

# UNIX INTERFACE

Per iniziare a conoscere le caratteristiche dell'interfaccia di UNIX e dei S.O. che ad esso si rifanno (LINUX, Free BSD, AIX, AUX, ULTRIX, ecc.) si raccomanda vivamente di leggere il par. VI (**THE SHELL**) dell'articolo originale che ne illustra le caratteristiche **The UNIX Time-Sharing System**, Communications of the ACM, 17(7), 365-375, 1974.

## La SHELL

Fornisce un'interfaccia tra l'utente e il Kernel.

L'interfaccia è molto semplice: è costituita da un **prompt** a cui l'utente risponde digitando una riga di comando

Esempi di prompt:

**%**

**\$**

Esistono diverse interfacce per i sistemi UNIX-like:

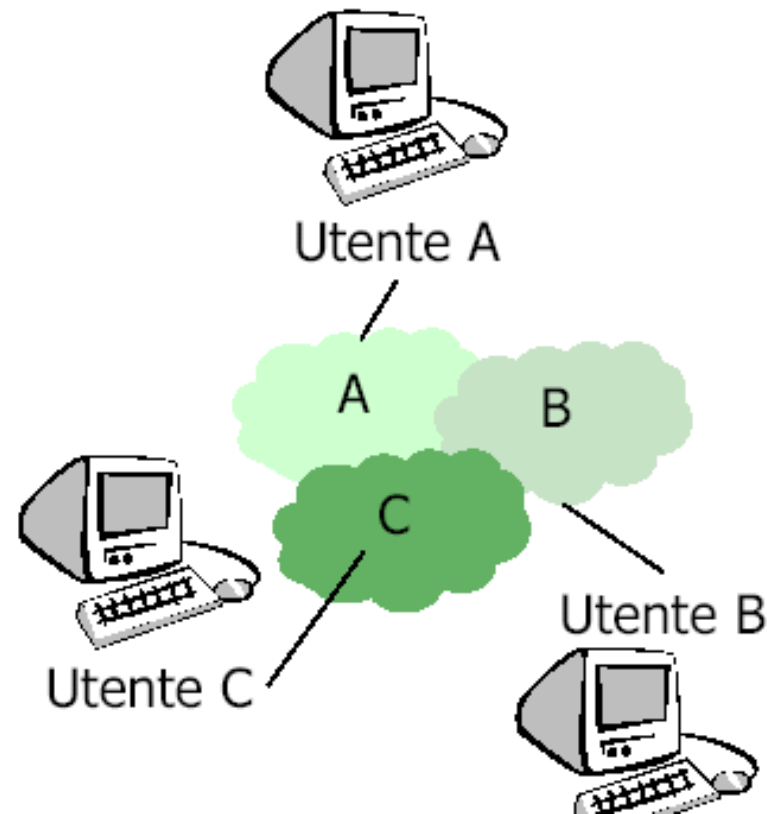
**Bourne** shell, **Korn** shell, **C**-shell, **TC**-shell, **Bash** shell

# UNIX: caratteristiche generali

- Sistema **multi-utente**
  - più utenti possono lavorare contemporaneamente sul sistema
- Sistema **multi-programmato** (Multi-processing)
  - Permette l'esecuzione simultanea di più programmi
  - l'esecuzione di un singolo programma può a sua volta essere scomposta in un insieme di attività concorrenti dette thread (Multi-threading)
- Time-sharing

# Sistema operativo multi-utente

- Ogni utente ha a disposizione un **account**, che comprende un certo numero di risorse:
  - Spazio su disco
  - Possibilità di eseguire certi programmi e/o accedere a file ...
  - ...



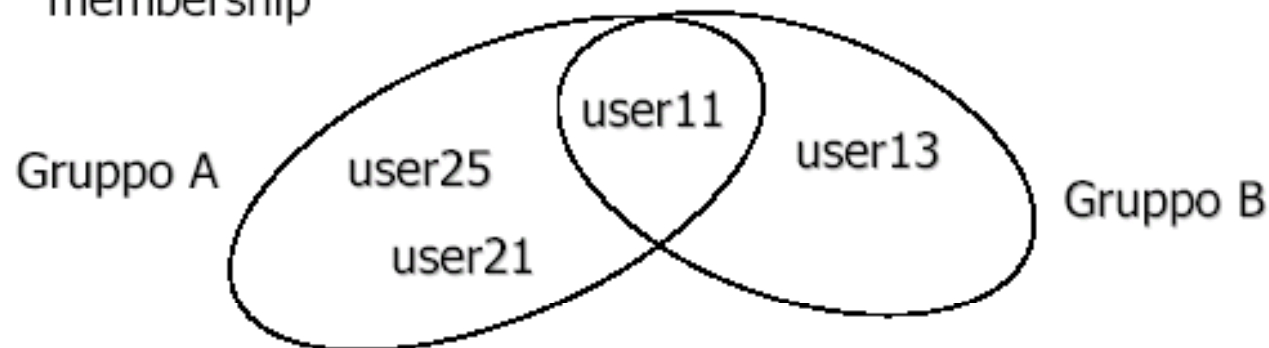
# Identificazione degli utenti

Ogni utente nel sistema è identificato da un **account**, che comprende:

- **User (o Login) Name** : una stringa alfanumerica unica di al più 8 caratteri, assegnata dall'amministratore di sistema
- **User-ID**: un numero intero (non negativo) unico, che corrisponde biunivocamente allo user name
  - User ID e User name sono pubblici
- **Password**: una stringa alfanumerica non necessariamente unica (più utenti potrebbero avere la stessa password)
- **Group Name** di al più 8 caratteri, associato biunivocamente ad un **Group ID** numerico: identifica il gruppo a cui l'utente appartiene

# Gruppi di utenti

- Un gruppo è un insieme di utenti, definito in genere in base al tipo di utenti
- Quando si crea l'account per un utente, questi viene posto almeno in un gruppo
  - Tuttavia un utente può far parte di uno o più gruppi, definiti dall'amministratore del sistema
- L'appartenenza ad un gruppo abilita certi permessi
- Ad ogni istante, solo l'appartenenza ad un gruppo è **attiva**
- Il file pubblico `/etc/group` contiene la lista dei gruppi e delle membership



# Il superuser

- È un utente privilegiato (user name = `root`), al quale sono riservati i compiti di amministrazione del sistema
  - Creazione utenti (`useradd`, `userdel`, `usermod`)
  - Modifica gruppi (`addgrp`, `delgrp`)

**N.B.** Al superuser non è inibito alcun accesso

- I meccanismi di sicurezza che Unix possiede per la protezione da accessi indesiderati sono di tre tipi:
  - **Accesso al sistema:** consentito soltanto agli utenti autorizzati, mediante uno schema di login/logout
  - **Accesso ai files:** consentito soltanto agli utenti autorizzati, mediante uno schema di permessi di accesso
  - **Accesso ai processi:** (ad esempio per terminarli) consentito solo agli utenti autorizzati

# Inizio di una sessione di lavoro: Procedura di Login

- Per iniziare una sessione di lavoro, un utente deve “allocarsi”, ossia accedere al sistema, tramite la **procedura di login**, in cui viene richiesto di notificare la propria identità fornendo user-name e password

**Password:** non viene visualizzata per sicurezza

**User name:** assegnato dall'amministratore del sistema

```
Login: user11
Password: *****
%
```

**Prompt:** la shell è in attesa di comandi



# Procedura di login

- Non appena l'utente digita user name e password, quest'ultima viene crittografata e il sistema effettua una ricerca nel file `/etc/passwd` contenente per ogni utente varie informazioni tra cui:
  - Password crittografata
  - User id
  - Lo shell di default
  - Nome completo dell'utente
- La password fornita è confrontata con quella memorizzata nel file relativamente allo user-id dell'utente
- Il sistema consente l'accesso solo se le due password coincidono
  - In caso contrario, è ripetuta la procedura di login
  - NB: ovviamente l'accesso è negato anche se si sbaglia a digitare lo user name



# Chiusura della sessione di lavoro

## Procedura di logout



- Per disconnettersi dal sistema, si utilizza il comando `logout` (CTRL+d), con il quale il sistema libera il terminale e lo rende disponibile ad altri utenti
  
- L'operazione di logout è importante per due motivi:
  - Evita che persone non abilitate possano utilizzare il terminale e quindi accedere alle informazioni dell'utente che ha aperto la sessione
  - Consente al sistema operativo di salvare correttamente le strutture dati e le variabili di ambiente associate all'utente, senza perdita di informazioni

# LE SHELL PIÙ DIFFUSE

Bourne shell, Korn shell, C-shell

## Bourne shell: sh

La prima, la più diffusa, la più semplice

## C shell: csh

Sviluppata a Berkeley, ha introdotto funzioni per l'uso interattivo

## Korn shell: ksh

La più recente delle tre; combina utili caratteristiche delle prime due. E' un superset di sh

Nota

POSIX.2 ha adottato più o meno sh, con caratteristiche di ksh

L'informazione riguardante la shell correntemente utilizzata è contenuta nella variabile di ambiente SHELL

Il comando `echo $SHELL` visualizza il valore di questa variabile, sotto forma di pathname

### Pathname Shell

/.../sh	Bourne shell
/.../csh	C shell
/.../ksh	Korn shell
/.../tcsh	TC shell
/.../bash	Bourne Again Shell

## La scelta della SHELL

E' possibile lanciare una nuova shell come un normale comando.

