ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Trieste

Nota tecnica interna

INFN-TS/TCN-03/01
13 Gennaio 2003

APPUNTI PER IL CORSO INTRODUTTIVO AL SISTEMA OPERATIVO UNIX

Claudio Strizzolo

Sommario

Questo documento contiene alcuni appunti utilizzati durante il corso introduttivo al sistema operativo Unix, tenuto dall'autore dal 25 novembre al 5 dicembre 2002, presso la Sezione di Trieste dell'INFN.

La presente documentazione non deve essere considerata esaustiva, e va quindi intesa principalmente come un pro-memoria per i partecipanti.

Corso introduttivo al sistema operativo Unix

Claudio Strizzolo
INFN Sezione di Trieste
25 Novembre – 5 Dicembre 2002

Introduzione

- Obbiettivi del corso:
  - introdurre i partecipanti all'utilizzo del sistema operativo Unix;
  - consentire di svolgere in modo autonomo i compiti più comuni su un sistema Unix;
  - fornire le basi per un eventuale approfondimento su temi specifici.
- La maggior parte degli argomenti trattati è valida per qualsiasi sistema Unix.
- Le tematiche trattate sono in gran parte di interesse generale. Alcuni cenni sono riferiti a configurazioni specifiche della Sezione di Trieste dell'INFN.

Alcune caratteristiche di Unix

- Sistema operativo multi-utente e multi-tasking.
- Standard "di fatto".
- Offre numerosi potenti strumenti di sviluppo.
- Disponibile su diverse architetture hardware (dai PC ai mainframe), in varie implementazioni. Ad es.:
  - Linux (Free),
  - Tru64 (Compaq),
  - SunOS, Solaris (Sun),
  - HP-UX (Hewlett-Packard),
  - ecc.

Cenni storici

- 1969: Ken Thompson sviluppa il sistema operativo Unix presso gli AT&T Bell Laboratories, prima su PDP-7, poi su PDP-11.
- 1973: il sistema viene riscritto in C; questo lo rende facilmente portabile su piattaforme diverse da quella originale.
- 1975: Unix inizia a diffondersi a livello universitario, anche grazie al fatto che i Bell Labs lo offrono a basso costo alle istituzioni accademiche.
- Negli anni seguenti Unix si diffonde anche in altri ambienti, e si arricchisce di funzionalità e nuovi strumenti di sviluppo.
- Nascono numerose varianti di Unix su diversi sistemi, ad opera di vari produttori.
- E Linux? Nasce nel 1991 ad opera di Linus Torvalds, come sistema operativo "free": il codice sorgente e i binari sono distribuiti gratuitamente.

Accesso al sistema

- L'accesso al sistema è consentito dopo l'immissione di uno **username** e della relativa **password**.
- Il sistema può presentarsi con un'interfaccia "a finestre" oppure con un prompt, come un terminale.
- Ogni utente ha una propria area di lavoro dedicata.
- Ogni utente appartiene ad uno o più **gruppi** di utenti. I gruppi consentono di condividere alcune risorse (file, dischi, ecc.) tra più utenti.
- Un sistema di **protezioni** consente di gestire l'accesso ai file da parte degli utenti.
- La directory in cui ci si trova appena eseguito il login viene chiamata **home directory**.
- Per chiudere la sessione di lavoro, si utilizza il comando **logout** o **exit**.

Un account speciale: root

- Normalmente ogni utente ha accesso ad alcune aree del sistema, ed ha poteri limitati. In questo modo l'utente può eseguire i propri compiti, senza danneggiare il sistema o files di altri utenti.
- Ogni sistema Unix ha uno username particolare, riservato all'amministratore del sistema. Lo username in questione è **root**.
- L'account root possiede i privilegi necessari ad eseguire qualsiasi operazione sul sistema.

I comandi di Unix

- Unix prevede la possibilità di usare molti comandi ed utility, eseguendoli a partire da un **prompt**.
- Di solito è disponibile anche un'interfaccia grafica "**a finestre**", che consente di effettuare molti compiti.
- **Importante: Unix è *case-sensitive*, ovvero sensibile ai caratteri maiuscoli/minuscoli. Questo vale per qualsiasi ambito: username, password, comandi, nomi dei file, ecc.**
- I comandi non si possono abbreviare.
- Per fermare un comando in esecuzione: <Ctrl-C>.
- <Ctrl-Z> di solito sospende l'esecuzione di un comando. È possibile ripristinare l'esecuzione con il comando **fg** (ved. più avanti).

Struttura dei comandi

- La struttura dei comandi è, di solito, la seguente:

```
$ comando [opzioni] [parametri]
```

- Normalmente le opzioni devono essere elencate prima dei parametri.
- Le **opzioni** sono quasi sempre precedute dal carattere "-". Ad esempio:

```
$ ls -l -R a* prova
$ tar -x -v -f archivio.tar miadir/
```

- Alcune opzioni possono essere seguite da un valore (ad es. "-f archivio.tar" nel comando precedente).
- Le opzioni si possono generalmente **raggruppare**:

```
$ ls -lR a* prova
$ tar -xvf archivio.tar miadir/
```

Documentazione in linea

- L'utility **man** consente di visualizzare la documentazione relativa ai comandi disponibili sul sistema.
- Ad esempio, per visualizzare la documentazione relativa al comando **ls**:

```
$ man ls
```

- Sono documentati non solo i comandi, ma anche molte applicazioni, librerie, compilatori, ...
- Ogni pagina del man contiene riferimenti ad altri comandi correlati.
- Per poter utilizzare man, è necessario conoscere il nome del comando che ci interessa.

Se non ricordate un comando

- L'opzione **–k** del comando **man** permette di cercare una parola chiave all'interno della documentazione in linea, restituendo la lista di tutti i comandi la cui descrizione contiene la parola chiave indicata. Ad esempio, per cercare tutti i comandi e le utility che hanno a che fare con un "editor" di testo:

```
$ man -k editor
```

- Su molti sistemi lo stesso risultato si ottiene con il comando **apropos**:

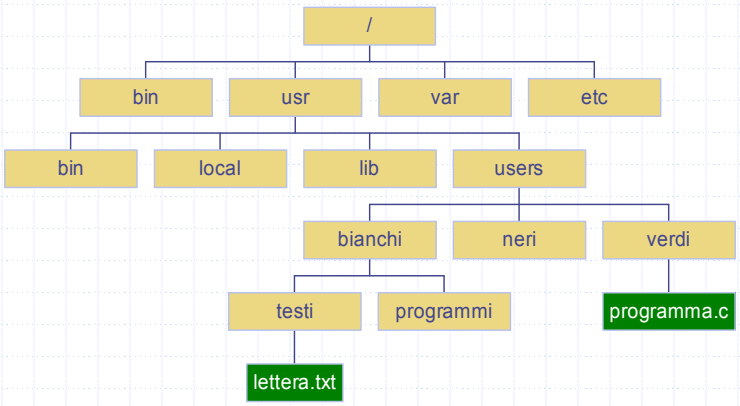
```
$ apropos editor
```

Cambiare la password

- Il comando normalmente utilizzato per cambiare la password su un sistema Unix è **passwd**.
- Sui sistemi centrali della Sezione è attivo un sistema di gestione delle autenticazioni chiamato Yellow Pages, che prevede l'uso del comando **yppasswd**.
- Anche per le password, Unix fa differenza tra caratteri maiuscoli e minuscoli.
- Sui sistemi della Sezione si consiglia di utilizzare una password lunga esattamente otto caratteri, per compatibilità con altri sistemi di autenticazione in uso (AFS).
- È attivo un sistema di controllo che impedisce l'immissione di password troppo banali, per motivi di sicurezza.

Il *File System* (1)

In un sistema Unix, i dati sono organizzati in una **struttura gerarchica** chiamata **filesystem**.

