Prima Esercitazione

Linux shell e linguaggio C

Linux / Unix: la shell

comandi, file system, utenti e gruppi

Accesso a Linux: *login*

- Per iniziare una sessione bisogna essere in possesso di una *combinazione*:
  - username (es. **`x135462`**, **`dll28493`**, ...)
  - password (es. **`dfh@2#q`**, **`**a890`**, **`aPP&x.`**, ...)
- nota: maiuscole / minuscole sono caratteri diversi! (**`**a890`** ≠ **`**A890`**)
- Accesso al sistema: **`login: x135462`**
  `Password: ******`

NB: per ottenere le credenziali per accedere alle macchine dei laboratori:

`https://infoy.ing.unibo.it/`

Linux: desktop e SHELL

- aprire il browser web (es. mozilla firefox) e collegarsi al sito web del corso:
  - applicazioni -> mozilla firefox
  - caricare http://lia.deis.unibo.it/Courses/sot1415
- gestione di finestre di shell (terminale):
  - aprire una nuova shell (applicazioni -> accessori-> terminale)
  - eseguire i comandi "**`date`**" "**`whoami`**" "**`who`**"
  - provare: "**`man`**" "**`man date`**" "**`man man`**"
  - uscire dallo shell: "**`exit`**" "**`CTRL+D`**"

il comando `passwd`

È possibile cambiare la propria password di utente, mediante il comando **`passwd`**

Verrà prima chiesta la vecchia password (per motivi di sicurezza)

NB: Se ci si dimentica della password, bisogna chiederne una nuova all'amministratore di sistema (utente *root*)

- Modificare la password con **`passwd`**
- Ripristinare la vecchia password

il comando pwd

Dopo il login, l'utente può cominciare a operare all'interno di uno specifico direttorio (la sua home).

- Visualizzare il direttorio corrente con **pwd** (print working directory)

```
dloreti@cloudpS:~$ pwd
/home/dloreti
```

il comando `ls`

provare il comando **`ls [-opzioni...] [file...]`**

- creare qualche file (vuoto):

  `ls –l` (list directory - as a List)

  `ls –la` (list directory - as a List -All)

  `ls a*` …

- creare qualche file (vuoto):

"**`> a1.txt`**" "**`> a2.txt`**" e riprovare "**`ls a*`**"

> **`*`** corrisponde a qualunque stringa, anche vuota.
> E' un metacarattere!

- Provare: **`ls –l /`** **…cosa fa?**

Filesystem hierarchy standard (FHS)

- **/** : Root
- **/bin** : File binari dei comandi essenziali
- **/sbin** : File binari dei comandi di sistema essenziali
- **/home** : Home degli utenti
- **/var** : Dati variabili
- **/boot** : File statici per operazioni di boot (avvio) della macchina
- **/dev** : File dispositivi
- **/etc** : File di configurazione
- **/lib** : Shared libraries e moduli del kernel
- **/media** : *Mount point* per *media* rimovibili
- **/mnt** : *Mount point* per operazioni di mount temporanee di FS
- **/opt** : Software applicativi
- **/tmp** : File temporanei
- ...

http://www.pathname.com/fhs/

Una digressione: il comando `mount`

- Collega un file system esterno in una certa posizione della gerarchia di file.
- Quando colleghiamo uno storage esterno (es: chiavetta USB), linux aggiunge un file in /dev/. La nuova device non è ancora utilizzabile. Occorre fare:

