Breve Storia di UNIX

Nei '60, dominano *mainframe* da milioni di $. Hanno SO:

- enormi (milioni di righe assembler)
- quindi poco efficienti e
- pieni di errori (correggendoli se ne introducevano altri!)

Alla fine degli anni 60, fallisce il progetto del mega SO **MULTICS**.

Intanto compaiono i *minicomputer* (50% prestazioni, 5% costo di un mainframe).

Nel 1969 un ex progettista di MULTICS, **Ken Thompson** trova alla **Bell Labs** un minicomputer per cui scrive in assembler un SO: è una versione ridotta di MULTICS: **UNIX**.

Più avanti, Thompson e **Dennis Ritchie** riscrivono UNIX in **C**, linguaggio ad alto livello, molto conciso e potente.
Ciò si riflette nella filosofia di UNIX:

Piccolo è bello

Essendo ad alto livello, il C è indipendente dall'hardware

Ciò ha permesso di *portare* UNIX su pressoché ogni hardware, oggi PC, workstation, mini e super computer.

UNIX, in versioni lievemente diverse o standard (POSIX), è disponibile su tutti e domina dalle workstation in su.

La prima sessione

La versione di UNIX con cui lavoreremo è Linux.

Alla prima esercitazione lo schermo si presenterà così:

```
...
Login:
```

- accanto a `Login:` scrivete lo *user id* che vi è stato assegnato e con cui sarete noti al sistema
- il sistema risponde chiedendo `Password:`
- i caratteri battuti come password sono invisibili
- se password corretta, il sistema mostra un *prompt*

D'ora in poi, l'input dalla tastiera è rappresentato in **`grassetto`** e va terminato con il tasto `Enter` anche detto `Return`.

Dunque ogni seduta al terminale inizia così:

```
...
Login: user1
Password:
$
```

Errori su user id o password vengono notificati:

```
...
Login: utenet1
Password:
Login incorrect
```

Prima sessione: la shell, la riga di comando

Prompt: risposta con cui il sistema si mostra pronto per altro input dopo aver elaborato input precedente (qui la password)

Il prompt non viene da UNIX direttamente, ma da un programma detto ***shell*** che, come quelli che potete scrivere voi, gira **su** UNIX

I caratteri battuti da voi sulla tastiera sono riprodotti sullo schermo sulla ***riga di comando***, accanto al prompt.

La riga di comando viene presa in considerazione dalla shell solo quando premete il tasto etichettato `Enter` o `Return`.

Fino ad allora, il suo contenuto si trova in un'area provvisoria detta ***buffer di input*** e può essere corretto con il *tasto di cancellazione*.

Alcuni tasti hanno effetti di *controllo*, p.es. le combinazioni ottenute

- premendo il tasto `Ctrl` e,
- *mentre lo si tiene premuto*, un'opportuna lettera, poi
- lasciandoli entrambi

Alcune combinazioni importanti sono:

`Ctrl C`: interrompe l'esecuzione di un programma

`Ctrl S`: blocca l'output che scorre sullo schermo

`Ctrl Q`: sblocca l'output bloccato con `Ctrl S`

La shell come interprete dei comandi
Opzioni dei comandi

La shell interpreta e fa eseguire i vostri comandi.

P.es. il comando `who` serve a vedere gli utenti del sistema

```
$ who
user1   tty1   Nov 20 11:20
utente2   tty4   Nov 20 08:12
$ who am i
user1
```

Se la shell non riconosce un comando, ve lo dice. P. es.

```
$ DATE
DATE: not found
$ date
Mon Nov 16 15:40:27 EST 1992
$
```

Il comando `DATE` non esiste: UNIX distingue `D` da `d` etc.

Provate invece `date`

L'effetto di alcuni comandi si può modificare facendoli seguire da ***opzioni***

Le opzioni sono costituite da un `-` seguito da uno o più caratteri.