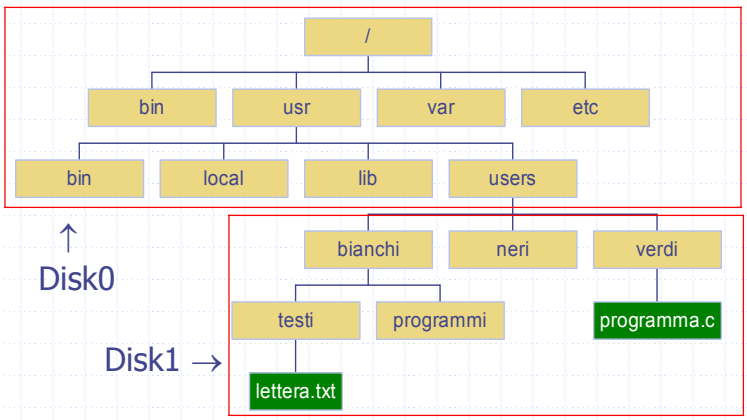


Il *File System* (2)

- Di solito su un sistema Unix esistono più filesystem, suddivisi per funzionalità.
- Ogni filesystem ha una **struttura ad albero** capovolto.
- I vari filesystem vengono "montati" in un'unica struttura: l'utente vede **un unico albero**, anche se esso è formato da diversi dispositivi fisici (dischi, cdrom, ecc.) o partizioni.
- Ogni disco (o partizione) viene "montato" in una directory chiamata **mount point**. Il montaggio dei dischi è a cura dell'amministratore del sistema.
- Un mount point può essere una directory qualsiasi: questo significa che un disco può essere montato in **qualsiasi punto** dell'albero.

Il *File System* (3)

- Dopo aver montato un disco sotto un mount point, le directory e i file in esso contenuti vengono visti a tutti gli effetti come parte di un unico albero.



Il *File System* (4)

- La **radice** dell'albero si trova in una directory indicata con il simbolo "/". Questa directory viene chiamata "**root**" (da non confondere con l'account "root"!).
- Ogni directory del filesystem può contenere diversi oggetti:
  - **File**. Possono essere raggruppati in diverse categorie:
    - Di testo (non eseguibili)
    - Eseguibili
    - File speciali (link, device, ecc.)
  - **Directory**
  - La directory speciale ".", che corrisponde alla directory **corrente**.
  - La directory speciale "..", che corrisponde alla directory "**padre**", ovvero quella che si trova immediatamente sopra a quella corrente.

Il *File System* (5)

- I dischi (o partizioni) montati su un sistema Unix non sono necessariamente locali: possono infatti essere montati anche dischi "esportati" (cioè resi disponibili in rete) da un altro computer, con sistemi di filesystem "remoti" come NFS (Network File System) o AFS (Andrew File System).

I nomi dei file

- Come già detto, i nomi dei file sono ***case-sensitive***.
- I nomi dei file possono contenere praticamente qualsiasi carattere, salvo un limitato sottoinsieme di segni che hanno un significato particolare (ad esempio "/"). Si consiglia comunque di limitarsi a lettere, cifre, underscore (_) e punti, per maggiore compatibilità con altri sistemi.
- Se si intende trasferire file da/a sistemi non Unix, si suggerisce di utilizzare al massimo un punto.
- Per quanto concerne la lunghezza dei nomi dei file, nelle versioni recenti di Unix di solito si arriva a 250 caratteri.

Caratteri jolly (1)

- In molti comandi, è possibile raggruppare insieme più file/directory utilizzando delle ***wildcard*** (caratteri "jolly"), ovvero caratteri con un significato particolare.
- *****: indica **zero o più** caratteri. Ad es. **a*** indica tutti i file il cui nome inizia per "a".
- **?**: indica **un singolo** carattere. Ad es. **a?** indica tutti i file il cui nome è composto esattamente da due caratteri, il primo dei quali è "a".
- **[...]**: indica un **qualsiasi insieme** di caratteri. Ad es. **[abc]*** indica qualsiasi file il cui nome inizia per "a", "b" o "c".
- **[a-z]**: indica tutti i caratteri compresi tra "a" e "z". Ad es. **[a-d]*** indica tutti i file il cui nome inizia per "a", "b", "c" o "d".

Caratteri jolly (2)

- Esempi:
  - a???
    - tutti i file il cui nome inizia per "a" ed è lungo esattamente 4 caratteri;
  - [a-zA-Z]*
    - tutti i file il cui nome inizia con una lettera maiuscola o minuscola;
  - *s
    - tutti i file il cui nome termina per "s";
  - [a-d][1-5].doc
    - tutti i file con estensione ".doc", il cui nome è lungo esattamente 2 caratteri, il primo dei quali è "a", "b", "c" o "d", ed il secondo una cifra compresa tra "1" e "5" (es. "a3.doc", "d5.doc")
  - page[1-3].txt
    - page1.txt, page2.txt, page3.txt

Path assoluti

- Con il termine ***path*** si indica il **percorso** necessario ad identificare un file o una directory.
- Un **path assoluto** è utilizzabile in qualsiasi punto del filesystem. Esso indica il percorso completo a partire dalla directory root ("/") fino a raggiungere il file desiderato:

```
$ ls /usr/users/verdi/testi/lettera.txt
```

  Ogni elemento del path corrisponde ad un livello nell'albero del filesystem.
- Un path assoluto particolare ("~") permette di indicare facilmente la propria home directory:

```
$ ls ~
$ ls ~/testi/prova.c
```

Path relativi

- Un **path relativo** indica un percorso relativo alla directory corrente. Ad esempio, supponendo di essere nella directory /usr/users/verdi:

```
$ ls testi/lettera.txt
$ ls ../rossi/programmi/prova.c
$ ls ../../../bianchi/testi/documento.txt
```

- **Attenzione alle barre** che si usano per separare i vari elementi del path: "/", non "\", come su altre piattaforme!

File: ls (1)

- Il comando **ls** mostra la lista dei file contenuti in una directory.
- È possibile specificare il path della directory desiderata e/o i file che si desidera elencare:

```
$ ls
$ ls ..
$ ls /usr/users/rossi
$ ls ~/miofile.txt
$ ls ../..
$ ls /usr/users/rossi/lettera*
```

- Se non viene specificata l'opzione **–a**, il comando non elenca i file/directory il cui nome comincia con un punto, ovvero i cosiddetti file "nascosti".
- L'opzione **–R** effettua una scansione anche delle subdirectory, a partire da quella corrente o quella specificata.

File: ls (2)

- L'opzione **–l** visualizza una serie di informazioni aggiuntive:
  - Tipo di file ("d"=directory, "-"=file, "l"=link, …).
  - Permessi (ved. in seguito).
  - Numero di link al file (ved. in seguito).
  - Proprietario e gruppo di appartenenza.
  - Data di creazione.

```
$ ls –l
drwxr-xr-x  2   rossi  users    4096 Jan 15 12:37  dati
-rw-rw-rw-  1   rossi  users   12375 Dec 12 15:07  lettera.txt
```

- L'opzione **–d** mostra solo il nome delle sottodirectory, anziché elencare anche il loro contenuto, come avviene per default.

File: visualizzazione di file di testo (1)

- Il comando **cat** mostra il contenuto di un file di testo:

```
$ cat lettera.txt
```

- Se si tenta di visualizzare un file non di testo, si corre il rischio di ottenere effetti indesiderati sul terminale, a causa di caratteri particolari (sequenze di escape, ecc.).
- Nel caso non si conosca a priori se un file sia realmente di testo, è bene appurarlo preventivamente con il comando **file**:

```
$ file pippo.txt
pippo.txt: ASCII text
$ file /usr/bin/zip
/usr/bin/zip: ELF 32-bit LSB executable, Intel 80386,
version 1 (SYSV), dynamically linked (uses shared libs),
stripped
```

File: visualizzazione di file di testo (2)

- Il comando **more** mostra un file di testo una pagina alla volta. Premendo la barra di spazio si visualizza la pagina successiva:

```
$ more lettera.txt
```

  Su alcuni sistemi esiste anche il comando **less**, che, a differenza di **more**, permette di spostarsi all'interno del file in entrambe le direzioni usando i tasti cursore.

- Il comando **head** mostra le prime *n* righe di un file:

```
$ head [-n] lettera.txt
```

- Il comando **tail** mostra le ultime *n* righe di un file:

```
$ tail [-n] lettera.txt
```

- Il numero di righe visualizzate per default dai comandi **head** e **tail** è 10.

File: cp

- Il comando **cp** copia un file in una diversa directory, e/o con un nome diverso:

```
$ cp file1 file2
$ cp file1 archivio/programmi/file2
$ cp /usr/users/tizio/file1 archivio/programmi/
```

- Le opzioni **–R** e **–r** consentono di copiare una directory con tutte le sue subdirectory:

```
$ cp –R archivio/programmi /miodisco/copiaprogrammi/
```

  -R e **–r** si comportano in modo diverso per quanto concerne link e altri file particolari. Consultare la documentazione online (man cp) per ulteriori dettagli.

