Politecnico di Torino
sede di Torino

Unix in Supposte:
come porre rimedio alla "febbre" da Unix

Maurizio Tranchero

III Facoltà di Ingegneria: Ingegneria Elettronica
matricola 118920

A.A. 200x–200(x+1)

1 Unix

Unix è un sistema operativo multiutente e multitasking creato negli anni '70 nei laboratori AT&T da il signor Ritchie (sì, quello che insieme a Kernigan ha creato il C... e questo dice molto) e Thompson. Ora molte industrie producono una loro versione proprietaria di Unix adatto ai sistemi che progettano (HW o SW) ad esempio c'è AIX di IBM, Solaris di SUN Microsystem... GNU/Linux invece è una libera implementazione di unix fatto da una serie di appassionati che permette di utilizzare tutta la potenza di Unix senza sborsare migliaia di euro per acquistare una licenza di un OS proprietario. Inoltre Linux è una implementazione regolamentata dalla GPL [GNU Public License (dove GNU sta per GNU is Not Unix, lo so è un acronimo ricorsivo, ma si sa che gli informatici sono strani)], che obbliga chi riutilizza porzioni del codice sotto la sua egida all'interno dei propri programmi, a distribuire il codice sorgente dell'intera applicazione (ma non vieta di venderla ad un costo esorbitante). Esistono altre implemntazioni di Unix libere come i vari BSD, che sono regolati da una licenza differente (BSD-licence) che non vincola il SW distribuito se non nel riportare il nome dell'autore originario.

Fatta quesrta doverosa introduzione sul mondo Unix, passiamo alle "supposte" promesse dal titolo. Unix è un sistema piuttosto sicuro (grazie alla sua struttura ed alla sua longevità) e per questo richiede di default un nominativo per ogni utente ed una password per permettere l'accesso alle macchine. Sempre per ragioni di sicurezza all'atto del login i caratteri della password non vengono visualizzati neppure come asterischi, per impedire di carpire la lunghezza della medesima. Inoltre la password deve essere composta da almeno (non è detto, ma spesso è così) 8 caratteri tra i quali ci deve essere almeno un carattere speciale, una lettera maiuscola o un numero. Gli utenti sono creati e gestiti dall'amministratore del sistema: l'utente `root`, colui che tutto può e sa.

Ah dimenticavo un'altra caratteristica di Unix è il fatto che sia, a differenza di altri ambienti, *case sensitive*: cd, cD, Cd, CD sono tutti nomi validi e differenti, quindi possono tutti convivere nella stessa directory senza problemi.

I comandi unix standard sono altamente specializzati, ossia ogni comando esegue una sola operazione, ma la fa al meglio, seguendo la filosofia K.I.S.S. (keep it simple stupid). Inoltre i programmi unix solitamente non forniscono un output se finiscono bene (d'altra parte uno è certo della corretta terminazione, non c'è il caso di riempire il terminale con acknowledge di qualsivoglia specie, è più importante ottenere un codice in caso di errore). Quindi quando io lancio un comando di rimozione (ne parlerò dopo) il comando di rimozione, se ha i permessi necessari per eseguire l'ordine, lo fa e sta zitto, altrimenti mi

comunica la mancata esecuzione specificando il problema in cui è incappato.

Altro punto a favore di unix è che la quasi totalità dei file di configurazione è fatta da di file ASCII, pienamente comprensibili dagli utenti (non come certi registri di configurazione presenti in alcuni OS che sono non documentati ed incomprensibili).

Ho detto che Unix è multitasking, questo vuol dire che possono essere eseguiti più programmi alla volta, ma se io sto usando un programma sotto un terminale come faccio a lanciarne un altro? Semplice è sufficiente lanciare il comando seguito dal carattere *e-commerciale* che dice al sistema: "mandalo in background". Ma, a volte, uno si dimentica di farlo e quindi può essere necessario avere un modo di interromperne momentaneamente l'esecuzione, questo si fa utilizzando la combinazione di tasti `CTRL+Z`, che manda in stop il processo, liberando il terminale. Se ora voglio posso usare il terminale per fare ciò che voglio, ma il programma interrotto è stato messo in condizione di non dare né ricevere risposte, quindi è come se fosse inattivo (non puoi clickare sui pulsanti, scriverci su...) solitamente questa situazione si traduce in due casi:

- uso la shell per un po' e poi voglio ritornare sul programma stoppato, per far questo mi è sufficiente scrivere il comanfo `fg` che riporta il processo in foreground;
- voglio utilizzare la shell per un tempo indefinito e, al contempo, continuare ad usare il programma in pausa come se fosse in background, questo si ha digitando il comando `bg` sulla console in questione.