P. es. `date -u` mostra l'ora nel formato interno (ora di Greenwich)

```
$ date -u
Mon Nov 16 14:41:22 GMT 1992
```

Prima sessione, argomenti dei comandi , default

`cal` *anno* mostra il calendario di *anno* (NB 92≠1992)

```
$ cal 92
                                92
      January                 February                  March

...Provate a fermare l'output con Ctrl S e a farlo ripartire con Ctrl Q

      October                 November                  December
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6                  1  2  3                     1
 7  8  9 10 11 12 13    4  5  6  7  8  9 10    2  3  4  5  6  7  8
14 15 16 17 18 19 20   11 12 13 14 15 16 17    9 10 11 12 13 14 15
21 22 23 24 25 26 27   18 19 20 21 22 23 24   16 17 18 19 20 21 22
28 29 30 31            25 26 27 28 29 30      23 24 25 26 27 28 29
```

`cal` *mese anno* mostra il calendario di *mese* in *anno*

```
$ cal 9 1752
   September 1752
Su Mo Tu We Th Fr Sa
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

I dati da cui dipende il comportamento di un comando,
come `9` e `1752` in `cal 9 1752`, si dicono ***argomenti*** del comando.

Per dei comandi alcuni argomenti sono ***opzionali***.
Se mancano, si assumono per essi valori di ***default*** (=mancanza).
P.es. per default `cal` assume il mese attuale dell'anno attuale:

```
$ cal
   November 1992
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

Nella forma generale dei comandi, le ***parti opzionali*** sono tra [e].
P. es. la forma generale di `cal` è: `cal` [[*mese*] *anno*]

Cambio password - Uscita da una sessione

Per cambiare password:

```
$ passwd
New password:ciao        # NB: il testo tagliato non compare sul terminale
Re-enter new password:ciao
Password is too easily broken.  Try again.
New password:ciaocaro
Re-enter new password:ciaocaro
$ passwd
Old password:ciaocaro
New password:ciaocari
Re-enter new password:ciaocari
$ passwd
Old password:ciaocara
Sorry.
$
```

Per terminare una sessione ci sono tre possibilità:

- `exit`
- `Ctrl-D`
- `logout`

Quale funziona dipende dalla shell del vostro sistema.

Terminare la sessione è necessario per evitare che qualcun altro possa usare il sistema al vostro posto (cioè col vostro user-id).

Provate a concludere la sessione e quindi a rientrare:

Il file system

Dati e programmi in UNIX sono memorizzati su *file* (documenti).
Nozione astratta di file: sequenza di byte (sequenza di 8 bit).

I byte contenuti in un file possono avere diverse interpretazioni:

- rappresentazione di caratteri (es. `01100011 01101001` → `ci`)
- rappresentazione di interi (es. `01100011 01101001` → `26979`)
- istruzioni e dati per il computer

Ma l'interpretazione dipende dall'uso che si fa del file:
per UNIX ogni file è solo una sequenza di byte.

Inoltre questo vale qualunque sia il supporto fisico per i byte:
per file su disco, dischetto, nastro magnetico, CD, etc.

Ogni file ha un nome (*nome semplice* nel seguito) di max *n* caratteri;
sui primi Unix, *n*=14, caratteri leciti: `A...Z a...z 0...9 _ . ,`

UNIX distingue tra lettere maiuscole e minuscole: `nome`≠`noMe`.

Esempi: `lezione.doc lezione.old file_mio 19.nov.92`

Un modo semplice per mettere nel file `agenda` i byte che rappresentano i caratteri `domani vacanza` è:

```
$ echo domani vacanza > agenda
```

Per vedere sullo schermo il contenuto di `agenda`:

```
$ cat agenda
domani vacanza
$
```

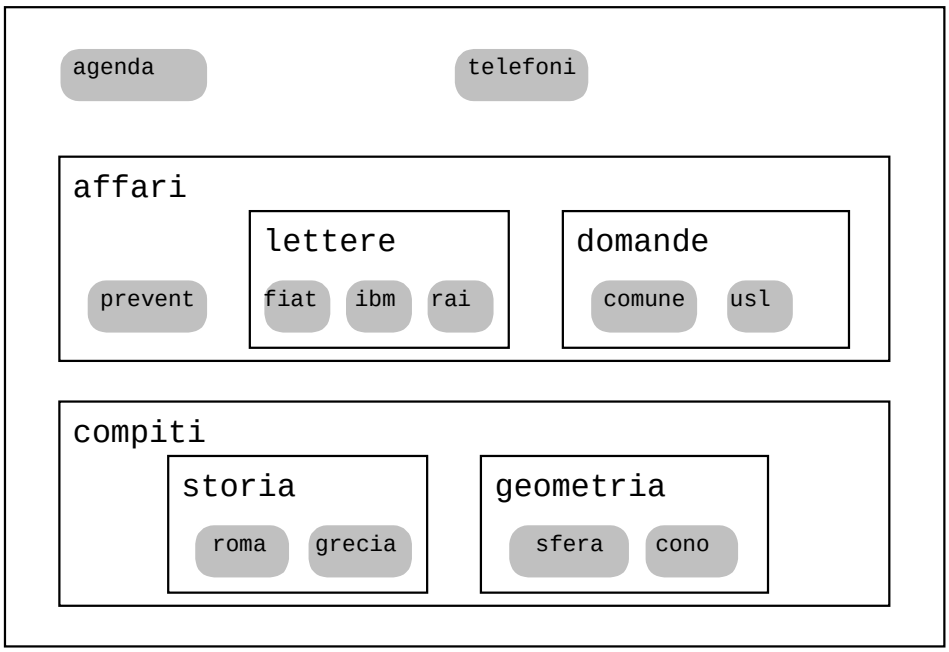
→ `cat` interpreta i byte come caratteri