- Altre opzioni di cp consentono ad esempio di conservare le proprietà dei file (proprietario, protezioni, ecc.), o di chiedere conferma prima di sovrascrivere eventuali file esistenti.

File: mv, rm

- Il comando **mv** sposta il file, eventualmente rinominandolo:

```
$ mv file1 archivio/programmi/
$ mv file1 file1.old
$ mv dati/abcd.dat ./
$ mv file1 archivio/programmi/file1.old
```

- Il comando **rm** cancella uno o più file:

```
$ rm file1 [file2 ...]
```

  L'opzione **–r** consente la rimozione di una directory e di tutti i file e le subdirectory in essa contenuti.
  L'opzione **-i** permette di chiedere conferma prima di eliminare un file.

- In Unix **di solito** non esiste un Trash da cui recuperare files o directory cancellati per errore.

Directory: pwd, mkdir, rmdir, cd

- **pwd** mostra il percorso della directory corrente.
- **mkdir** crea una *directory*:

```
$ mkdir appunti
```

- **rmdir** elimina una *directory* **vuota**. Se la *directory* contiene dei *file*, è necessario usare il comando **rm –r**.

```
$ rmdir appunti
```

- **cd** cambia la *directory* corrente:

```
$ cd /usr/users/rossi
$ cd testi/lettere
$ cd ..          (sale alla directory padre)
$ cd             (va alla "home directory")
$ cd ~           (va alla "home directory")
$ cd ~/programmi
```

- Per rinominare una *directory* si usa il comando **mv**, come per i *file*.

Alcune *directory* importanti

- / — radice del file system (root);
- /bin, /sbin — contengono comandi e utility di sistema;
- /usr/bin, /usr/sbin — contengono altri comandi e utility;
- /dev — contiene file speciali che consentono il controllo dei dispositivi connessi al sistema (dischi, nastri, terminali, ecc.);
- /tmp — utilizzata per la creazione di file temporanei;
- /etc — contiene file di configurazione del sistema (utenti, file system, rete, stampanti, ecc.)
- /home, /usr/users — di solito, aree disco degli utenti;
- /afs — file system di AFS.

Comandi per i file system

- Il comando **df** mostra la lista dei filesystem montati sul sistema:

```
$ df
Filesystem      1k-blocks      Used Available Use% Mounted on
/dev/hda7         4127076   2276048   1641384  59% /
/dev/hda6           31079      5820     23655  20% /boot
/dev/hda9        12831836    233496  11946508   2% /home
nas:/users       20120992   5496240  14624752  28% /usr/users
```

- Il comando **du** calcola lo spazio occupato (in blocchi o in Kbyte) da filesystem e directory:

```
$ du /var/lib
31388        /var/lib/rpm
4            /var/lib/games
4            /var/lib/misc
12           /var/lib/alternatives
1532         /var/lib/slocate
(...)
```

Alcuni comandi di uso frequente (1)

- Per maggiori dettagli sui comandi seguenti, consultare la documentazione online (man).
- **whoami**
  - visualizza lo *username* con il quale l'utente ha eseguito il login.
- **date**
  - mostra data e ora. Lo stesso comando consente anche all'amministratore di cambiare data e ora sul sistema.
- **uname**
  - mostra il sistema operativo installato sul sistema. Con l'opzione **–a** mostra anche altre informazioni, tra le quali la versione del sistema operativo.
- **who**
  - visualizza la lista degli utenti collegati al sistema.
- **hostname**
  - mostra il nome assegnato al sistema.

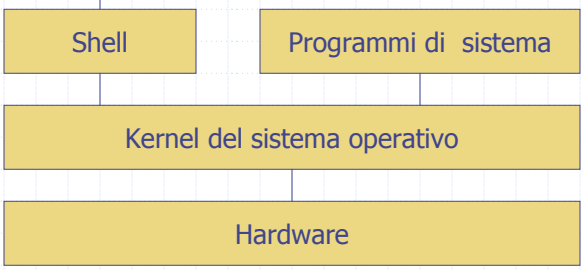
Alcuni comandi di uso frequente (2)

- **clear**
  - "clear screen".
- **diff**
  - elenca le differenze tra due *file* di testo.
- **sort**
  - ordina un *file* (senza modificarlo).
- **grep**
  - esegue la ricerca di stringhe in uno o più *file*.
- **wc**
  - conta le righe, le parole, o i caratteri presenti in un *file*.
- **cut**
  - estrae una porzione di una stringa. Es.:

```
$ cut -f1 -d " " miofile.txt
```

La shell (1)

La **shell** costituisce **l'interfaccia tra l'utente ed il sistema operativo**: interpreta il comando immesso dall'utente ed esegue il corrispondente programma di sistema.



La shell (2)

- Esistono varie shell con caratteristiche diverse, ad es.:
  - Bourne shell (sh), particolarmente adatta per scrivere le cosiddette "script" (programmi contenenti comandi di sistema).
  - C shell (csh), adatta per uso interattivo.
  - tcsh, derivata da csh.
  - bash, derivata da Bourne ma con diverse caratteristiche utili per l'uso interattivo (di solito è il default su Linux).
- La **shell di login** viene stabilita dall'amministratore, ma può essere cambiata dall'utente, di solito con il comando **chsh**.
- Presso la Sezione di Trieste, normalmente l'amministratore assegna inizialmente la **tcsh** o la **bash**, che si prestano bene ad un uso interattivo.
- La scelta di una shell dipende dall'uso che si intende farne (interattivo/non interattivo) e dalle abitudini dell'utente.

Script di shell

- Praticamente tutte le shell permettono sia un uso interattivo che la stesura di **script**, ovvero **file di comandi** di shell (concettualmente simili ai file batch del DOS, ma con potenzialità di gran lunga superiori).
- Alcune shell si prestano meglio all'uso interattivo, altre alla realizzazione di script. Una script può essere scritta anche con una shell diversa da quella di login.
- Le script sono veri e propri programmi, nei quali si possono utilizzare anche delle funzionalità che normalmente non si usano in modo interattivo: definizioni di variabili, strutture di controllo (loop, condizioni), ecc. In alcuni casi queste potenzialità non hanno nulla da invidiare a quelle di un vero linguaggio di programmazione.
- La sintassi delle script di shell dipende dalla shell utilizzata.
- La programmazione delle script di shell esula da questo corso di base.

Script di shell eseguite al login (1)

- Alcune script di shell hanno un significato particolare.
- Ogni shell cerca ed esegue, se disponibili, alcune **script particolari** quando viene avviato un nuovo processo, per configurare l'ambiente di lavoro (ad esempio eseguendo dei comandi particolari o definendo delle variabili "di ambiente").
- I nomi delle script di shell eseguite automaticamente allo startup dei processi **dipendono dalla shell**.
- Alcune script vengono create dall'amministratore di sistema, e sono eseguite al login di ogni utente: di solito si trovano nella directory /etc, e contengono i settaggi validi per tutti gli utenti.
- Ogni utente ha la facoltà di creare altre script nella propria home directory, che verranno automaticamente eseguite dopo le script di login create dall'amministratore. I nomi di tali script dipendono dalla shell utilizzata dall'utente (ved. scheda seguente).

Script di shell eseguite al login (2)

- Bourne shell e simili eseguono la script **~/.profile.**
- csh, tcsh e simili eseguono due script:
  - **~/.login**: viene eseguito una sola volta, al login;
  - **~/.cshrc**: viene eseguito alla partenza di ogni singolo processo avviato dall'utente.
- bash esegue due script:
  - **~/.bashrc**: viene eseguito ogni volta che un utente inizia un processo;
  - **~/.bash_profile**: viene eseguito una sola volta, al login. Questa script di solito richiama ~/.bashrc, in modo che anche quest'ultima venga eseguita ad ogni login.

Un esempio di shell: tcsh (1)

- tcsh è una shell derivata dalla csh.
- Presenta numerose funzionalità utili soprattutto per **l'uso interattivo**. Alcune di esse sono disponibili, con caratteristiche simili, anche in altre shell, come la bash.
- Alcune funzioni interessanti:
  - **history**: utilizzando i tasti cursore è possibile scorrere la lista dei comandi precedentemente digitati.
  - **alias**: si possono definire degli alias per comandi di uso frequente, o con sintassi complesse, con il comando **alias**:

```
$ alias dir ls –la
```

    Dopo questa definizione, digitare il comando **dir** sarà equivalente a **ls –la**.