Analizzati questi aspetti del sistema, nei prossimi paragrafi vedremo alcun caratteristiche importanti della struttura di Unix ed alcuni comandi utili.

2 Struttura del File System

Importante, per capire alcune caratteristiche di Unix, è capire come è organizzata la struttura delle directory. Tutto parte dalla "root" directory (non l'utente root, la directory), che rappresenta l'origine di tutto (come l'utente root rappresenta l'origine di ogni cosa), da essa si diramano tutte le altre. Questo è un punto di differenza notevole rispetto al mondo Windows, infatti mentre in windows la struttura parte dal disco che la contiene, la struttura del file-system unix è a più alto livello di astrazione: le partizioni fisiche del disco vengono *montate* al caricamento dell'OS sotto determinati *mount point* (che in realtà sono nodi dell'albero delle directory). Grazie a questo posso avere partizioni situate su dischi differenti, anche remoti, ma che vengono

viste sempre come directory locali e quindi non ho problemi del tipo: "ho inserito la mia chiavetta USB sul mio PC, il sistema l'ha chiamata disco E e mi ha spostato il disco E come disco F, ma ora il mio programma XYZ[1]non funziona più perché ha le librerie sull'ex disco E", cose che possono capitare su sistemi operativi poco evoluti...

Questa è la struttura standard di directory in un sistema Unix

```
|-- bin
|-- boot
|-- dev
|-- etc
|-- home
|-- initrd
|-- lib
|-- mnt
|-- opt
|-- proc
|-- root
|-- sbin
|-- sys
|-- tmp
|-- usr
`-- var
```

dove le cartelle (principali) contengono:

bin gli eseguibili di sistema e di base;

boot file necessari per il boot del sistema;

dev tutti i device fisici connettibili al sistema;

etc tutti i file di configurazione generale dei programmi;

home tutte le home directory degli utenti registrati sul sistema (eccetto l'amministratore *root*;

lib le librerie condivise dai programmi;

mnt i mount point dei dispositivi removibili (cdrom, zip, floppy...);

sbin programmi di amministrazione e quindi molto spesso utilizzabili dal solo utente root;

[1]Nota bene non è il programma di City Hunter (Ah, ah, ah che bellissima battuta)

usr i programmi, le documentazioni per gli utenti.

Facciamo un esempio, supponiamo di avere un disco con 4 partizioni: 1 con windows (la prima), 2 con linux (una partizione di root ed una di home) ed una di swap[2]. All'avvio del SO viene analizzato il file `fstab` che contiena la tabella delle partizioni e quindi contiene delle linee che descrivono le 4 partizioni precedenti e come debbono essere utilizzate. A questo punto dal file viene letto che esiste una partizione radice, da cui dipendono le altre e che va montata per prima e questo viene fatto; poi viene individuata la partizione di swap (che permette la paginazione su disco in caso la memoria fisica non sia sufficiente per i nostri usi) e viene montata in modo da poter essere utilizzata. La situazione è la presente

```
/
|
+-/dev/[tanti file]
|
+-/boot/[tanti file]
|
+-/mnt/[tanti file]
|
+-/home/
|
+-/var/[tanti file]
|
+-/usr/[tanti file]
|
+-/local/[tanti file]
[...]
```

come si vede dalle linee precedenti (immense le mie doti di grafico), la directory /home/ non contiene file al suo interno: questo avviene perché la partizione contenente la directory /home/ non è stata ancora montata. Ora, andando avanti nella descrizione del processo di boot, viene montata la home e quindi il processo di boot può continuare. La directory con windows non è detto che venga montata all'avvio, ma è possibile che sia data all'utente la possibilità di farlo in caso di bisogno in un secondo momento. Come spero abbiate intuito (e se non è così è colpa mia che non so spiegarmi bene) è possibile mettere ogni singola directory su di un dispositivo differente (o su di una partizione differente) in modo da poterne controllare le dimensioni, in

[2]La partizione di swap è una parte del disco rigido utilizzata dal sistema operativo per la paginazione su disco

modo da poter implementare politiche di backup (tutto ciò che mi serve è in una o due directory di un disco che ad esempio può essere rimuovibile).

