
```

Opzioni

-b	ignora gli spazi iniziali
-d	(modo alfabetico) confronta solo lettere, cifre e spazi
-f	ignora la differenza maiuscolo/minuscolo
-n	(modo numerico) confronta le stringhe di cifre in modo numerico
-o file	scrive i dati da ordinare in file
-r	ordinamento inverso
-t carattere	usa carattere come separatore per i campi
-k s1,s2	usa i campi dalla posizione S1 alla S2

99

## Bash – Selezione di Campi

```
cut -clista file
```

```
cut -flista [-dchar] [-s] file
```

- *lista* specifica un intervallo del tipo
  - *a,b* significa 'a' e 'b'
  - *a-b* significa da 'a' a 'b'

100

## Bash – Selezione di Campi

- Opzioni
  - c           seleziona per caratteri
  - f           seleziona per campi  
Il campo è definito dal separatore  
(default carattere TAB)
  - dchar       char è usato come separatore
  - s           considera solo le linee che contengono il separatore
- Esempi
  - cut -c1-12 file  
prende i primi 12 caratteri di ogni riga del file
  - cut -c1, 4 file  
prende il campo 1 e 4 di ogni riga del file
  - cut -f1-4 file  
prende i primi 4 campi di ogni riga del file

101

## Bash – Selezione di Campi

- Altri esempi
  - cut -d: -f1,5 /etc/passwd  
Estrae user e nome completo degli utenti
  
  - ps -x | cut -d" " -f1  
Elenca i PID dei processi nel sistema

102

## Bash – wc

**wc [-c] [-l] [-w] file**

Legge i file nell'ordine e conta il numero di caratteri, linee e parole

Opzioni

- c    conta solo i caratteri
- l    conta solo le righe
- w    conta solo le parole

- Esempio

```
ps -x | tail +2 | wc -l
```

Conta il numero di processi attivi (tail +2 per togliere l'intestazione)

103

## Bash – uniq

**uniq [-u][-c] file**

- Trasferisce l'input sull'output sopprimendo duplicazioni contigue di righe
- Assume che l'input sia ordinato
- Opzioni
  - u    visualizza solo righe non ripetute
  - c    visualizza anche il contatore del numero di righe

104