Le directory

La *directory* si può immaginare come un “contenitore” di:

- file
- altre directory

Ogni utente ha sul sistema una directory di lavoro o *home directory*
Ad esempio la home directory di `user1` potrebbe raffigurarsi:

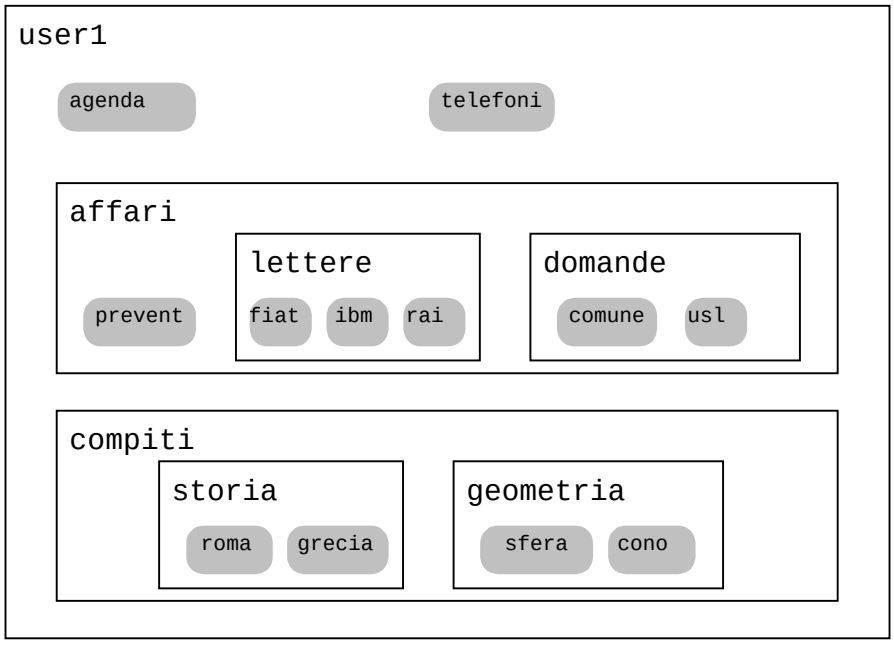


La directory `user1` contiene
la *sub-directory* `affari`, che contiene
le subdirectory `lettere` e `domande`,
la subdirectory `compiti`, che contiene
le subdirectory `storia` e `geometria`

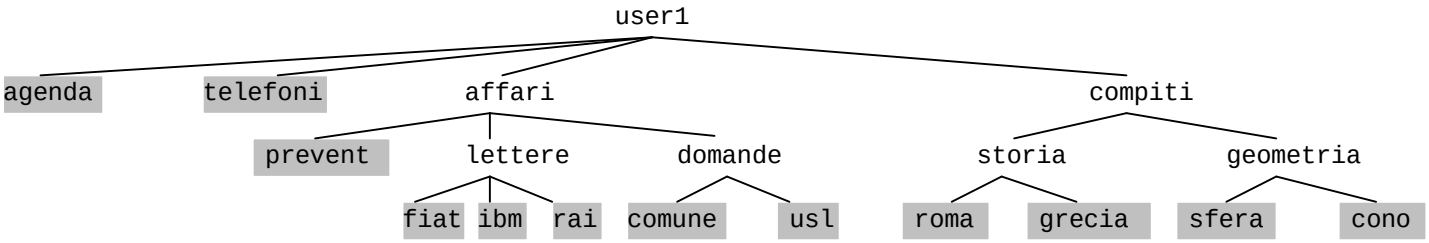
Inoltre, queste directory contengono i file mostrati in grigio.