Come preannunciato le partizioni e i dispositivi possono essere montati e smontati a run-time, a seconda delle necessità dell'utente. Il comando atto alla gestione delle partizioni è `mount`. `mount` si occupa di leggere il file /etc/fstab e di agire di conseguenza caricando la partizione passata come argomento nella locazione del filesystem relativa. Ad esempio

```
mount /dev/sda1
```

monta il device contenuto in /dev/sda1 (tipica chiavetta USB) e lo monta nella directory di /mnt/ ad esso assegnato. Invece per la rimozione del dispositivo, non è possibile toglierlo e basta perché altrimenti si rischia di corrompere il FS del dispositivo in questione. Infatti ogni trasferimento di dati da o per un sistema removibile, è fatto in maniera dilazionata da parte del kernel: sono operaioni asicrone e quindi fino a quando il disposotivo non è smontato non siamo sicuri che il trasferimento sia terminato. Il comando per questo è `umount` che ha la seguente sintassi

```
umount /dev/nome_device
```

Prima di affrontare la trattazione degli altri principali comandi utili per l'utilizzo di un sistema Unix, è necessario introdurre il più importante: `man`. Il comando `man` è l'accesso principale alla documentazione del sistema e quindi permette di accedere all'help di ogni eseguibile/interpretabile che disponga di una propria "man page". Infatti digitando

```
man nome_comando
```

si ottiene come output la stampa video delle informazioni relative all'utilizzo, alle opzioni e, a volte, alcuni esempi relativi al parametro specificato (`nome_comando`), che può essere sia un programma di sistema sia un programma utente, sia la funzione (o la libreria) di un linguaggio di programmazione (ad esempio se non mi ricordassi la sintassi della `printf()` del C, mi basterebbe digitare

```
man printf
```

per ottenere un risultato simile a questo

```
INTF(1)                  User Commands                  PRINTF(1)

NAME
       printf - format and print data
```

```
SYNOPSIS
       printf FORMAT [ARGUMENT]...
       printf OPTION

DESCRIPTION
   NOTE:  your shell may have its own version of printf which
       will supercede the version described here. Please refer to
       your shell's documentation for details about the options
       it supports.

       Print ARGUMENT(s) according to FORMAT.

       --help display this help and exit

       --version
              output version information and exit

       FORMAT controls the output as in C printf.  Interpreted
       sequences are:

       \"     double quote
[...]
```

) quindi tenetevi bene a mente il comando `man`, in quanto è un'attrezzo molto utile che deve essere consultato sempre in caso di problemi.

2.1 Permessi sui file

Essendo un sistema multiutente, per garantire una certa privacy agli utenti è prevista la possibilità di impostare dei permessi a ciascun file-directory. Ogni utente, quando creato, viene aggiunto ad un gruppo, per questo ogni file ha tre tipi di permessi (lettura, scrittura, esecuzione) sia per il proprietario, sia per il gruppo di appartenenza e sia per gli altri utenti. I permessi si impostano utilizzando il comando

```
chmod [options] mode file...
```

dove options (è tra parentesi quadre quindi è un'opzione facoltativa, mentre mode e file sono parametri necessari ed indicano rispettivamente il modo di protezione e il/i file a cui si devono applicare. I modi di protezione possono essere espressi in ottale o in maniera "a carattere". In ottale è sufficiente

utilizzare le nostre competenze con il calcolo in binario, considerando che i permessi sono in ordine *rwx* (reading, writing ed execution); e sono indicati per proprietario, gruppo ed altri si ottiene che se voglio impostare i permessi di lettura ed esecuzione (101b=5) al proprietario, modifica e lettura (110b=6) al gruppo e nessuno (000b=0) agli altri devo operare così:

```
chmod  560 file...
```

I permessi "a carattere" sono altrettanto semplici: è sufficiente indicare l'obiettivo del cambio (u=user, g=group, o=others —non nel gruppo—, a=all users) e se si vuole aggiungere o togliere l'attributo. In *pauca dicta*

```
chmod  a+x file...
```

abilita l'esecuzione per ogni utente,

```
chmod  ug-x file...
```

disabilita l'esecuzione per il proprietario e per il gruppo e via dicendo.