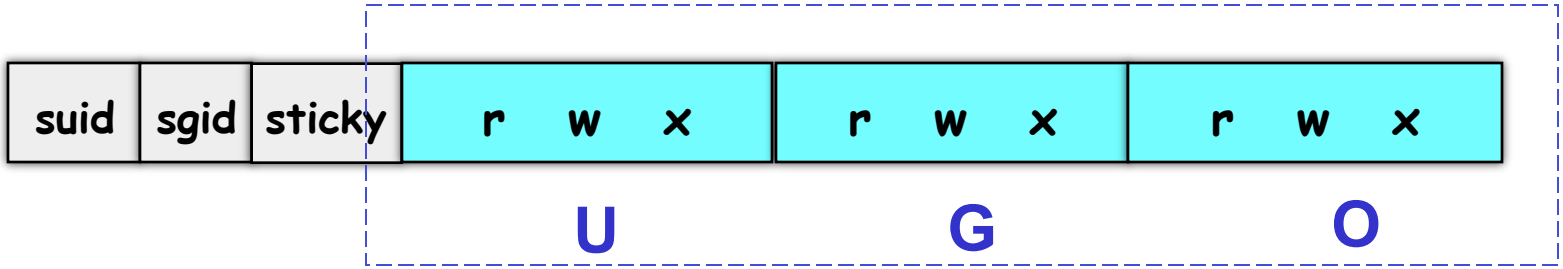
  `mount [device] [directory]`

  - **`[device]`**: nome del file in /dev/ . Rintracciabile con comando **`fdisk -l`**
  - **`[directory]`** : nome directory a cui collegare il contenuto della USB.
- Moderne distro linux supportano l'automatic mount

protezione

proprietà, accessi, bit di protezione

Bit di Protezione:
lettura, scrittura, esecuzione

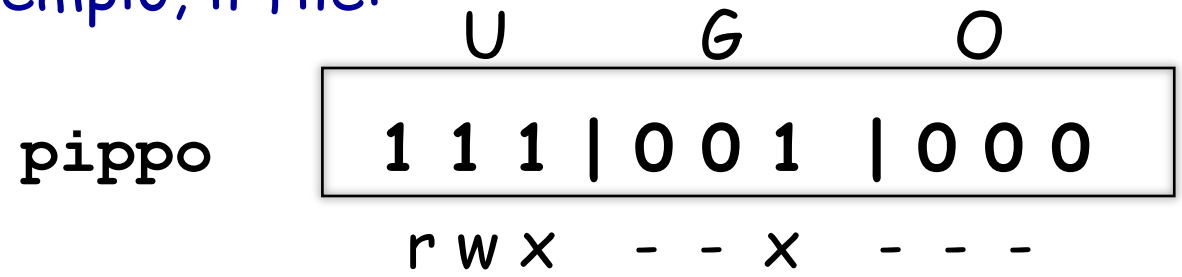


9 bit di lettura (read),scrittura (write), esecuzione(execute) per:

- utente proprietario (**U**ser)
- utenti del gruppo (**G**roup)
- tutti gli altri utenti (**O**thers)