    Se si usa il comando alias senza parametri, viene mostrata la lista degli alias definiti.

Un esempio di shell: tcsh (2)

- (continua: Alcune funzioni interessanti)
  - **completamento dei nomi dei file**: nell'immissione di un comando, si può digitare solo la prima parte del nome di un file o di una directory, e poi premere il tasto <Tab>: la shell provvederà a completare automaticamente il nome del file, a meno che non ci sia qualche ambiguità.
  - **raggruppamento di comandi**: i comandi si possono immettere su righe separate, oppure su un'unica linea, separandoli con ";":

```
$ cd ~
$ ls

$ cd ~; ls
```


tcsh: variabili di shell

- tcsh consente di definire delle variabili (comando **set**).
- Questa caratteristica è particolarmente utile nella realizzazione di script di shell, ma è possibile definire le variabili anche per un uso interattivo (nota: il comando **echo** visualizza una costante o il valore assegnato ad una variabile):

```
$ set mionome = Mario
$ echo $mionome
Mario
$ set nomecognome = "Mario Rossi"
$ echo "Mi chiamo $nomecognome"
Mi chiamo Mario Rossi
```

tcsh: variabili di ambiente (1)

- Alcune variabili sono predefinite dalla shell, e vengono utilizzate per controllare alcuni valori particolari. Alcuni esempi:
  - HOME il percorso della home directory dell'utente cui appartiene il processo;
  - SHELL il percorso del comando corrispondente alla shell;
  - PATH la lista dei percorsi nei quali la shell cerca i comandi e i programmi da eseguire, qualora non ne venga esplicitamente specificato il percorso;
  - PWD il percorso della directory corrente (ved. comando **pwd**);
  - OSTYPE il sistema operativo in uso;
  - ecc.
- Molte di queste variabili vengono normalmente definite all'interno delle script di shell utilizzate per configurare l'ambiente di lavoro prima dell'esecuzione di un processo (per la tcsh: .login e .cshrc).

tcsh: variabili di ambiente (2)

- Per definire/ridefinire una variabile di ambiente si usa il comando **setenv**:

```
$ setenv PATH "/bin:/sbin:/usr/sbin:/usr/local/bin"
```

- Il comando **printenv** visualizza il valore assegnato ad una variabile di ambiente (o a tutte, se non ne viene specificata una in particolare):

```
$ printenv
HOME=/usr/users/rossi
SHELL=/bin/tcsh
PWD=/usr/users/rossi/programmi
OSTYPE=linux
PATH=/bin:/sbin:/usr/sbin:/usr/local/bin
(...)
```

tcsh: il file .login

- Il file .login viene eseguito automaticamente quando l'utente inizia una sessione di lavoro.
- Di solito viene utilizzato per definire/ridefinire alcune variabili di ambiente valide per l'intera sessione.
- Ad esempio:

```
$ cat .login
#!/bin/csh
setenv PATH .:/bin:/sbin:/usr/bin:/usr/local/bin
setenv TERM vt100
echo "Buongiorno!"
```


tcsh: il file .cshrc

- Il file .cshrc viene eseguito automaticamente ogni volta che viene avviato un nuovo processo che utilizza la csh o la tcsh.
- Questo file viene utilizzato per definire variabili e parametri specifici per la shell in uso.
- Ad esempio:

```
$ cat .cshrc
#!/bin/csh
alias dir ls -la
alias del rm -i
set prompt = "MarioRossi> "
set history = 100
```

Redirezioni (stdin, stdout, stderr) (1)

- Normalmente, per i processi interattivi, la shell considera come **standard input** e **standard output** il terminale dell'utente, ovvero rispettivamente la tastiera e il video.
- Esiste inoltre un canale per lo **standard error**, che di solito coincide con il video.
- L'utente può ridefinire l'input e/o l'output per un comando, in modo che esso legga i dati da un file, oppure scriva l'output su un file.
- Per inviare l'output di un comando su un file, si usa il simbolo ">":

```
$ ls -l > lista_files.txt
```


Redirezioni (stdin, stdout, stderr) (2)

- Un modo semplice per creare un file di testo:

```
$ cat > lista.txt
arancia
mela
pera
albicocca
^D
```

- Questo comando esegue un cat dello stdin (tastiera), redirigendo l'output verso un file.
- Il Ctrl-D chiude il file.

Redirezioni (stdin, stdout, stderr) (3)

- La ridirezione dell'output può essere usata anche per concatenare più file di testo:

```
$ cat uno.txt due.txt > uno_e_due.txt
```

- Il simbolo ">>" consente di reindirizzare l'output su un file **accodandolo**, invece di sovrascriverlo:

```
$ cat uno.txt due.txt > numeri.txt
$ cat tre.txt >> numeri.txt
```

- Per reindirizzare lo standard error di solito si usano i caratteri ">&", ma questo dipende dalla shell. Ad esempio:

```
$ mioprogramma > out.log >& error.log
```

- Per ridefinire l'input di un comando si usa il simbolo "<":

```
$ sort < miofile.txt
$ mailx tizio@dominio.it < testo.txt
$ mioprogramma < mieidati.dat > risultati.dat
```


Pipes (1)

- Le shell consentono di utilizzare un sistema di ***pipes*** per ridirigere l'output standard di un comando (vale a dire l'output che normalmente viene mostrato sul terminale) come input di un altro comando.
- La catena di *pipes* può essere allungata per quanti comandi si desidera.
- Il *pipe* viene realizzato utilizzando il carattere "|" (barra verticale) tra un comando e l'altro:

Comando a | Comando b | Comando c .....

- In questo modo l'output del comando "a" non viene mostrato a video, ma passato come input al comando "b", il cui output a sua volta viene passato come input al comando "c", e così via.

Pipes (2)

- Alcuni esempi:

```
$ ls -l *.doc | more
$ ls -l | grep programma | sort | more
$ who | sort
$ who | sort | lp
$ cat file.txt | mail -s "Il mio testo" tizio@dominio.org
```


Permessi dei file (1)

- L'**accesso** ai file e alle directory è regolato da un sistema di **permessi**. Essi controllano la possibilità di eseguire tre operazioni:
  - Lettura ("r").
  - Scrittura ("w").
  - Esecuzione ("x"); per le *directory* significa visualizzazione del contenuto e possibilità di accedere alle subdirectory.
- Ogni operazione può essere consentita/negata a tre classi di utenti:
  - Il **proprietario** del file ("u" = user).
  - I componenti dello stesso **gruppo** cui appartiene il file ("g" = group).
  - Qualsiasi altro **utente** ("o" = others).

Permessi dei file (2)

- Il comando **ls –l** mostra i permessi di *file* e *directory*:

```
$ ls -l
drwxr-x---  2 rossi users  512   Dec 12 12:31 dati
-rwxr--r--  1 rossi users  7899  Mar 11 15:21 foto.jpg
-rwxr-x---  1 rossi users  1927  Jul 15 08:01 prog.c
drw-------  2 rossi users  512   Apr 27 09:02 testi
```

- I permessi si leggono a **gruppi di tre caratteri**, a partire dal secondo carattere di ogni riga (il primo identifica il tipo di file). Il primo gruppo di tre caratteri identifica i permessi per il proprietario del *file*, il secondo per il gruppo, il terzo per gli altri utenti. Le lettere identificano i permessi accordati, I trattini sono al posto di quelli non accordati, nell'ordine **rwx**.

Permessi dei file (3)

- Il comando **chmod** consente di cambiare i permessi di *file* e *directory*, ovviamente se si hanno i privilegi necessari per farlo. La sintassi è:

```
$ chmod [-R] permessi filename .....
```

L'opzione **–R** agisce in modo ricorsivo anche nelle *subdirectory*.

- I permessi possono essere indicati in due modi:
  - Una **lista** di permessi, separati da virgole.
  - Un **numero ottale** di tre cifre.