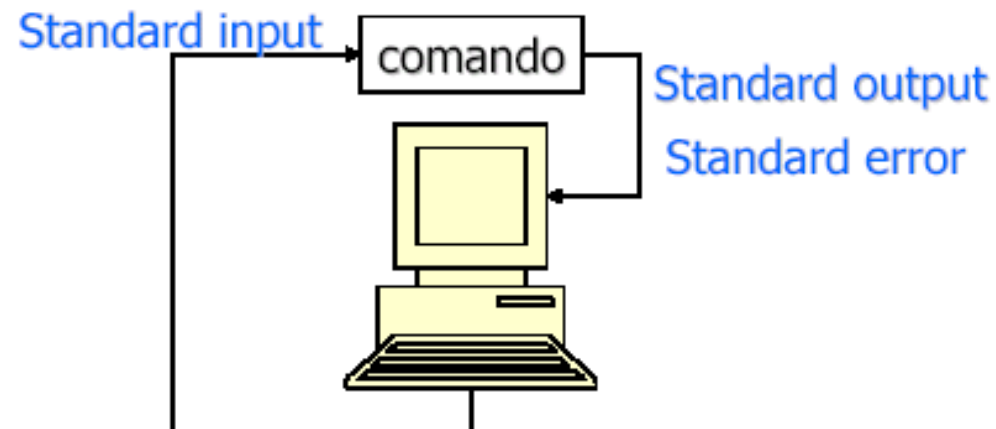
```
login: user11
Password:
Last login: Tue May 23 21:17:30 ...
Sun Microsystems Inc. ...
% sh
$ ksh
$ csh
% ^D $ ^D
$ ^D % ^D logout
```

### Startup files

- Durante le operazioni di inizializzazione, la shell personalizza l'ambiente di uso ...
- ... eseguendo uno o più script di inizializzazione, contenuti in file di pathname prefissato (**startup files**) che si trovano, tipicamente, alcuni nella home directory dell'utente, altri in directory generali e sono definiti dall'amministratore del sistema
- **Nota:** il meccanismo è diverso per ogni shell, e può essere diverso a seconda delle modalità di esecuzione della shell (shell di login oppure no, shell interattiva oppure no, ...)

# Ridirezione dei file standard

- Normalmente, i file standard sono associati al video e alla tastiera



- La shell può variare queste associazioni di default **ridirigendo** i files standard su qualsiasi file nel sistema

- La sintassi di comandi di ridirezione usa
  - < per ridirigere l'input
  - > per ridirigere l'output
- << e >> servono per scrivere in coda ad un file senza perdere il contenuto originale

# Ridirezione dello standard output

`comando argomenti > (>>) file`

- Ridirige lo standard output del comando sul file:
  - se file non esiste, viene creato
  - se file non esiste, viene riscritto (>) oppure il nuovo output viene accodato (>>)

```
% who
roberto pts/1 Jun 10 23:12 (polillo.inet.it)
% who > whofile
% cat whofile
roberto pts/1 Jun 10 23:12 (polillo.inet.it)
```



# Ridirezione dello standard input

```
comando argomenti < (<<) file
```

- Ridirige lo standard input del comando sul file, cioè fa sì che l'input di un comando provenga, anziché dalla tastiera, da un file

```
% mail user12 < memo
```

Ridirige lo standard input al comando `mail`,  
in modo che provenga dal file `memo`

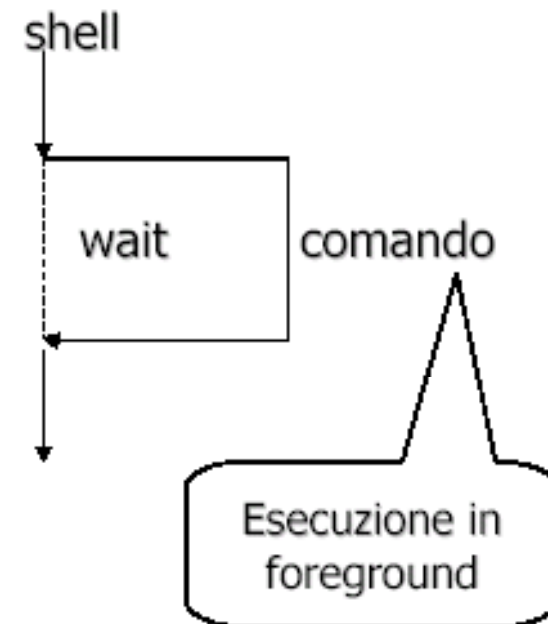
L'effetto totale è quello di inviare il  
contenuto del file `memo` come  
messaggio all'utente `user12`

Consente di inviare messaggi  
ad altri utenti

12

# Esecuzione in foreground

- La shell, utilizzando i servizi del kernel, crea un nuovo processo che esegue il comando, provvedendo a "passargli" in modo opportuno gli eventuali parametri
- Normalmente, dopo avere creato il processo che esegue il programma richiesto, la shell si pone in uno stato di wait, in attesa che il programma termini ...
- ... dopo di che, la shell riprende la sua esecuzione per acquisire il prossimo comando (riappare il prompt)