bit di protezione:
lettura, scrittura, esecuzione

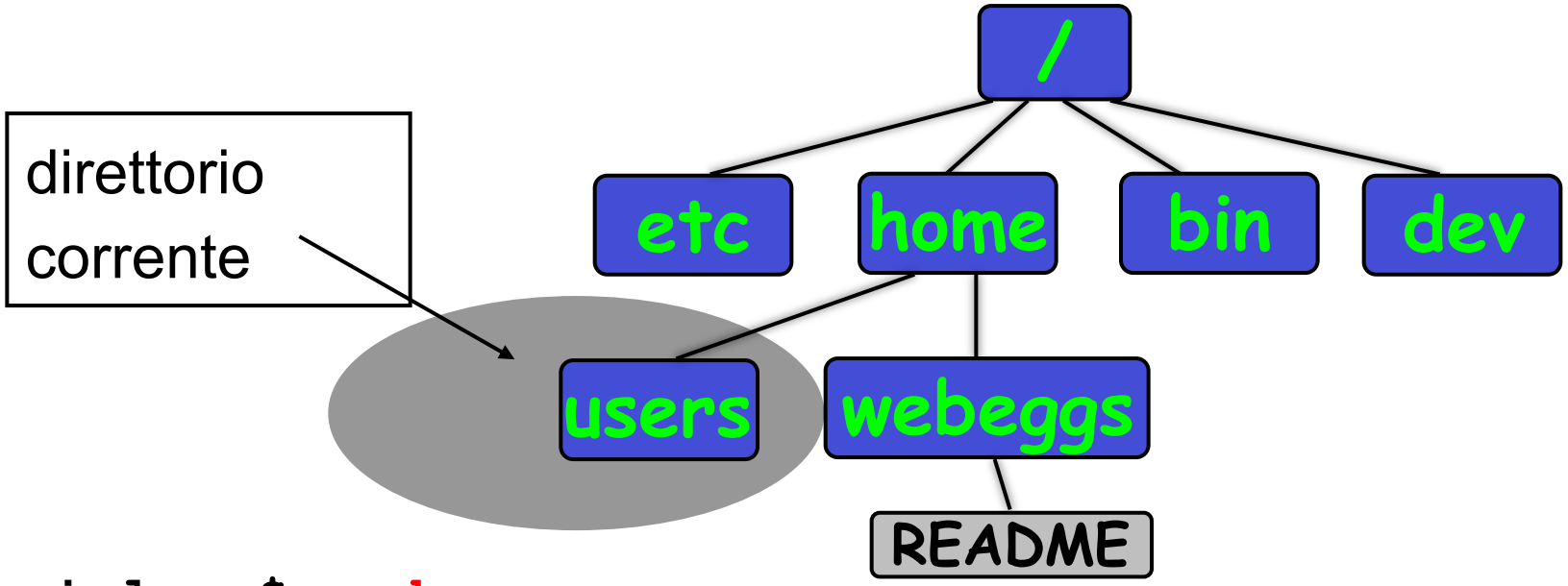
Ad esempio, il file:



- è leggibile, scrivibile, eseguibile per il proprietario
- è solo eseguibile per gli utenti dello stesso gruppo
- nessun tipo di modalità per gli altri

- formato ottale: 111 => 7; 010 => 2; ... -rwx--x--- => 0710

nomi relativi / assoluti: *esempio*




```
daniela:-$ pwd
 /home/users
daniela:-$ ls -l /home/webeggs/README        (1)
daniela:-$ ls -l ../webeggs/README           (2)
```

(1) e (2) sono equivalenti

comandi per la gestione del file system

cd, rm, cp, cat, mv, mkdir, rmdir, chmod, chgrp, chown

il comando `cd`: change directory

È possibile '**spostarsi**' da un direttorio attraverso il comando `cd`. La sintassi è:

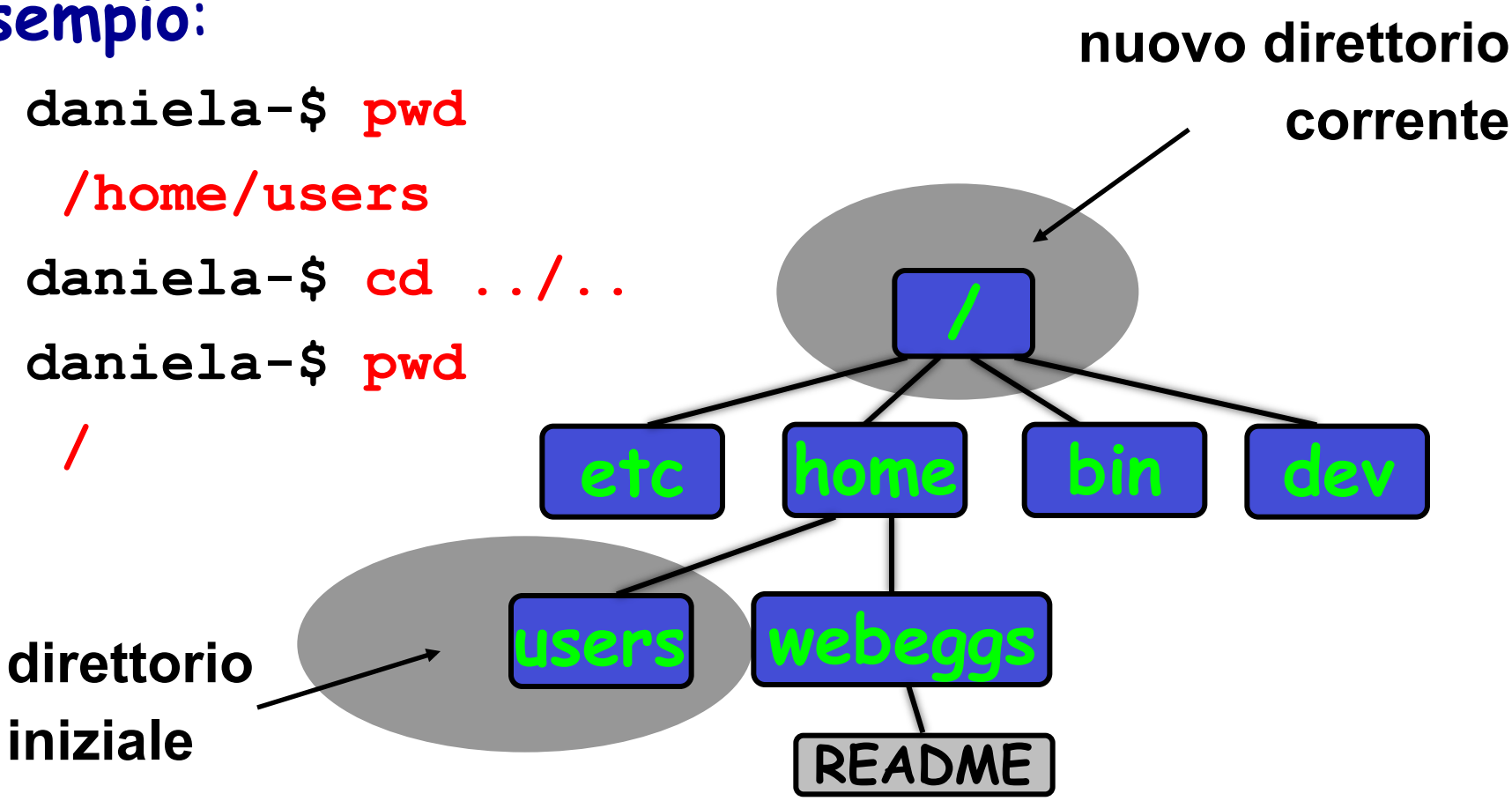
```
cd [<nuovo direttorio>]
```

- il direttorio destinazione si può esprimere con il nome relativo oppure assoluto
- se l'argomento non viene specificato, il nuovo direttorio è la home directory dell'utente
- per spostarsi all'interno di un determinato direttorio bisogna avere per tale direttorio i diritti di esecuzione

il comando cd

Esempio:

```
daniela-$ pwd
  /home/users
daniela-$ cd ../..
daniela-$ pwd
  /
```