Permessi dei file (4)

- Nel caso della **lista di permessi**, ogni permesso è composto da tre parti:
  - Categoria di utenti: **ugoa** (user, group, other oppure all).
  - Operatore: **-+=** (togli, aggiungi o assegna questi permessi).
  - Operazioni: **rwx** (read, write o execute).
- Ad esempio, per assegnare ad un file i permessi rwx al proprietario, ed rx al gruppo:

```
$ chmod u=rwx,g=rx miofile.txt
```

Per togliere a chiunque il permesso di scrivere su un file:

```
$ chmod a-w miofile.txt
```


Permessi dei file (5)

- Nel caso del numero ottale di tre cifre, ogni cifra raggruppa i permessi per una categoria di utenti, nell'ordine user, group e others. La tabella riassume le definizioni dei permessi in ottale:

Permessi	Cifra ottale
---	0
--x	1
-w-	2
-wx	3
r--	4
r-x	5
rw-	6
rwx	7

Es.: `chmod 744 testo.txt`

Permessi dei file (6)

- Capita talvolta di dimenticarsi di rendere eseguibile un programma che avete scritto/installato. Se, tentando di eseguire un programma, succede questo:

```
$ mioprogramma
mioprogramma: Permission denied
```

  probabilmente la cosa si risolverà attivando il flag x. Ad esempio:

```
$ chmod u+x mioprogramma
$ chmod 700 mioprogramma
$ chmod 755 mioprogramma
```

- Su Unix, i programmi, o meglio, i file eseguibili, non sono identificati con un'estensione particolare (".exe", ".com" o simili): quello che conta è il permesso di esecuzione.

Permessi dei file (7)

- Ai file e alle directory creati dagli utenti vengono assegnati dei permessi "di default", stabiliti dall'amministratore di sistema. L'utente ha la facoltà di vedere/cambiare i propri permessi di default utilizzando il comando **umask**. Questo comando viene di solito inserito nelle script eseguite automaticamente al login, e prevede un modo particolare di specificare i permessi, per mezzo di una "maschera". Per maggiori informazioni consultare la documentazione online.
- Il comando **cp** normalmente assegna ai file di destinazione i permessi "di default" dell'utente che esegue il comando.
- È possibile fare in modo che al file di destinazione siano assegnati, ove consentito, i permessi del file di partenza, specificando l'opzione **–p** del comando **cp**.
- Lo stesso vale anche per la copia di directory:

```
$ cp -Rp sorgente destinazione
```

Gruppi di utenti e permessi dei file

- Un utente può appartenere ad uno o più gruppi.
- L'appartenenza ad un gruppo consente la condivisione di risorse (file, ma non solo) tra gli utenti appartenenti allo stesso gruppo, agendo in modo opportuno sui permessi assegnati ai file.
- Il comando **groups** mostra la lista dei gruppi cui l'utente appartiene.
- Il comando **chgrp** consente di assegnare un gruppo diverso ad uno o più file. Ad esempio:

```
$ groups
users alpha beta
$ ls -l pippo.txt
-rw-------   1 rossi  users  127 Oct 17 15:01 pippo.txt
$ chgrp alpha pippo.txt
$ ls -l pippo.txt
-rw-------   1 rossi  alpha  127 Oct 17 15:01 pippo.txt
```


Ricerca di file (1)

- Il comando **find** permette di cercare file o directory a partire da un certo path, in tutte le sue subdirectory.
- Sono disponibili svariati criteri di ricerca: si consiglia di consultare il **man find** per maggiori dettagli.
- Alcuni esempi:

```
$ find . -name prova.txt -print
```

(cerca il file prova.txt a partire dalla directory corrente. Se lo trova ne stampa il percorso)

```
$ find /usr/users/verdi -type d -print
```

(cerca tutte le directory che si trovano al di sotto di /usr/users/verdi)

- Tramite opportune opzioni è possibile eseguire anche comandi complessi su ogni file/directory "trovato" dal **find**.

Ricerca di file (2)

- Alcuni esempi di comandi complessi:

```
$ find . -mtime 1 -print
```

(cerca i file modificati nelle ultime 24 ore).

```
$ find . -name "*.log" -atime +7 -exec rm {} \;
```

(trova e rimuove i file il cui nome termina con l'estensione .log non acceduti nell'ultima settimana).

```
$ find . -name "*" -exec grep pippo {} /dev/null \;
```

(trova, in tutti i file, le righe che contengono la stringa "pippo". Visualizza tali righe precedute dal nome del file che le contiene).

Ricerca di file particolari su Linux

- Su Linux esistono alcuni comandi per ricercare file particolari
- **locate**: esegue una ricerca per sottostringa in un database di file noti (di sistema) che viene aggiornato periodicamente in modo automatico. Di solito si usa per cercare comandi, file di configurazione o altri file di sistema:

```
$ locate ghostview
```

- **which**: mostra il percorso completo di un comando, se presente in una delle directory specificate nella variabile di ambiente PATH:

```
$ which head
/usr/bin/head
```

Mostra "quale" programma chiamato **head** viene eseguito quando immettiamo il comando **head**.

Stampa (1)

Nota: I comandi esposti in questa parte sono strettamente legati alla configurazione dei sistemi presso l'INFN di Trieste.

- La stampa di un *file* viene eseguita con il comando **lp**.
- Il comando prevede varie opzioni, ad esempio:
  - **-d *coda*** : seleziona la coda sulla quale stampare il file.
  - **-n *numero*** : stampa più copie dello stesso file.

```
$ lp –d phaser lettera.txt
```

- Consultare il man per ulteriori informazioni.

Stampa (2)

- Il comando **lpstat –a** permette di visualizzare la lista delle code di stampa disponibili sul sistema.
- Presso l'INFN di Trieste sono attualmente definite quattro code per ogni stampante. Le code permettono di stampare con modalità diverse:
  - *Nomecoda*: stampa su faccia singola.
  - *Nomecoda*/K2: stampa in fronte/retro.
  - *Nomecoda*/N2: stampa due pagine per foglio, con orientamento "landscape".
  - *Nomecoda*/K2N2: applica entrambe le opzioni precedenti.
  - Es.: `$ lp –d phaser/K2 lettera.txt`

I link (1)

- Un **link** è un puntatore ad un *file*. Ogni *file* ha almeno un puntatore, che viene creato al momento della creazione del *file* stesso.
- Aggiungendo altri *link* ad un file è possibile raggiungere lo stesso da diversi punti del filesystem, ad esempio da directory diverse.
- Ci sono due tipi di *link*: **hard link** e **link simbolici**.
- I **link simbolici** permettono di aggirare alcuni limiti degli hard link, ed in particolare:
  - Consentono agli utenti di creare un link ad una directory, senza privilegi particolari.
  - Consentono di creare link tra filesystem diversi.

I link (2)

- Per creare un link si usa il comando **ln**.
- Esempio di **hard link**:

```
$ ln dati/file_originale.txt mio_link.txt
```

  Ogni modifica a *mio_link.txt* si riflette anche su *file_originale.txt*, e viceversa. Se cancelliamo *file_originale.txt*, il documento rimane accessibile come *mio_link.txt*.
- Esempio di **link simbolico**:

```
$ ln -s dati/file_originale.txt mio_link.txt
```

  mio_link.txt non è una copia di *file_originale.txt*, ma un **puntatore** ad esso. Se cancelliamo *file_originale.txt*, il link *mio_link.txt* punta ad un oggetto che non esiste più: di conseguenza il documento non è più accessibile.

I link (3)

Un esempio di uso dei link:

```
$ ls -l file.txt
-rw-------  1 verdi users  20 May  3 09:14 file.txt
$ ln file.txt hardlink
$ ln -s file.txt symlink
$ ls -l file.txt hardlink symlink
-rw-------  2 verdi users  20 May  3 09:14 file.txt
-rw-------  2 verdi users  20 May  3 09:14 hardlink
lrwxrwxrwx  1 verdi users   8 May  3 09:15 symlink -> file.txt
```


Compressione di file (1)

- La compressione consente di ridurre lo spazio occupato da un file su disco, senza alcuna perdita di dati.
- Ci sono diversi software di compressione/decompressione, che lavorano con diversi algoritmi.
- Alcuni comandi di compressione/decompressione:
  - **compress/uncompress** (*file* in formato .Z, standard su Unix):

    ```
    $ compress file1 file2 … filen
    $ uncompress file1.Z file2.Z … filen.Z
    ```

  - **gzip/gunzip** (*file* in formato .gz; non standard su Unix, ma largamente diffuso e molto efficiente)
  - zip/unzip (*file* in formato .zip)
  - pack/unpack (*file* in formato .z)

Compressione di file (2)

- Ogni *file* compresso viene rimpiazzato da uno con lo stesso nome, più un'estensione che dipende dal formato di compressione (es. "pippo.txt" → "pippo.txt.Z").
- I file compressi con alcuni comandi (ad es. **compress** e **gzip**) possono essere espansi anche con vari programmi molto diffusi su piattaforme diverse dallo Unix.