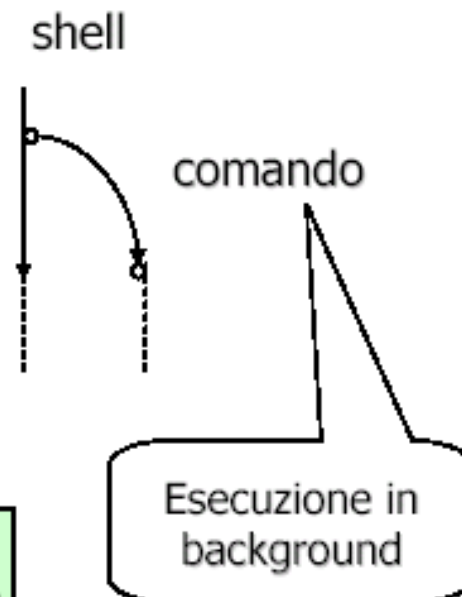


**Nota:** Il comando termina producendo un **exit code** intero, che sarà disponibile in una variabile di shell

# Esecuzione in background

`%comando argomenti &`

- In questo caso la shell crea il processo che esegue il comando richiesto, e, senza attenderne la terminazione, riprende subito la esecuzione, consentendo di lanciare altri comandi



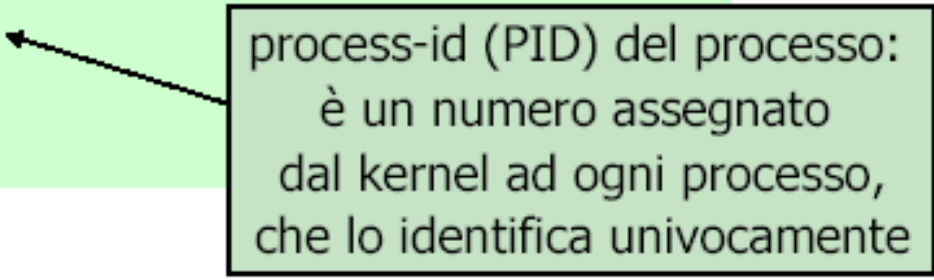
**Nota:** In questo caso l'**exit code** non sarà disponibile

**Esempio (sh) :**

```
$ cc prog.c > out &
```

```
10759
```

```
$
```



process-id (PID) del processo:  
è un numero assegnato  
dal kernel ad ogni processo,  
che lo identifica univocamente

Al logout dell'utente, eventuali processi in background vengono automaticamente terminati ...

... a meno che essi non siano stati "protetti" con il comando

**nohup** comando

# I PRINCIPALI COMANDI DI UNIX

## Formato dei comandi

**comando** [ *argomenti* ... ]

Gli *argomenti* possono essere:

- opzioni o flag (-)
- parametri

separati da almeno un separatore

### Esempio:

```
ls -l -F file1 file2 file3
```

          └──┬──┘     └──┬──┘  
          flag       parametri

### Forme equivalenti

```
ls -F -l file1 file2 file3  
ls -lF file1 file2 file3  
ls -Fl file1 file2 file3
```

# Convenzioni notazionali per i comandi

- termini in neretto devono essere scritti esattamente come appaiono
- termini in corsivo devono essere rimpiazzati da valori opportuni
- [ ] significa “opzionale”
- ... significa “una o più volte”

## Esempio:

```
ls [-aAcCdfFgiLqrRstu1] [ filename...]
```



# Il comando `who`

`who [ . . . ] [ am I ]`

- **who**: lista il nome, terminale, e data/ora di login di tutti gli utenti allocati
- **who am I**: come sopra, per l'utente che lo esegue

```
% who
marco pts/0 Mar 4 20:24 (alice.di.uniba.it)
sara pts/1 Mar 5 16:24 (194.20.15.99)
fabio pts/2 Mar 4 20:23 (alice.di.uniba.it)
%
% who am I
sara pts/1 Mar 5 16:24 (194.20.15.99)
```

# Il comando `id`

`id[-a]`

- `identifier`: visualizza user-id (uid), user name, group-id (gid), group name dell'utente (gruppo attivo)
- con l'opzione `-a`, visualizza tutti i gruppi di appartenenza

```
% id
uid=207(user11) gid=100(usrmail)
%
```

## Il comando `passwd`

000

passwd

- consente ad ogni utente di definire, e in seguito modificare, la propria password

```
%passwd
```

```
passwd: Changing password for user11
```

Old password:

New password:

Re-enter new password:

0.0%

# Il comando `passwd`

```
% passwd
passwd: Changing password for user11
Old password:
New password:
Passwords must differ by at least 3 positions
New password:
Password is too short - must be at least 6
characters.
New password:
Password must contain at least two alphabetic
characters and
at least one numeric or special character.
Too many failures - try later.
%
```