Ora rimangono le opzioni. Ce ne sono diverse, ma più utilizzata è "-R" che dice di effettuare la modifica per tutti i file e le sottodirectory a partire da quella data per argomento.

PS: da notarsi che un programma non è utilizzabile finché non ha i permessi di esecuzione settati, non è sufficiente che sia un eseguibile, devo abilitare esplicitamente l'esecuzione su di esso.

2.2 Comandi di gestione directory e file

Già ma come faccio a sapere i file presenti in una directory e quali sono i permessi ad essi associati? Semplice con il comando `ls`. `ls` (list) permette di effettuare esplorazioni all'interno di directory e di conoscere alcuni dettagli su di loro. Lanciato da solo permette di conoscere semplicemente il contenuto della cartella, ma presenta alcune opzioni interessanti:

- "-l" (long format) visualizza una lista che include i permessi dei file, il numero di link ad esso associati (ve lo spiego dopo) il proprietario, il gruppo la dimensione in byte la data e l'ora di cambiamento del file ed il nome del file stesso;
- "-a" visualizza tutti i file anche quelli nascosti, ma in modalità normale (cioè come `ls` e non come `ls -l`);

Abbiamo parlato di link ad un file, ma cosa sono? Di link in Unix ce ne sono di due categorie: hard e simbolici. Un link hard si crea così

```
ln nome_sorgente nome_destinazione
```

e crea un secondo nome al file sorgente, ossia un altro modo per poterlo chiamare, se io rimuovo `nome_sorgente`, e vedo a vedere il link, lo trovo ancora bello e vispo, perché un file viene eliminato solo quando tutti i suoi hard link sono stati rimossi. Un link simbolico invece si crea così

```
ln -s nome_sorgente nome_destinazione
```

ed assume il classico significato di puntatore al file che indica solo un modo per raggiunderlo, ma se simuovessi il file sorgente il link perderebbe la sua efficacia e diverrebbe inutilizzabile.

Altri comandi che possono tornare utili sono quelli per la creazione-rimozione delle directory: `mkdir` e `rmdir`. Entrambi hanno una sintassi molto simile a quella del comando `md` sotto ambiente DOS, infatti è sufficiente digitare

```
mkdir nome_dir
```

per creare una nuova cartella e

```
rmdir nome_dir
```

per rimuoverla — se la directory è completamente vuota. Nel caso questo non sia verificato si può utilizzare il comando `rm`. Questo comando è utile per rimuovere ogni tipo di file, ma attenzione è un comando unix e quindi non chiede conferma di quanto sta facendo — d'altra parte voi siete utenti unix ergo non sbagliate mai. La sintassi di `rm` è semplice:

```
rm nome_file
```

questo rimuove un file su cui ho permissi di scrittura else no; alcuni parametri utili sono:

- "-r" (recursive) che permette di rimuovere una directory e tutto il suo contenuto (permessi permettendo);
- "-i" opzione per newbie, utilizzata perché dice al programma "sono un povero utente windows che agisce senza pensare, mi puoi chiedere conferma di ogni file che voglio eliminare?"

ATTENZIONE! Non utilizzate mai il seguente comando tratto dal mondo DOS

```
rm -r *.*
```

perché così state dicendo al sistema operativo "rimuovi ricorsivamente ogni file che a partire dalla directory corrente che corrisponde a *.*". A prima vista questo può sembrare innoquo, ma vediamo come viene interpretato dalla shell:

1. il carattere "*" viene interpretato come zero, una o più occorrenze di un qualsiasi carattere, ma mai il carattere "." se si trova ininizio di espressione; il primo asterisco, quindi, può assumere il significato di stringa vuota;

2. il secondo asterisco non ha limitazioni di significato quindi può essere interpretato come carattere ".";

3. il comando viene interpretato come

   ```
   rm -r ..
   ```

   che comporta la rimozione di tutta la directory superiore al punto dove ci troviamo;

4. ora basti pensare che l'interpretazione viene effettuata ogni volta che si cambia di directory per vedere come facilmente si procede alla rimozione di tutto il file-system.

Quindi non usatelo mai neppure per sbaglio, in quanto anche se non siete amministratori di sistema e quindi non avete i permessi di scrittura su ogni file, richiereste di eliminare ogni file presente nella vostra directory home. A volte è però necessario utilizzare un comando che elimini tutti i file e le cartelle all'interno di una directory, questo risultato si ottiene mediante `rm -r *`.