Archivi tar (1)

- Il comando **tar** serve per archiviare gruppi di *file* o *directory*.
- L'archivio ottenuto può essere salvato su nastro o in un *file* su disco.
- Il comando **tar** ha molte opzioni. Le principali:
  - **x**: estrae i *file* contenuti in un archivio;
  - **c**: crea un archivio;
  - **r**: appende uno o più *file* alla fine di un archivio;
  - **v**: "verbose mode";
  - **t**: elenca i file contenuti in un archivio;
  - **f** *nomefile*: specifica che l'archivio deve essere salvato nel file *filename* (o che i file devono essere estratti dall'archivio denominato *filename*);
  - se non viene specificata l'opzione"**f**", viene utilizzato un dispositivo di default, che di solito coincide con un'unità nastro.

Archivi tar (2)

- Alcuni esempi:
  - `$ tar -cf mieifiles.tar /usr/users/rossi`

    (crea un'archivio dei file e delle directory a partire da /usr/users/rossi)
  - `$ tar -tvf mieifiles.tar`

    (elenca i file contenuti nell'archivio denominato mieifiles.tar)
  - `$ tar -xvf mieifiles.tar`

    (estrae I file contenuti nell'archivio mieifiles.tar)
  - `$ tar -cv /usr/users/rossi`

    (crea un'archivio sull'unità nastro di default)

Archivi tar (3)

- Su molti sistemi Unix il comando **tar** non consente alcuna forma di compressione. Nulla ci vieta però di fare questo:

```
$ tar -cf mieifiles.tar /usr/users/rossi
$ gzip mieifiles.tar      (compress mieifiles.tar)
```

- Su alcuni Unix (ad es. Linux), il comando **tar** prevede alcune opzioni per la compressione/decompressione del file tar:
  - "z": comprime/espande con il comando "gzip". Crea/espande un file con estensione .tar.gz.
  - "Z": comprime/espande con il comando "compress". Crea/espande un file con estensione .tar.Z.
- Ad esempio:

```
$ tar -zcvf archivio.tar.gz /usr/users/pippo/mieifiles
```

Cenni sul networking (1)

- I sistemi Unix supportano i protocolli di rete **TCP/IP**.
- TCP/IP è implementato su quasi tutti i sistemi operativi. Questo rende possibile la comunicazione tra i sistemi Unix e nodi di tipo diverso.
- Ad ogni computer collegato alla rete è assegnato un **indirizzo numerico** univoco, formato da quattro byte (es. 140.105.6.100).
- Esiste una struttura di **Name Server** che consente di assegnare ad ogni nodo un indirizzo letterale. Essa permette inoltre di organizzare la rete in modo gerarchico con una struttura "**a domini**".
- L'indirizzo letterale è formato da due parti, nome della macchina ("quark") e nome del dominio ("ts.infn.it"): quark.ts.infn.it.
- Il database distribuito che fornisce le traduzioni tra nomi e indirizzi numerici si chiama **DNS** (Domain Name Service).

Cenni sul networking (2)

- Il comando **nslookup** (o **dig**, su Linux) consente di interrogare il DNS per ottenere le traduzioni nome ➠ numero o viceversa.
- Tramite i protocolli TCP/IP sono disponibili su Internet diversi servizi, ad esempio:
  - posta elettronica;
  - trasferimento di file (ftp, sftp);
  - login remoto (telnet, ssh);
  - filesystem distribuito (ad es. tramite NFS);
  - WWW.

Login remoto: telnet

- Il comando **telnet** permette di collegarsi ad un nodo remoto.
- All'apertura della connessione, vengono richiesti lo username e la password per accedere al sistema remoto.
- Una volta aperta la connessione, i comandi digitati vengono eseguiti sul nodo remoto, mentre l'output dei comandi compare sul terminale locale:

```
$ telnet quark.ts.infn.it
Connected to quark.ts.infn.it
Escape character is '^]'.
  (…)
Login: rossi
Password:
  (…)
$ hostname
quark.ts.infn.it
$
```


Trasferimento di file con ftp (1)

- Per trasferire file tra macchine remote si usa il comando **ftp**.
- All'apertura della connessione, vengono richiesti lo username e la password per accedere al sistema remoto:

```
$ ftp quark.ts.infn.it
Connected to quark
220 quark.ts.infn.it FTP server (…) ready
Name (quark:rossi): bianchi
331 Password required for bianchi.
Password:
230 User bianchi logged in.
ftp>
```

- Al prompt di ftp si possono usare diversi comandi per controllare il trasferimento (ved. scheda successiva).

Trasferimento di file con ftp (2)

- Alcuni comandi di ftp:
  - **ls, pwd, cd**: equivalenti ai corrispondenti comandi di Unix.
  - **bin, ascii**: selezione del formato dei file da trasferire.
  - **get**: preleva un singolo file dalla macchina remota (ad es. get pippo.txt).
  - **put**: scarica un singolo file sulla macchina remota (ad es. put pippo.txt).
  - **mget, mput**: come get e put ma per gruppi di file (ad es. mget *.txt)
  - **quit**: esce da ftp

Login remoto sicuro: ssh

- Il comando **ssh** funziona in modo simile al **telnet**, ma stabilisce una connessione "sicura": la tramissione viene crittata e le informazioni trasmesse, ad esempio ciò che viene digitato sulla tastiera, non sono leggibili lungo il percorso.
- **Si consiglia di usare sempre ssh al posto di telnet, se possibile!**
- Esempio:

```
$ ssh -l rossi quark.ts.infn.it
rossi@quark's password:
   (…)
$ hostname
quark.ts.infn.it
$
```

Trasferimento sicuro di file con scp

- Il comando **scp** permette di copiare file tra macchine remote in maniera "sicura". Analogamente a ssh, i dati vengono crittati e non sono quindi leggibili lungo il percorso.
- Esempio:

```
$ scp rossi@quark:/home/rossi/pippo.txt ./
rossi@quark's password:
pippo.txt       100% |*************|   1321    00:00
$
```

- **Utilizzate preferibilmente scp al posto di ftp, se possibile!**
- Esiste anche un ftp sicuro chiamato **sftp**, che funziona in maniera molto simile a quella di ftp.

Posta elettronica

- Sui sistemi Unix si possono trovare diversi *client* per accedere alla posta elettronica: **mail, mailx, pine,** ecc.
- Per la Sezione di Trieste, ricordate che il server di posta è su una macchina indipendente, e che la posta può essere acceduta solo tramite i protocolli **POP** e **IMAP**, da qualsiasi computer della Sezione (consultare la documentazione relativa).
- Per un uso occasionale anche programmi come **mail** o **mailx** possono andare bene.
- Per un uso più frequente, si consiglia di utilizzare **pine**, oppure qualche mail agent come Netscape. Ricordatevi di configurare il vostro programma preferito per poter accedere al server di posta!
- L'interfaccia utente di **pine** è a menu. In ogni videata è attivo un help.

CD e floppy disk (1)

- Le modalità per accedere ad un dispositivo (CD o floppy disk) dipendono dalla versione di Unix. Gli esempi di seguito si riferiscono a Linux. Vedremo solo alcuni cenni, perché la materia è piuttosto complessa.
- Per poter accedere al contenuto di un CD o di un floppy disk, bisogna prima **montare** il dispositivo in qualche punto del filesystem, come avviene per qualsiasi altro disco.
- Di solito (ma non necessariamente!) ciò è consentito all'utente che ha aperto una sessione direttamente sulla macchina.
- Il comando **mount** viene utilizzato per montare il disco nel filesystem:

```
$ mount /mnt/floppy
$ mount /mnt/cdrom
```

- Dopo il **mount**, il contenuto del dispositivo è accessibile sotto /mnt/cdrom o /mnt/floppy.