L’albero delle directory

I contenuti delle directory si possono rappresentare, anzichè con delle scatole, come in



con degli *alberi* di cui file e directory sono i ***nodi***:



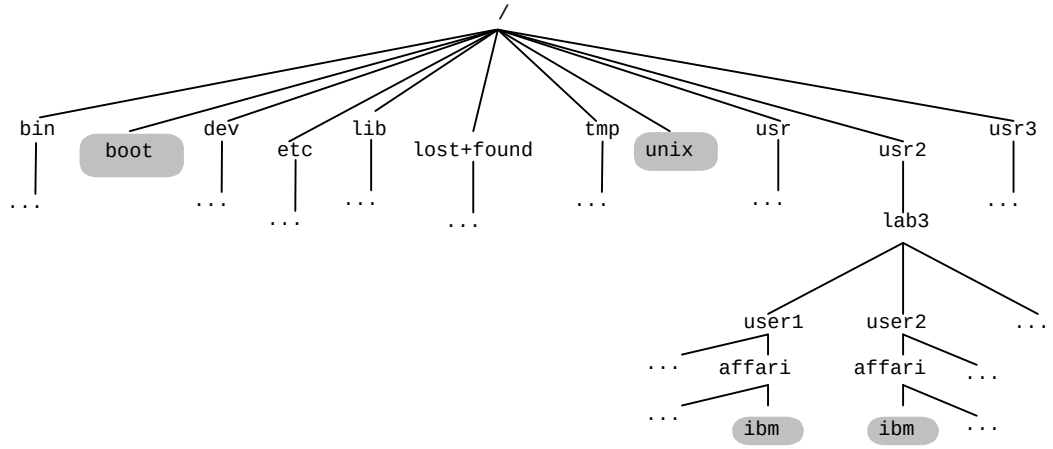
Ogni directory mostrata qui (tranne `user1`) ha un solo *genitore*, in cui è contenuta e di cui si dice *figlia*.

La struttura delle directory si dice *gerarchica* perché la relazione *genitore-figlio* determina una *gerarchia*.

Root, dir corrente, cammini e nomi di file

Di norma una directory come `user1` si trova dentro altre.
Ma esiste una directory che non è contenuta in nessun altra:
si chiama ***root*** (radice dell'albero) e si indica con la barra (slash) `/`

L'albero completo in cui si trova quello di `user1` a pag. 9 è:



Ad ogni istante è definita una ***directory corrente o di lavoro***.

File e directory, intesi come *nodi* di un albero, non possono essere individuati *univocamente* con un *nome semplice*, come p.es. `ibm`.

Per questo ogni file o directory con nome semplice *f*
ha un ***nome completo*** o ***pathname*** che può avere due forme:

- ***assoluta***: localizza *f* sull'albero *rispetto alla root*;
  è data dai nodi **da** `/` **fino** a *f* compreso, separati da `/`
  p.es. `/usr2/lab3/user1/affari/ibm`
- ***relativa***: localizza *f rispetto alla directory corrente*;
  è data dai nodi **dalla** directory corrente *esclusa*
  fino a *f compreso*, separati da `/`,
  p.es `affari/ibm` se la dir corrente è `/usr2/lab3/user1/`

NB: - un n. completo è assoluto se inizia per `/`, relativo altrimenti
- nei nomi il carattere `/` ha 2 usi: (1) root e (2) separatore di dir

Muoversi tra directory: pwd cd

Il comando pwd scrive sul terminale la directory corrente. P. es.

```
$ pwd
/usr2/lab3/user1
```

Inoltre, il vostro prompt può non essere $, ma la directory corrente, seguita da un contatore dei comandi dati:

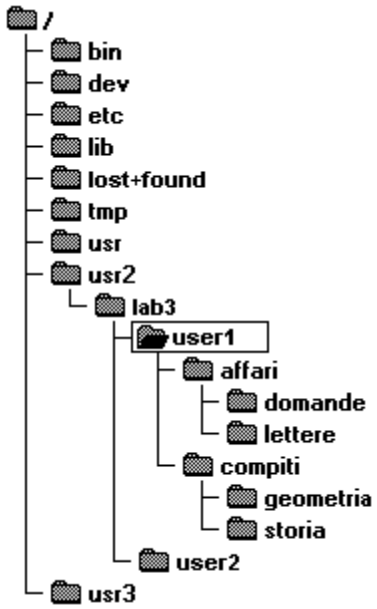
```
/usr2/lab3/user1 $ date
Sat Nov 14 10:23:50 EST 1992
/usr2/lab3/user1 $ pwd
/usr2/lab3/user1
```