# Il comando `last`

**`last`**[username]

- **`last`** username: fornisce informazioni sull'ultimo login dell'utente identificato dallo username specificato
- **`last`**: fornisce informazioni sull'ultimo login di tutti gli utenti

```
%last user11
user11 pts/1 194.20.15.99 Sun Mar 5 16:24 still
logged in
user11 pts/1 194.20.15.99 Sun Mar 5 16:19 - 16:23(00:03)
wtmp begins Fri Sep 9 16:12
%
```

# Il comando `su`

~~~~~

**`su`**[username]

- **`s`**ubstitute **`u`**ser: permette di allocarsi come un diverso utente (identificato dallo username specificato), senza chiudere la sessione di lavoro
- Se l'argomento `username` manca, si assume `root` (il superuser) per default

```
%su
Password:
%
```



# Il comando `newgrp`

`newgrp` `groupname`

- **`newgroup`**: associa l'utente al gruppo specificato
- se il gruppo ha una password, e se l'utente non è membro di quel gruppo, il comando richiede la password

# Il comando `date`



**`date [ . . . ]`**

- Modifica o visualizza, nel formato specificato, la data e l'ora corrente

```
% date  
Mon Mar 4 16:41:05 MET 2002  
%
```

# Il comando `man`

- Unix ha un manuale di riferimento, accessibile in linea tramite il comando **`man`**
- Il manuale è organizzato in **sezioni** e **sottosezioni**
  - Ogni sezione è composta di **pagine** (logiche)
  - Ogni pagina descrive **un singolo argomento** (es.: un comando)
  - Ogni sezione o sottosezione inizia con una pagina chiamata `intro`

## **Esempio:**

- Sezione 1: Commands
- Sezione 2: System Calls
- Sezione 3: Library Functions
- Sezione 4: Administrative Files
- ...

# Il comando `man`

`man`

`man`[opzione ...]titolo ...

- Visualizza le pagine del manuale specificate mediante i suoi parametri
- Il parametro `-s` permette di specificare la sezione

```
% man who
. . .
%
% man -s 2 kill
. . .
% man -s 2 intro
```

Se il numero di sezione non è specificato, viene selezionata la prima occorrenza

# Il comando `man`

## Esempio di utilizzo

```
% man who
WHO (1)          USER COMMANDS WHO (1)
NAME
    who - who is logged in on the system
SYNOPSIS:
    who [ who-file ] [ am i ]
DESCRIPTION:
    .....
```

```
% man man
man(1) User Commands man(1)
NAME
man-find and display reference manual pages
SYNOPSIS
man [ - ] [ -adFlrt ] [ -M path ] [ -T macro-package ]
[ [ -s section ] title ... ] ...
man [ -M path ] -k keyword ...
man [ -M path ] -f filename ...
AVAILABILITY
SUNWdoc
DESCRIPTION
man displays information from the reference manuals.
It displays complete manual pages that you select by title, or
one-line summaries selected either by keyword (- k), or by
the name of an associated file (-f).
<omissis>
OPTIONS
-a Show all manual pages matching title within the MANPATH
search path. Manual pages are displayed in the order found.
<omissis>
```



```
% man man (CONTINUA)
USAGE
Sections
Entries in the reference manuals are organized into sections.
A section name consists of a major section name, typically a single
    digit, optionally followed by a subsection name, typically one or
    more letters. ...
<omissis>
FILES
/usr/share/man  root of the standard manual page directory subtree
<omissis>
SEE ALSO
apropos(1), cat(1), col(1), eqn(1), more(1), nroff(1), refer(1),
    tbl(1), troff(1), vgrind(1), whatis(1), catman(1M), eqnchar(5),
    man(5)
<omissis>
BUGS
The manual is supposed to be reproducible either on a Phototypesetter
    or on an ASCII terminal. However, on a terminal some information
    (indicated by font changes, for instance) is lost.
<omissis>
Sun Microsystems Last change: 14 Sep 1992
5
%
```

# Il comando `whatis`

**`whatis`** [`comando...`]

- Mostra solo la sezione NAME della pagina del manuale
- Equivale a `-man -f`

```
% whatis time date
time time (1) - time a command
time time (2) - get time
date date (1) - print and set the date
%
```

# Il comando `apropos`

`apropos` [word...]

- Cerca i comandi la cui descrizione nel manuale (NAME) contiene le parole specificate
  - Non distingue fra maiuscole e minuscole
- Equivale a `-man -k`

```
% apropos manual
```

```
catman catman (1m) - create the cat files for the manual
```

```
man man (1) - find and display reference manual pages
```

```
man man (5) - macros to format Reference Manual pages
```

```
mansun mansun (5) - macros to format Reference Manual pages
```

```
route route (1m) - manually manipulate the routing tables
```

```
whereis whereis (1b) - locate the binary, source, and
```

```
manual page files for a command
```

```
%
```

# I COMANDI PER IL CONTROLLO DEI PROCESSI

## Il comando `ps`



`ps [ opzioni ]`

- **process status**: visualizza alcune informazioni sui processi attivi
- Opzioni:
  - Nulla: solo i processi associati al terminale
  - **-f** lista estesa
  - **-e** tutti i processi