CD e floppy disk (2)

- La sintassi vista per il comando **mount** è un caso particolare: il comando mount ha di solito una sintassi molto più complessa.
- Normalmente solo l'amministratore di sistema può montare un dispositivo in un determinato punto del filesystem.
- Il comando mount per il CD consente di montare dischi in formato **ISO9660**.
- Dopo aver utilizzato il dispositivo, bisogna **smontarlo**, ovvero scollegarlo dal filesystem, **prima** di estrarre il CDROM o il floppy:

```
$ umount /mnt/floppy
$ umount /mnt/cdrom
```

Gli "mtools" (1)

- Gli **mtools** sono un gruppo di utilities di public domain, che si trovano spesso sui sistemi Unix.
- Permettono di accedere con facilità ai **dischi DOS** (inclusi i floppy disk) da una macchina Unix.
- Hanno una sintassi che richiama quella dei normali comandi DOS (copy, dir, type, del, ecc.). I comandi di solito hanno gli stessi nomi degli equivalenti DOS, preceduti dalla lettera "m": **mcopy, mdir, mdel**, ecc.
- Ai vari dispositivi vengono assegnate delle lettere "alla DOS". Ad esempio, di solito il floppy coincide con A:.
- Alcuni esempi:
  - `mcopy pippo.txt a:`
  - `mcopy a:pippo.txt pluto.txt`
  - `mcopy *.gif a:`

Gli "mtools" (2)

- In alcuni casi è necessario includere i nomi dei file o delle directory tra virgolette, specialmente quando si usano le wildcard (ad es. "*"), se non si desidera che queste ultime vengano interpretate dalla shell:
  - `mcopy "a:*.gif" immagini/`

  Consultare il man degli mtools per ulteriori dettagli.
- Altri esempi:
  - `mdir a:`
  - `mtype a:pippo.txt`
  - `mdel a:pippo.txt`
  - `mdel "a:*.txt"`
  - `mren a:pippo.txt pluto.txt`
  - `mformat a:`
- Ogni comando ha una serie di opzioni (Unix-like). Consultare il man dei singoli comandi per ulteriori dettagli.

Processi (1)

- Ogni comando in esecuzione su un sistema Unix viene chiamato **processo**.
- La shell che parte quando si esegue il login, i comandi Unix digitati dall'utente, qualsiasi programma eseguito: sono tutti esempi di processi.
- Unix è un sistema operativo **multitasking**, ed è quindi in grado di eseguire più processi contemporaneamente.
- Ogni utente può avere più processi in esecuzione nello stesso momento.
- Ogni processo ha un numero identificativo univoco che lo identifica: il *Process Identifier* (**PID**).

Processi (2)

- I processi possono essere eseguiti in due modalità:
  - **foreground**, con controllo interattivo dalla console (video e tastiera);
  - **background**, non controllabile in modo interattivo.
- Normalmente i comandi digitati da tastiera vengono eseguiti in foreground. Per eseguirli in background è necessario specificare il carattere "&" alla fine della linea:

```
$ miocomando parametri
$ miocomando parametri &
```

- I comandi **fg** e **bg** permettono di far passare in foreground un comando che era in background, o viceversa (ved. il man).

Processi (3)

- Ogni processo utilizza almeno tre *file* aperti:
  - **stdin**: *standard input*, ovvero il *file* da cui provengono i dati in ingresso al processo. Per i processi interattivi di solito corrisponde con la tastiera;
  - **stdout**: *standard output*, ovvero il *file* su cui viene scritto l'output. Per i processi interattivi di solito è il video;
  - **stderr**: *standard error*, ovvero il *file* su cui vengono scritti eventuali messaggi di errore. Per i processi interattivi di solito coincide con il video.

Processi (4)

- Il comando **ps** permette di controllare lo stato dei processi:

```
$ ps
  PID TTY          TIME CMD
23904 pts/3    00:00:00 tcsh
26125 pts/3    00:00:00 ps
```

- Per vedere anche i processi degli altri utenti:

```
$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
(...)
strizzol   23931   23911 0 08:08 pts/4    00:00:00 pine
strizzol   24259   23895 0 08:21 pts/2    00:00:00 ssh castor
root       25587       1 0 09:15 ?        00:00:00 /usr/sbin/sshd
(...)
```

- Consultare il man per altre opzioni.

Processi (5)

- Il comando **kill** permette di terminare l'esecuzione di un proprio processo, indicandone il PID:

```
$ kill 23931
```

- L'utility **top**, non standard ma disponibile su molti sistemi, permette di osservare in modo dinamico i processi attivi.

Daemons ("demoni")

- Sono **processi** che girano in **background**.
- Di solito gestiscono **servizi di sistema**, ad es.:
  - stampa (lpd)
  - esecuzione di comandi periodici (cron)
  - servizi di rete (inetd)
- Generalmente partono al boot del sistema, e sono attivi fino allo shutdown, oppure partono quando viene richiesto il relativo servizio e terminano appena esaurito il loro compito.
- Il primo processo che parte su un sistema Unix è un daemon: **init** (che ha sempre PID=1), il **padre** di tutti gli altri processi. Parte al boot ed esegue quanto specificato nel file /etc/inittab.

L'editor vi (1)

- Sui sistemi Unix si possono trovare diversi editor: **vi**, **emacs**, **ed**, ecc.
- **vi** è l'editor più diffuso sui sistemi Unix.
- Poco intuitivo ma molto potente.
- L'interfaccia utente è **full-screen** ma piuttosto spartana.
- Per editare un file esistente, o per creare un nuovo file con un certo nome:

  `$ vi` *nomefile* (es. vi pippo.txt)
- Esiste anche la possibilità di aprire un file esistente in sola lettura (per evitare di danneggiarlo inavvertitamente), utilizzando il comando:

  `$ view` *nomefile*
- Su Linux, il comando **vimtutor** fa partire un corso interattivo su vi. Semplice e interessante!

L'editor vi (2)

- **vi** ha diverse modalità di funzionamento. Le principali sono:
  - **input mode**: è la modalità normale per la digitazione del testo;
  - **command mode**: in questa modalità i tasti diventano comandi che agiscono sul testo (ad es. "a"=append, "x"=cancella un carattere, ecc.). Il tasto "**Esc**" permette di passare da *input mode* a *command mode*. Il tasto "**i**" (insert) consente di ritornare in input mode;
  - **last line mode**: il cursore si sposta sull'ultima linea dello schermo, dopo di che è possibile immettere dei comandi anche complessi (ad es. per effettuare sostituzioni di stringhe). Il tasto "**:**" permette di passare da *command mode* a *last line mode*.
- **Tutti** i comandi che vedremo si usano in *command mode*. Quelli che iniziano per ":" forzano il passaggio al *last line mode*.
- Vedremo solo **una minima parte** dei molti comandi di **vi**.

L'editor vi (3)

- Inserimento e cancellazione di testo:
  - **i** inizia inserimento prima del carattere corrente (insert);
  - **a** inizia inserimento dopo il carattere corrente (append);
  - **x** cancella un carattere;
  - **dd** cancella la riga corrente;
  - *n* **dd** cancella *n* righe;
  - **dw** cancella la parola corrente;
  - *n* **dw** cancella *n* parole;
  - **dG** cancella fino alla fine del file;
  - **J** unisce la riga corrente con la successiva (Join).

L'editor vi (4)

- Salvataggio e uscita da **vi**:
  - **:q!** esce da **vi** senza salvare i dati (quit!);
  - **:w** salva i dati senza uscire da **vi** (write);
  - **:w** *nomefile*
    salva il file con un nome diverso ("Save as..");
  - **:wq** salva i dati ed esce da **vi** (write and quit);
- Ripetizione di comandi:
  - **.** riesegue l'ultimo comando eseguito;
- Annullamento di comandi:
  - **u** annulla l'ultimo comando eseguito (undo).

L'editor vi (5)

- Come spostarsi all'interno del testo:
  - in *input mode* di solito si possono usare i tasti cursore;
  - **G** va alla fine del file;
  - **1 G** va all'inizio del file;
  - *n* **G** va alla *n*-esima riga del file;
  - **<ctrl> f** va avanti di una pagina (Forward);
  - **<ctrl> b** va indietro di una pagina (Backward);
  - **w** va avanti di una parola;
  - **b** va indietro di una parola;
  - **$** va alla fine della riga;
  - **^** va all'inizio della riga.

L'editor vi (6)

- Copia e spostamento di parti di testo:
  - **vi** utilizza un buffer di lavoro nel quale si possono copiare parti del testo, che possono essere poi inserite in un punto diverso del documento;
  - i comandi di cancellazione **d***x* (es. **dd**) copiano nel buffer il testo che viene eliminato (ctrl-x di Word);
  - il comando:
    - *n* **yy**

    copia le *n* righe sottostanti nel buffer (ctrl-c di Word);
  - per incollare ("*paste*") il contenuto del buffer in un punto diverso del documento (ctrl-v di Word):
    - **p** copia il buffer a partire dalla riga sottostante;
    - **P** copia il buffer a partire dalla riga soprastante.