Il comando cd rende *d* la nuova directory corrente

Il comando cd senza argomento rende la home dir corrente

serve a spostarsi nella directory di lavoro, in modo da dare nomi più brevi ai file di accesso più frequente

P. es. per chiamare il file /etc/passwd semplicemente passwd:




```
/usr2/lab3/user1 $ cat /etc/passwd
gp;;2;1;Giuseppe Pappalardo;/usr2/gp;/bin/sh
user1;;3;1;Franco;/usr2/lab3/user1;/bin/sh
/usr2/lab3/user1 $ cd /etc
/etc $ cat passwd
gp;;2;1;Giuseppe Pappalardo;/usr2/gp;/bin/sh
user1;;3;1;Franco;/usr2/lab3/user1;/bin/sh
/etc $ cd
/usr2/lab3/user1 $
```

Listare file: `ls`

Il comando `ls` mostra il contenuto della directory corrente

```
/usr2/lab3/user1 $ ls
affari
agenda
compiti
telefoni
```

Il formato completo di `ls` è: `ls` [*opzioni*] [*lista di file o dir*]

Le opzioni più importanti sono:

`ls -a`	elenca anche i file (normalmente invisibili) il cui nome comincia per `.`
`ls -l`	elenca in formato lungo
`ls -s`	elenca per dimensione crescente
`ls -t`	elenca a partire dal file più recente
`ls -C`	incolonna l'elenco
`ls -R`	elenca ricorsivamente anche le subdir

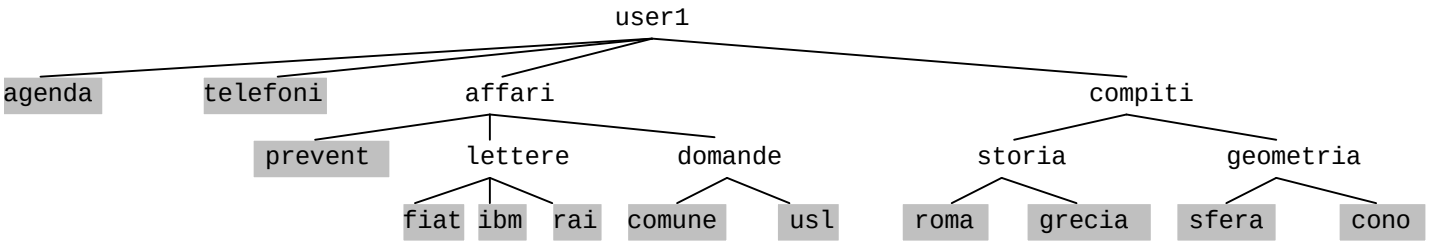
Esempi (notare come le opzioni di `ls` si possono combinare):

```
/usr2/lab3/user1 <120> ls -aC
.login   affari   agenda   compiti   telefoni
/usr2/lab3/user1 <121> ls -C affari
domande  lettere  prevent
/usr2/lab3/user1 <122> ls -C affari agenda
agenda

affari:
domande  lettere  prevent
/usr2/lab3/user1 <124> ls -l
total 6
drwxrwxrwx 1 user1              0 Nov 13 22:14 affari
-rwxrwxrwa 1 user1             16 Nov 14 11:52 agenda
drwxrwxrwx 1 user1              0 Nov 13 22:14 compiti
-rwxrwxrwa 1 user1            895 Nov 14 11:52 telefoni
/usr2/lab3/user1 <125> ls -lt
total 6
-rwxrwxrwa 1 user1            895 Nov 14 11:52 telefoni
-rwxrwxrwa 1 user1             16 Nov 14 11:52 agenda
drwxrwxrwx 1 user1              0 Nov 13 22:14 compiti
drwxrwxrwx 1 user1              0 Nov 13 22:14 affari
```

Listare file ricorsivamente: `ls -R`

Per vedere la struttura gerarchica dell'albero delle directory, cioè:



si può usare il comando `ls -R` *dir*

```
/usr2/lab3/user1 <126> ls -RC
affari    agenda    compiti   telefoni

affari:
domande  lettere  prevent

affari/domande:
comune  usl

affari/lettere:
fiat  ibm   rai

compiti:
geometria  storia

compiti/geometri:
cono   sfera

compiti/storia:
grecia  roma
```

Il comportamento di `ls -R` *dir* si dice *ricorsivo* perché mostra:

- il contenuto della directory *dir*,
- il contenuto delle subdirectory di *dir*, se ce ne sono
- il contenuto delle subdirectory delle subdirectory, se ce ne sono
- e così via

Caratteri jolly ? e *: nomi di file abbreviati

Per i nomi di file si possono usare i caratteri *jolly* (*wildcard*) ? e *

? sta per qualsiasi carattere

* sta per una sequenza arbitraria (di 0, 1, 2, 3...) caratteri.

```
/usr2/lab3/user1 $ echo sabato trippa > agenda1
/usr2/lab3/user1 $ echo domenica gnocchi > agenda2
/usr2/lab3/user1 $ ls -C agenda?
agenda1    agenda2
/usr2/lab3/user1 $ ls -C agenda*
agenda     agenda1     agenda2
/usr2/lab3/user1 $ ls -C ag*
agenda     agenda1     agenda2
```

Un “jolly” che rimpiazzi insiemi di caratteri più piccoli che non ? si ottiene racchiudendo l’insieme desiderato tra [e]. P. es.

[aef] sta per a oppure e oppure f

[1-5] sta per 1 oppure 2 oppure 3 oppure 4 oppure 5

Queste tecniche si possono combinare. P. es.

```
/usr2/lab3/user1 $ ls -C ag*a?
agenda1  agenda2
/usr2/lab3/user1 $ echo merc parto > agenda4
/usr2/lab3/user1 $ echo ven digiuno > agendex
/usr2/lab3/user1 $ ls -C ag*[ae][x1-4]
agenda1  agenda2  agenda4  agendex
```

È la shell che espande ? e *, per ogni comando (non solo ls).

```
/usr2/lab3/user1 $ cd aff*/let*
/usr2/lab3/user1/affari/lettere $ cd
/usr2/lab3/user1 $
```

Adesso cancellate tutte le nuove agende ma non agenda:

```
/usr2/lab3/user1 $ rm agend??
/usr2/lab3/user1 $ ls age*
agenda
```

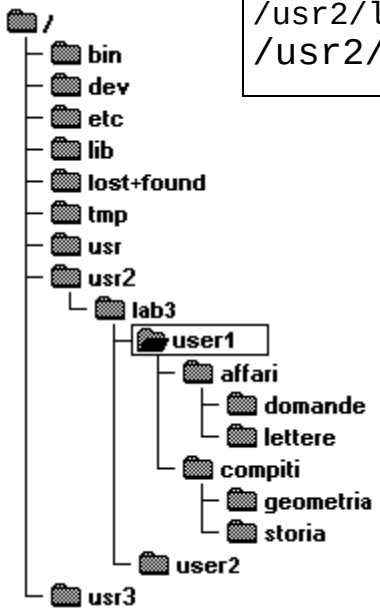
Le directory . e . .

I caratteri . e . . sono nomi speciali di directory:

- . rappresenta la directory corrente
- . . rappresenta la directory genitore di quella corrente

L'esempio sotto va letto ricordando il solito albero delle directory:

```
/usr2/lab3/user1 $ ls -C
affari    agenda    compiti   telefoni
/usr2/lab3/user1 $ ls -C .
affari    agenda    compiti   telefoni
/usr2/lab3/user1 $ cd aff*/let*
/usr2/lab3/user1/affari/lettere $ ls -C ../..
affari    agenda    compiti   telefoni
/usr2/lab3/user1/affari/lettere $ cd ../lettere
/usr2/lab3/user1/affari/lettere$ cd ../dom*/../../../user1
/usr2/lab3/user1 $
```



Creare e rimuovere directory: `mkdir` e `rmdir`

`mkdir` *d* crea una directory di nome *d*

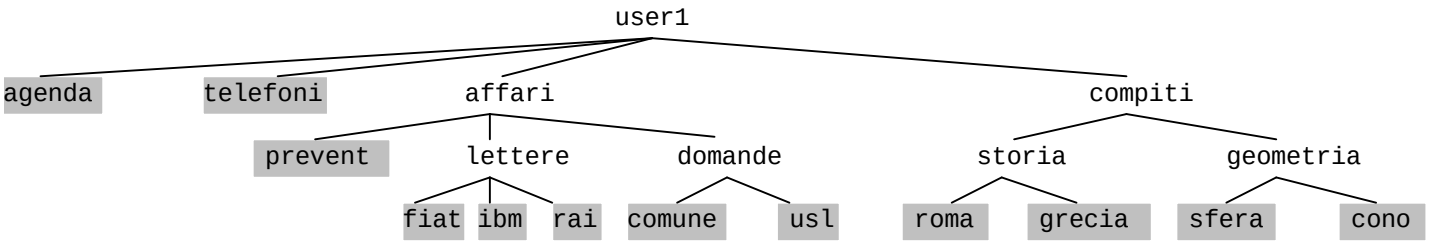
Si ha un errore se *d* esiste o se:
d = [/]d_1/d_2/ . . . /d_n/d_{n+1} e [/]d_1/d_2/ . . . /d_n non esiste