```
% ps
PID TTY TIME CMD
935 pts/5 0:01 csh
1193 pts/5 0:00 ps
```

**PID**: process-ID

**TTY**: terminale che controlla il processo

**TIME**: tempo cumulativo di esecuzione

**CMD**: comando

```
% ps -f
UID PID PPID C STIME TTY TIME CMD
roberto 10715 10713 161 10:25:28 pts/1 0:02 -csh
roberto 10885 10715 26 11:36:32 pts/1 0:00 ps -f
%
```

**UID:** process owner

**PID:** process-ID

**PPID:** parent process-ID

**STIME:** starting time: ora, minuto, secondo in cui il processo è stato creato

**TTY:** terminale che controlla il processo

**TIME:** tempo cumulativo di esecuzione

**CMD:** comando

# Controllo di processi

- Esistono vari comandi per il controllo dei processi (non in sh)
- Essi permettono, normalmente, di:
  - sospendere un processo
  - far ripartire un processo sospeso
  - **terminare** un processo
  - mandare in background un processo di foreground
  - portare in foreground un processo di background

# Sospendere l'esecuzione (la shell)



`sleep time`

- Sospende l'esecuzione per un intervallo di tempo specificato da `time` (numero di secondi)
- Durante quell'intervallo nessun comando può essere lanciato

`wait [processid]`

- Sospende l'esecuzione sino alla terminazione dell'esecuzione di un processo figlio (in background), identificato da `processid`.
- Se nessun argomento è specificato, sospende la esecuzione fino alla terminazione di **tutti** i processi creati dalla shell

# Sospensione e riattivazione di processi



- `$> Ctrl-Z` : sospende un processo  
eseguito in foreground
- `$> fg` : riattiva in foreground un processo  
sospeso
- `$> bg` : riattiva in background un processo  
sospeso



# Terminare un processo

- Un processo può essere terminato solo dal suo **proprietario** (l'utente che lo ha creato) o dal superuser
  - Per terminare un processo di foreground: CTRL+C
  - Per terminare un processo di background:  
`kill processid`

```
% sleep 10000 &  
[1] 11468  
% kill 11468  
[1] Terminated sleep 10000  
%
```

# Il comando `kill`

```
kill [- signal] processid ...
```

- Termina un processo o invia il **segnale** specificato ai processi indicati
- I processi possono ignorare il segnale ricevuto, oppure trattarlo
- Se non fanno nessuna di queste due cose, essi vengono terminati
- **Alcuni segnali:**
  - TERM -15: Terminazione del processo (è il segnale di default)
  - KILL -9: Terminazione del processo (non può essere ignorato né trattato)
  - STOP: Sospensione del processo (non può essere ignorato né trattato)
  - CONT Ripartenza di un processo sospeso

```
kill processid ...  
kill -15 processid ...  
kill -TERM processid ...  
kill -9 processid ...
```

Se un processo rimane orfano (viene terminata lo shell padre), esso viene adottato dal processo di sistema 1 (init)

```
% sh
$ sleep 10000 &
11336
$ ps -f
UID PID PPID C STIME TTY TIME CMD
roberto 11335 11274 38 15:10:30 pts/2 0:00 sh
roberto 11338 11335 26 15:10:49 pts/2 0:00 ps -f
roberto 11274 11272 176 14:57:04 pts/2 0:01 -csh
roberto 11336 11335 14 15:10:38 pts/2 0:00 sleep
10000
$ kill -9 11335 # qui -15 non basta !
Killed
% ps -f
UID PID PPID C STIME TTY TIME CMD
roberto 11341 11274 12 15:11:55 pts/2 0:00 ps -f
roberto 11274 11272 2 14:57:04 pts/2 0:01 -csh
roberto 11336 1 14 15:10:38 pts/2 0:00
sleep 10000
%
```

# LA COMPOSIZIONE DEI COMANDI IN UNIX

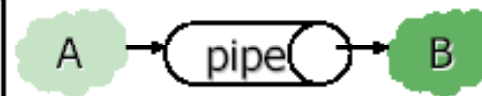
- In UNIX ogni comando è definito per effettuare una sola operazione
- Quando serve una nuova operazione, si preferisce combinare comandi esistenti piuttosto che costruirne di nuovi
  - **Compound commands:** ottenuti combinando comandi esistenti mediante simboli speciali () e ;  
Esempio: `(date;ls)> file`
- Le **pipeline** sono lo strumento fondamentale a supporto di questa filosofia

# Pipelines: pipe di comandi

- L'operatore di pipe | richiede alla shell di connettere fra loro due (o più) comandi, ridirigendo lo standard output del primo nello standard input del successivo, e così via