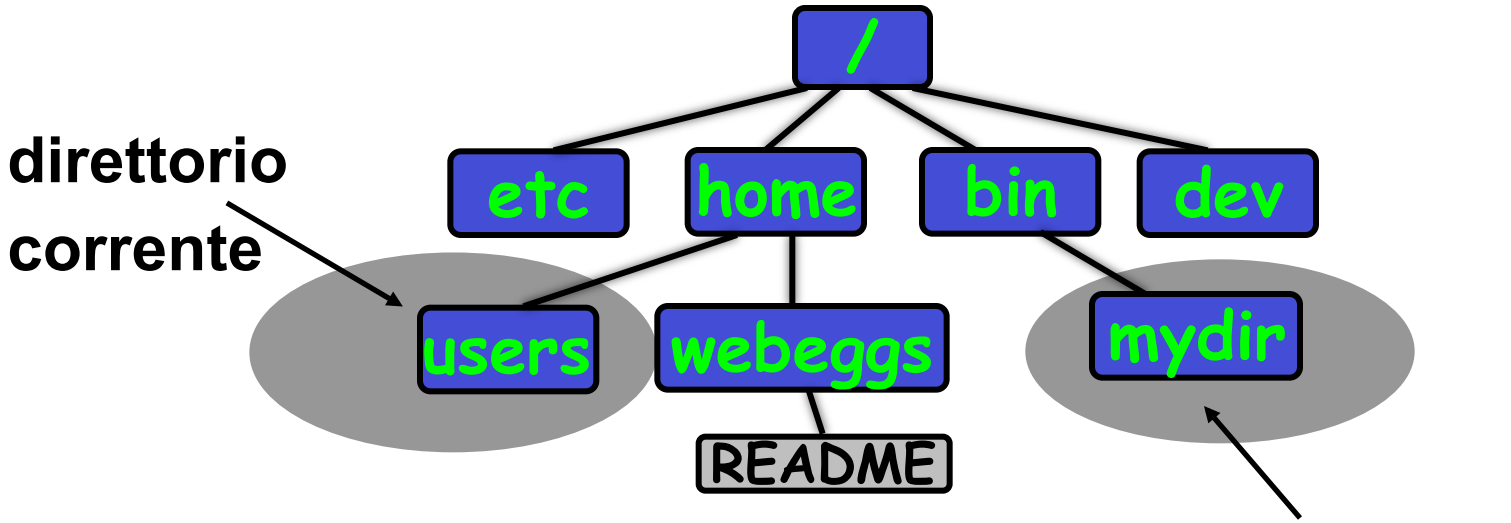


modifica del file system: direttori

- Creazione di un direttorio: **`mkdir <nomedir>`**
- Eliminazione : **`rmdir <nomedir>`**
- per creare un direttorio è necessario avere i diritti di scrittura nel direttorio all'interno del quale lo si vuole inserire
- per eliminare un direttorio è necessario avere i diritti di scrittura di tale direttorio

Esempio di `mkdir`

creo un nuovo direttorio utilizzando il percorso relativo o assoluto:




```
:~$ mkdir /bin/mydir
:~$ mkdir ../../bin/mydir
```

lettura di file di testo

- è necessario avere i diritti di lettura per visualizzare il contenuto di un file di testo
- **`cat [<nomefile>...]`**: visualizza l'intero file
- **`more [<nomefile>...]`**: visualizza per videate
- altri comandi:
  - **`grep <stringa> [<nomefile>...]`** (ricerca di una stringa in un file),
  - **`wc [-lwc] [<nomefile>...]`** (conteggio di righe / parole / caratteri)

cancellazione, copia e spostamento di file

- eliminazione di un file:

  `rm <nomefile>`

- copia di un file (e diritti):

  `cp <nomefile> <nuovofile>`

- spostamento di un file (e diritti):

  `mv <nomefile> <nuovofile>`

- è necessario avere i diritti di scrittura sul file per modificarlo

esempi

```
:~$ cp quasimodo sera
:~$ ls
quasimodo       sera
```

(copia il file)

```
:~$ mv quasimodo poesia
:~$ ls
poesia          sera
```

(sposta/rinomina il file)

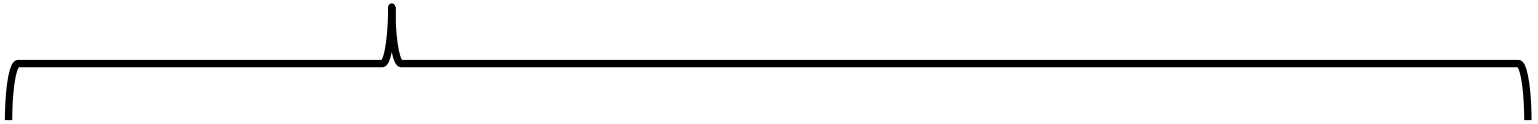
```
:~$ rm poesia
:~$ ls
sera
```

NB: Per eliminare un file occorre avere diritto di scrittura sulla directory che lo contiene

modifica dei bit di protezione

- **chown / chgrp** permettono di modificare la proprietà di un file
- è possibile cambiare i permessi dei propri file attraverso il comando chmod:

```
chmod <mode> <nomefile>
```



```
[ugoa][[+-=][rwxXstugoa...]... ][,...]
```

oppure: formato ottale dei bit di protezione

esempio chmod

```
:~$ ls -l file1.c
-rw-rw-r--  1   anna   staff ... file1.c

:~$ chmod 0666 file1.c
:~$ ls -l file1.c
-rw-rw-rw-  1   anna   staff ... file1.c

:~$ chmod a-w,u=rw file1.c
:~$ ls -l file1.c
-rw-r--r--  1   anna   staff ... file1.c
```