`rmdir` *d* cancella la directory *d* purché sia vuota

```
/usr2/lab3/user1 $ mkdir tmp
/usr2/lab3/user1 $ ls -l
total 6
drwxrwxrwx 1 user1              0 Nov 13 22:14 affari
-rwxrwxrwa 1 user1           1747 Nov 14 11:52 agenda
drwxrwxrwx 1 user1              0 Nov 13 22:14 compiti
-rwxrwxrwa 1 user1            895 Nov 14 11:52 telefoni
drwxrwxrwx 1 user1              0 Nov 15 11:58 tmp
/usr2/lab3/user1 $ mkdir tmp/d1/d2
mkdir: path not found for "tmp/d1/d2"
/usr2/lab3/user1 $ mkdir tmp/d1
/usr2/lab3/user1 $ ls -l tmp/d1
total 0
/usr2/lab3/user1 $ rmdir tmp/d1
/usr2/lab3/user1 $ ls -l tmp/d1
ls: File or directory "tmp/d1" is not found
```

Esercizio

Create nella vostra home directory (p.es. `user1`) la struttura:



usando: **`mkdir` *`d`*** per creare le directory *d*

`echo . . . > f` per creare il file *f* contenente il testo ...

quindi eseguite tutti i comandi mostrati nei riquadri finora e successivamente.

Copiare file: cp

cp *f1* [*f2* ...] *dir* crea delle copie dei file *f1*... dentro *dir*

- *dir* deve esistere come directory
- se *f1* esiste già dentro *dir* viene sovrascritto

cp *f1 f2* crea una copia del file *f1* di nome *f2*

- *f2* deve essere un file o non esistere (altrimenti vedi caso precedente)
- se *f2* esiste già viene sovrascritto

```
/usr2/lab3/user1 $ ls -C
affari    agenda    compiti   telefoni   tmp
/usr2/lab3/user1 $ cp agenda telefoni tmp
/usr2/lab3/user1 $ ls -l tmp
total 6
-rwxrwxrwa 1 user1 0            17 Nov 15 23:40 agenda
-rwxrwxrwa 1 user1 0           895 Nov 15 23:40 telefoni
/usr2/lab3/user1 $ echo domani lavoro > agenda1
/usr2/lab3/user1 $ cp agenda1 tmp/agenda
/usr2/lab3/user1 $ ls -l tmp
total 3
-rwxrwxrwa 1 user1 0            15 Nov 14 23:44 agenda
-rwxrwxrwa 1 user1 0           895 Nov 14 23:40 telefoni
```

Con cp -r *f1* [*f2* ...] *dir*, se *f1...* è una directory,
viene copiata ricorsivamente, cioè insieme alle sue subdirectory.

```
/usr2/lab3/user1 $ cp -r affari tmp
/usr2/lab3/user1 $ ls -RC tmp/affari

tmp/affari:
domande  lettere  prevent

tmp/affari/domande:
comune  usl

tmp/affari/lettere:
fiat  ibm   rai
```

Spostare file: mv

mv *f1* [*f2* ...] *dir* sposta *f1...* dentro *dir*

- *dir* deve esistere come directory
- se *f1* esiste già dentro *dir* viene sovrascritto
- *f1* [*f2* ...] può essere una directory (verrà copiata ricorsivamente dentro *dir*)

mv *f1 f2* cambia il nome di *f1* in *f2*

- *f2* deve essere un file o non esistere (altrimenti vedi caso precedente)
- se *f2* esiste già viene sovrascritto

```
/usr2/lab3/user1 $ ls -C
affari    agenda    compiti   telefoni    tmp
/usr2/lab3/user1 $ echo ciao > tmp/saluto
/usr2/lab3/user1 $ ls -C tmp/saluto
tmp/saluto
/usr2/lab3/user1 $ mv tmp/saluto .
/usr2/lab3/user1 $ ls -C
affari    agenda    compiti   saluto   telefoni    tmp
/usr2/lab3/user1 $ ls -C tmp/saluto
ls: File or directory "tmp/saluto" is not found
/usr2/lab3/user1 $ mv saluto tmp/file.ciao
/usr2/lab3/user1 $ ls -C
affari    agenda    compiti   telefoni    tmp
/usr2/lab3/user1 $ ls -C tmp/fil*
tmp/file.ciao
/usr2/lab3/user1 $
```

Uso di `cp` e `mv`

Nell'uso di `cp` e `mv` possono darsi vari casi per gli argomenti *source* (sorgente da cui si copia) e *target* (su cui si copia).