```
command1 | command2 | ...
```

- La shell, per eseguire `A | B`
  - crea un file temporaneo speciale, di tipo **pipe** (invisibile all'utente)
  - ridirige lo standard output di A sulla pipe, e crea un processo che esegue A
  - ridirige lo standard input di B sulla pipe, e crea un processo che esegue B
- Quando A e B terminano, la pipe è distrutta



I due processi si sincronizzano automaticamente

## Pipeline: esempi

```
% ps -fe | grep user11
user11 11453 11451 80 15:55:43 pts/1 :02 - csh
user11 11524 11453 10 16:12:32 pts/1 0:00 grep user11
user11 11523 11453 48 16:12:31 pts/1 0:00 ps -fe
%
```

# I filtri



- Sono programmi che trasformano il loro standard input nel loro standard output
- Sono molto utili per realizzare pipelines
- Alcuni filtri utili
  - `sort` ordinamento
  - `grep` seleziona linee di specificata struttura
  - `prep` scompone linee in parole
  - `uniq` sopprime linee ripetute
  - `head` mostra le prime linee
  - `tail` mostra le ultime linee
  - `wc` conta caratteri, linee, parole
  - `pr` impagina un testo

# Il comando `sort`



```
sort [ options] [ file...]
```

- ordina il file ottenuto concatenando i file indicati, e scrive il risultato sullo standard output
- se non è indicato alcun file, ordina lo standard input
- Opzioni:
  - r (reverse) inverte il senso dell'ordinamento
  - b ignora i blank iniziali (spazi e tabulazioni)
  - n sort numerico
  - .... e molte altre

# Il comando `sort`: esempio

```
% cat numeri
```

```
10  
1  
2  
5  
77  
750  
8
```

```
% sort numeri
```

```
1  
10  
2  
5  
750  
77  
8
```

```
% sort -r numeri
```

```
8  
77  
750  
5  
2  
10  
1
```

```
% sort -n numeri
```

```
1  
2  
5  
8  
10  
77  
750
```



# Il comando `grep`

```
grep[opzione] pattern [file]
```

- **g**lobal **r**egular **e**xpression **p**rint: cerca nei file specificati o (se mancano) nello standard input le linee che contengono il pattern (stringa), e le trasferisce sullo standard output
- Alcune opzioni:
  - c (count) produce solo il numero delle linee che contengono il pattern
  - v produce solo le linee che **non** contengono il pattern
  - n ogni linea prodotta è preceduta dal suo numero d'ordine nel file
  - ...

## Il comando `grep`: esempio

- Una semplice agenda telefonica

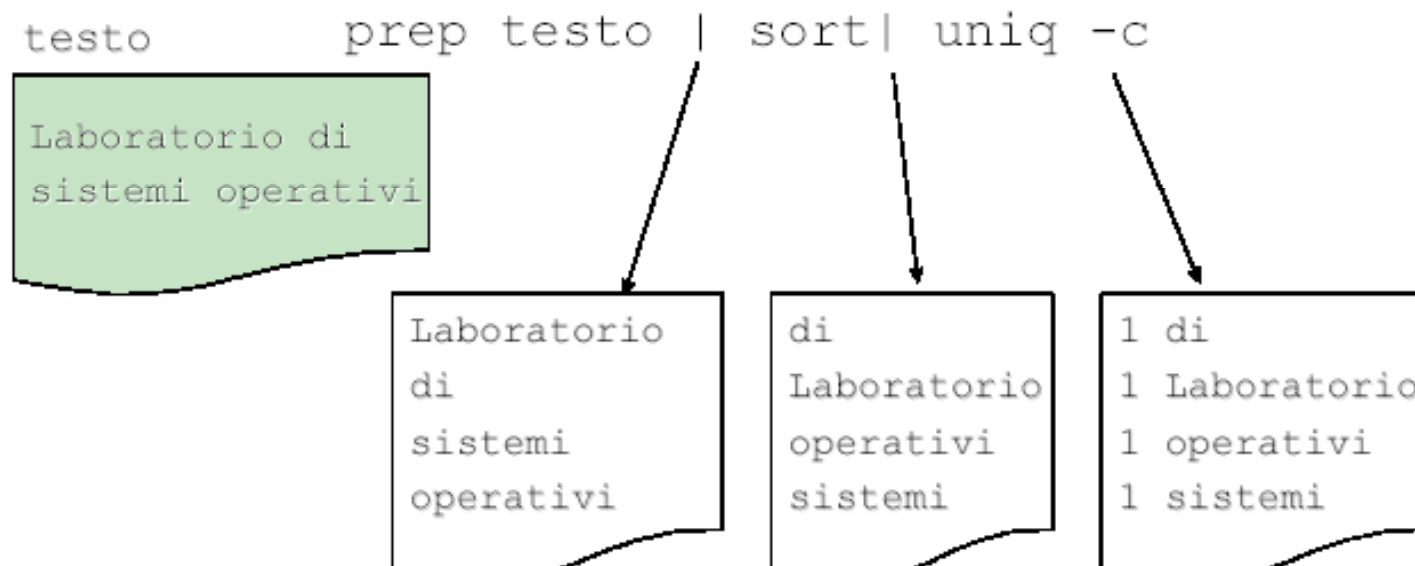
```
% cat > elencotel << :
> roberto 48000529
> marco 3452328
> mario 5567843
> luigi 67421467
> :
% grep marco elencotel
marco 3452328
%
```

# Il comando `grep`: esempi