$([6]_{10} = [110]_2)$

esempi modifica file (diritti)

```
$ ls -l quasimodo
-rw-r--r-- 1    loreti staff ... quasimodo
$ chmod 0400 quasimodo
$ mv quasimodo subito
$ ls -l subito
-r-------- 1    loreti staff ... subito
$ rm subito
rm: remove 'subito',overriding mode 0400[y/n]?
```

($[4]_{10} = [100]_2$)

E' una feature di **rm**, **NON** un fatto di permessi!
Il permesso di cancellare un file dipende dai permessi di directory

Programmare in linguaggio C

editor, compilatore, parametri

Editor

- Esistono vari editor offerti dal sistema:
  - vi: editor standard UNIX (1976)
  - Vim: (vi improved) versione estesa presente in tutte le distribuzioni unix
  - Kate, Kwrite (kde)
  - Gedit (gnome)
  - Emacs
  - ...

Compilazione sorgenti - C

- un file.c non è direttamente eseguibile dal processore. Occorre tradurlo in linguaggio macchina, compilarlo.
- Comando **`gcc <file>`**:
  - compila **`<file>`** producendo il **file eseguibile** **`a.out`**
  - per dare nome diverso al file prodotto: opzione **`-o`**
- Esempio: **`gcc file_exec.c -o f_ex`**
- occorre rendere f_ex eseguibile:

  `chmod u+x f_ex`
- Esecuzione: **`./f_ex <parametri>`**

Parametri della linea di comando: *argc, argv*

```
main (int argc, char *argv[]){ }
```

- **`int argc`**: rappresenta il numero degli argomenti effettivamente passati al programma; anche il nome stesso del programma (nell'esempio, f_ex) e` considerato un argomento, quindi argc vale sempre almeno 1.
- **`char **argv`**: vettore di stringhe, ciascuna delle quali contiene un diverso argomento. Per convenzione, **argv[0]** contiene il **nome del programma stesso**.

Esempio

```
// sorgente di file_exec.c
main(int argc, char ** argv[])
{ … }
```

- invoco l'eseguibile generato coi seguenti parametri:

```
-:$ ./f_ex roma 1 X
```

- quindi:

```
argc=4
argv[0]= "f_ex"
argv[1]= "roma"
argv[2]= "1"
argv[3]= "X"
```

NB: gli argomenti sono passati tutti come stringhe

Esercitazione 1 - Obiettivi

Programmazione C

- Gestione dei parametri in ingresso
  - argomenti ***argc*** ed ***argv***
  - controllo di correttezza dei parametri in ingresso
- Gestione delle stringhe
  - libreria string.h

Esercitazione 1 – Testo (1/2)

Si realizzi un programma C con un'interfaccia del tipo

`listaTreni treno1 .... trenoN`

e che prenda in ingresso un numero arbitrario di stringhe rappresentanti il codice di un treno, nel formato:

`<TIPOTRENO><NUMEROTRENO>`

- <TIPOTRENO> stringa di due caratteri che rappresenta il tipo di treno (per semplicità, si assuma che abbia i valori "IC", "ES", "RG", rispettivamente per le tipologie Intercity, Eurostar e Regionale)
- <NUMEROTRENO> identificativo numerico univoco del treno, composto da 4 cifre

Esercitazione 1 – Testo (2/2)

Il programma deve:

- controllare che sia stato passato ***almeno un treno***
- controllare che ogni codice passato sia ***conforme alle caratteristiche*** sopra indicate (in particolar modo, rispetti la lunghezza di 6 caratteri)
- stampare a video i soli identificativi dei treni, ***raggruppati per categoria***