Ciascuno degli argomenti può essere:

- nome di file o
- nome di directory o
- nome con cui non esistono file o dir

Non tutte le possibilità però sono ammesse:

`cp`	*source*	*target*	Result
	file	*file*	Ok (sovrascive)
	file	*dir*	Ok (copia dentro dir)
	file	*non esiste*	Ok (crea copia)
	dir	*qualsiasi*	errore: *dir* is a directory
	non esiste	*qualsiasi*	errore: No such file or directory

`cp` *src1 src2 dir* equivale a `cp` *src1 dir* e `cp` *src2 dir*, quindi:

`cp`	*src1*	*src2*	*Target*	Result
	file	*non file*	*dir*	errore: copia solo il file *src1* dentro *dir*
	non file	*file*	*dir*	errore: copia solo il file *src2* dentro *dir*
	qualsiasi	*qualsiasi*	*non esiste*	errore: *target* must be a directory
	qualsiasi	*qualsiasi*	*file*	errore: *target* must be a directory

`mv`	*source*	*target*	Result
	file	*file*	Ok
	file	*dir*	Ok
	file	*non es.*	Ok (rinomina)
	dir	*dir*	Ok (sposta *source* dentro *target* ricorsivamente)
	dir	*non es.*	Ok (rinomina)
	dir	*file*	errore: `cannot rename`
	non esiste	*qualsiasi*	errore: No such file or directory

Cancellare file: `rm`

`rm` *f1 f2 ...* cancella (**ri*m*uove**) i file *f1 f2 ...*

`rm -r` *dir* cancella la directory *dir* insieme con tutte le subdirectory

Attenzione: ***non*** c'è modo di recuperare i file cancellati!
`rm *` e `rm -r` *dir* sono ***molto*** pericolosi!

```
/usr2/lab3/user1 $ echo squola > errore
/usr2/lab3/user1 $ ls errore
errore
/usr2/lab3/user1 $ cat errore
squola
/usr2/lab3/user1 $ rm errore
/usr2/lab3/user1 $ ls errore
ls: File or directory "errore" is not found
/usr2/lab3/user1 $
```

A pag. 18 si era copiato ricorsivamente `affari` in `tmp`:

```
/usr2/lab3/user1 $ ls -RC tmp/affari

tmp/affari:
domande  lettere  prevent

tmp/affari/domande:
comune  usl

tmp/affari/lettere:
fiat  ibm   rai
```

si può provare a cancellare `tmp/affari` così

```
/usr2/lab3/user1 $ rmdir tmp/affari
rmdir: non-empty directory "tmp/affari"
/usr2/lab3/user1 $ rm -r tmp/affari
/usr2/lab3/user1 $ ls -C tmp/affari
ls: File or directory "tmp/affari" is not found
```

Esercizio finale su file e directory

1. Nella propria home directory, creare una directory `mondo`,
2. in questa, creare le 5 directory:

   `africa america asia europa oceania`
3. all'interno di ciascuna directory-continente creare almeno 5 directory chiamate con nomi di paesi di quel continente
4. dentro ciascuna directory-paese creare almeno 5 file chiamati con nomi di città di quel paese;
   il file-città può essere creato con `echo` e contenere un numero di fantasia (la popolazione della città), p. es.:

   `echo 380000 > catania`
5. alla fine, eseguire `ls -RC` per vedere il risultato
   (consiglio: per bloccare lo schermo, premere `Ctrl S`,
   per sbloccarlo, premere `Ctrl Q`)
6. cancellare `mondo` e il suo contenuto senza usare `mkdir`

Un esempio di come potrebbe apparire l'albero della directory `mondo` durante l'esercizio.

```
mondo
+---africa
+---america
¦   +---usa
¦   ¦       new_york
¦   ¦
¦   +---canada
+---asia
+---europa
¦   ¦
¦   +---germania
¦   ¦       bonn
¦   ¦
¦   +---italia
¦           catania
+---oceania
```