```
% ls
dir1 dir2 exe1 exe2 file1file2
% ls -l -F
dir1/
dir2/
exe1*
exe2*
file1
file2
% ls -l -F | grep '^.*/*$'
dir1/
dir2/
% ls -l -F | grep -v '^.*[\\*/]$'
file1
file2
%
% ls -l -F | grep '^.*\\*$'
exe1*
exe2*
```

# Esempio di pipeline

- Listare in ordine alfabetico tutte le parole differenti di un testo, con accanto il numero di occorrenze



# Il comando `head`



```
head [ -n] [ filename...]
```

- Copia le prime `n` linee di ogni file sullo standard output (default: `n=10`)
- Se nessun file è specificato, copia linee dallo standard input

```
% cat elencotel  
roberto 48000529  
marco 3452328  
mario 5567843  
luigi 67421467  
% head -2 elencotel  
roberto 48000529  
marco 3452328  
%
```

A"

# Il comando `tail`

```
tail [ options] [ file]
```

- Copia il file sullo standard output, iniziando da un "posto" specificato (linee, blocchi o caratteri dall'inizio o dalla fine del file)
- Se non è specificato nessun file, copia lo standard input

```
% cat elencotel
roberto 48000529
marco 3452328
mario 5567843
luigi 67421467
% tail +3 elencotel
marco 3452328
luigi 67421467
%
```

# Il comando `wc`

`wc` [opzione] [file]

- Conta il numero di linee (`-l`), il numero di parole (`-w`), il numero di caratteri (`-C`) e il numero di byte (`-c`) di uno o più file
- L'opzione di default è `-lwc`

```
% wc elencotel  
4 8 66 elencotel
```

n. linee

n. parole

n. byte

# Paginatori



- Mostrano a video uno o più file, una pagina alla volta, ed eseguono comandi di scorrimento forniti dall'utente
- Ce ne sono vari:
  - `more`
  - `page`
  - `pg`

## ■ Uso di `more` in pipeline

- Il file da visualizzare può essere anche fornito a `more` sullo standard input
- Un uso tipico:

```
ls -R | more
```

Lista ogni sottodirectory

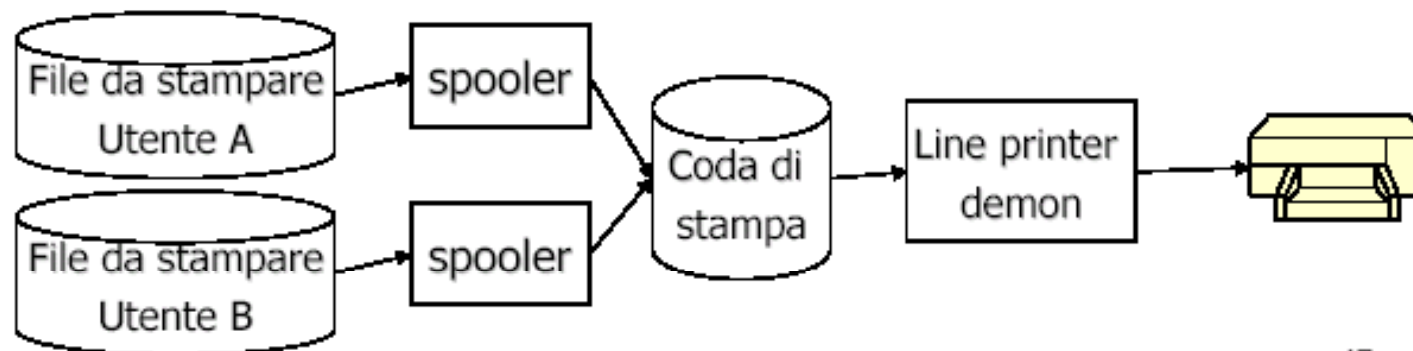


# Stampa di file

- Per stampare un file prodotto da un comando, si potrebbe semplicemente ridirigerne lo standard output sul file speciale che rappresenta la stampante, ad es.:

```
sort file > /dev/lp1
```

- Ma stampe concorrenti di più utenti risulterebbero interfoliate ...
- ... quindi è opportuno usare tecniche di spooling
- "**spool**"=simultaneous **p**eripheral **o**perations **o**ffline



# Il comando `lpr`

```
lpr [opzione] file
```

- **line printer spooler**: incoda la richiesta di stampa alla stampante