L'editor vi (7)

- Ricerca di testo:
  - **/*stringa*** cerca la stringa desiderata;
  - **n** cerca l'occorrenza successiva.
- Sostituzione di testo:
  - **:% s/*stringa1*/*stringa2*/g**

    sostituisce tutte le occorrenze di *stringa1* con *stringa2*, nell'intero documento;
  - **:.,. s/*stringa1*/*stringa2*/g**

    sostituisce tutte le occorrenze di *stringa1* nella sola riga corrente.
- Inserimento del contenuto di un altro file:
  - **:*n* r *nomefile***

    inserisce il contenuto del file *nomefile* dopo la riga *n*.

"La rete non funziona!" (1)

- Se da un sistema Unix (ma non solo...) non riuscite a collegarvi ad un determinato nodo (via ftp, telnet, ssh, scp, http, ...), potete fare qualche controllo per avere un'idea di cosa sta succedendo.
- **ping** *indirizzo*
  - Il comando **ping** invia dei pacchetti di dati con dimensione fissa al nodo indicato, e verifica se questo risponde e in quanto tempo. Se non specificato altrimenti, viene inviata una sequenza infinita di pacchetti (Ctrl-c per fermarla).

    Se il nodo risponde, vuol dire che è raggiungibile in rete: forse solo il servizio che stavate cercando di utilizzare è inattivo (temporaneamente o definitivamente) oppure il nodo si trova in uno stato anomalo ma... la rete funziona!

"La rete non funziona!" (2)

- Se il nodo non risponde al ping, vuol dire che probabilmente non è raggiungibile (il nodo potrebbe anche essere configurato in modo da non rispondere al ping). Per tentare di capire dove si trova l'interruzione nel collegamento, utilizzate il comando **traceroute**, che mostra passo passo tutti i nodi attraversati per giungere a destinazione. Se il report di traceroute resta "appeso" ad un certo punto, avete scoperto dov'è l'interruzione.

```
$ traceroute www.cern.ch
1  140.105.6.254 (140.105.6.254)  0.404 ms  0.294 ms  0.252 ms
2  rc-infnts.ts.garr.net (193.206.132.5)  2.072 ms  2.026 ms  1.957 ms
3  mi-ts-2.garr.net (193.206.134.149)  8.149 ms  8.027 ms  8.430 ms
4  rtg-rt.mi.garr.net (193.206.134.206)  9.740 ms  9.396 ms  9.098 ms
5  garr.it1.it.geant.net (62.40.103.89)  9.598 ms  9.491 ms  11.079 ms
6  it.ch1.ch.geant.net (62.40.96.33)  22.743 ms  22.469 ms  22.484 ms
7  swice2-p6-1.switch.ch (62.40.103.18)  21.742 ms  21.686 ms  21.756 ms
8  cernh7-gb1-1.cern.ch (192.65.184.222)  21.777 ms  21.808 ms  21.807 ms
9  cernh2.cern.ch (192.65.185.2)  26.777 ms  22.944 ms  23.206 ms
10 webr2.cern.ch (137.138.28.230)  24.090 ms  22.920 ms  23.560 ms
```


"La rete non funziona!" (3)

- Se il nodo risulta "sconosciuto", ed avete indicato il nome, potete provare con l'indirizzo numerico. Per conoscere l'indirizzo numerico a partire dal nome di un nodo, o anche viceversa, usate il comando **nslookup**:

```
$ nslookup nome-oppure-indirizzo
```

  Ad esempio:

```
$ nslookup quark.ts.infn.it
$ nslookup 140.105.6.151
```

- Su Linux viene consigliato l'uso dei comandi **host** oppure **dig**.
- In tutti i casi sopra indicati, se non viene restituito l'indirizzo numerico a partire dal nome (o viceversa) vuol dire che il nodo non è stato trovato sul Name Server: per questo risulta sconosciuto!

NFS (Network File System) (1)

- È un'architettura *client/server* che consente la **condivisione centralizzata** di *file system* fra diversi calcolatori in rete, anche con architetture diverse (PC Win, PC Linux, Sun, HP, ...).
- Permette la gestione centralizzata di file acceduti da diversi computer.
- Esempio: nella Sezione di Trieste lo spazio disco dove si trovano le *home directory* degli utenti (/usr/users) è fornito da un *server* NFS a tutti i nodi che hanno bisogno di "vedere" il medesimo disco utenti.
- Ogni *client* NFS monta i *file system* forniti da un *server* NFS e li "vede" come se fossero dischi "locali":

```
$ df
Filesystem   1k-blocks      Used Available Use% Mounted on
/dev/hda2      4127108   1909784   2007676  49% /
/dev/hda1        31079      2880     26595  10% /boot
/dev/hda3      9661440     32828   9137828   1% /home
/dev/hda6       101089     19976     75894  21% /usr/vice
nas:/users    20120992   5111096  15009896  26% /usr/users
```


NFS (Network File System) (2)

- **Server**:
  - È il nodo che esporta i *file system*.
  - Quando un *client* richiede un certo *file system* fornito dal *server*, quest'ultimo controlla che il *client* sia autorizzato a ricevere quel *file system*, nel *file* /etc/exports.
- **Client**:
  - Tramite un comando di **mount**, richiede al *server* un certo *file system*.
- Di solito i *file system* necessari al sistema vengono montati automaticamente al *boot*.
- In qualche caso l'utente può avere necessità di montare un *file system* via NFS. Per maggiori dettagli consultare **man mount**.

X windows (1)

- **Xwindows** è un **sistema grafico a finestre** ormai quasi standard su vari sistemi.
- È **indipendente** dal sistema operativo (a differenza di altri sistemi a finestre).
- È **modulare**: ogni componente (le librerie, l'interfaccia grafica, la gestione della sessione, ecc.) è indipendente e può essere gestito autonomamente.
- È basato su un'architettura **client-server**: le applicazioni grafiche (client) accedono al display di un nodo locale o remoto attraverso un processo server. La comunicazione tra client e server può avvenire anche attraverso la rete.
- Le applicazioni X comunicano con un display attraverso un **protocollo di rete** chiamato *X protocol*.

X windows (2)

- Una volta instaurata la comunicazione tra applicazione e display, la connessione è **trasparente per l'utente**: l'applicazione può girare su un sistema remoto, accettando l'input di mouse e tastiera da un altro nodo, e inviando l'output su un display connesso a quest'ultimo. È quindi possibile eseguire un'applicazione grafica (ad es. ghostview, xcalc, ecc.) su un nodo remoto e visualizzare l'output grafico sul display locale.
- Il server può abilitare/disabilitare i client alla connessione X.
- Il comando **xhost** permette di controllare l'abilitazione. Viene eseguito sul server, ovvero il nodo sul quale si trova il display grafico:

```
[tizio@mynode] $ xhost + remotenode
[tizio@mynode] $ xhost – remotenode
```

- Per default solo il nodo locale è abilitato alla connessione con l'X server.

X windows (3)

- Una volta abilitata la connessione, sul client dobbiamo ridefinire la variabile di ambiente DISPLAY, in modo che punti al server X. Il comando da utilizzare dipende dalla shell:
  - Con la tcsh:

```
[tizio@remotenode] setenv DISPLAY mynode.ts.infn.it:0
```

  - Con la bash:

```
[tizio@remotenode] export DISPLAY=mynode.ts.infn.it:0
```

- Il comando **ssh** supporta il sistema X: se dal nodo locale vi collegate in ssh su un nodo remoto, non è necessario ridefinire la variabile DISPLAY: l'output delle applicazioni grafiche eseguite sul nodo remoto viene comunque visualizzato sul display del server X su cui vi trovate.

Alcune fonti per l'approfondimento

- Elementi base sul sistema operativo Unix:
  - http://www.units.it/~nircdc/doc/baseunix/
- Dispensa sullo Unix:
  - http://www.units.it/~nircdc/doc/oldunix/DUunix.html
- Unix tutorial for beginners:
  - http://www.ee.surrey.ac.uk/Teaching/Unix/
- UNIXhelp for users:
  - http://unixhelp.ed.ac.uk/