Un altro comando utile per la gestione dei file è il comando `grep` che si occupa di ricercare stringhe all'interno di altri file. La sintassi più utilizzata è la seguente

```
grep -i stringa file0 file1 file2 ...
```

in cui il parametro "-i" è opzionale ed indica la ricerca case insensitive (se omesso la ricerca è effettuata case sensitive), stringa è la stringa da ricercare e i file*n* sono i file in cui viene effettuata la ricerca.

Per effettuare la copia di un file si utilizza il comando `cp` con sintassi

```
cp [-i -r] sorgente destinazione
```

dove sorgente è il nome del file sorgente e destinazione è il path della destinazione in cui va posizionato il nuovo file. Attenzione la destinazione va sempre espressa anche quando si intende la directory corrente; in questo caso è comodo utilizzare il carattere "." per indicare la directory corrente. L'opzione "-i" è analoga a quella in `rm` mi chiede conferma in caso di sovrascrittura di file già esistenti; l'opzione "-r" effettua la copia ricorsiva (utile nel caso di directory). Spesso è necessario rinominare i file o spostarli, per questo si utilizza il comando `mv`. La sintassi è semplice

```
mv [-i] sorgente destinazione
```

e rispecchia la sintassi di `cp`.

Spesso è utile creare un archivio di file e comprimerlo per effettuare più efficientemente trasferimenti voluminosi, in queste condizioni ci possono tornare utili il programma `tar` e il programma `gzip`. Il comando `tar` si occupa di archiviare in un unico file tutti quelli passatigli per argomenti e di effettuare anche l'operazione contraria. Questa è la sintassi

```
tar [options] file0 file1 file2 ...
```

dove le opzioni solitamente utilizzate sono:

- x (extract) estrae dal file .tar i file contenuti;
- z (unzip) comprime-decomprime gli archivi compressi con `gzip`;
- j (bunzip) comprime-decomprime gli archivi compressi con `bzip2`;
- t (test) mostra il contenuto di un archivio;
- f (file) indica che il parametro successivo è il nome del file su cui si vuole effettuare l'operazione;
- v (verbose) è prolisso dicendoti cosa sta facendo;
- c (compress) archivia i file.

`tar` si occupa di archiviare (anche se le opzioni moderne "z" e "j" permettono anche di effettuare compressioni-decompressioni), ma per compattare si usa il programma `gzip`. Questo è un programma molto semplice da utilizzare, infatti con

```
gzip file
```

si comprime il file, che cambierà il suo nome aggiungendo al nome del file l'estensione ".gz"; mentre con

```
gunzip file.gz
```

si decomprime "file.gz" che cambia il suo nome divenendo "file". Per concludere questi due comandi si utilizzano spesso in combinazione per creare archivi compressi. Per la compressione si opera in siffatta maniera

```
tar cv file0 file1 ... | gzip > archivio.tar.gz
```

oppure

```
tar czvf archivio.tar.gz file0 file1 ...
```

che utilizza il carattere *pipe* "|", che si occupa di redirigere l'output del programma alla sua sinstra nell'input del programma alla sua destra, e il carattere "¿" che si occupa di redirigere su di un file l'output del programma alla sua sinistra. Per la decompressione, invece si procede così

```
gunzip archivio.tar.gz | tar xvf -
```

oppure

```
tar xzvf archivio.tar.gz
```

Per la ricerca dei file all'interno delle directory si usa il programma `find`. Questo programma ha una sintassi piuttosto complessa, ma cela una potenza ed una duttilità elevata. La sintassi è la seguente

```
find [path] [options] [pattern]
```

dove il path è il percorso in cui effettuare la ricerca; options può assumere varie connotazioni (tra cui -name, -iname, -size ...) e pattern è il nome della stringa di ricerca. Il pattern di ricerca può utilizzare le wildcars, ma è necessario anteporvi il backslash "\" come carattere di escape. Per una trattazione completa di `find` è meglio consultare la "man page" rispettiva; ma qui di seguito riporto alcuni tipici utilizzi: ricerca per

- nome (case sensitive)

  ```
  find /home/maurizio -name presepe.c
  ```

- nome (case insensitive)

  ```
  find . -iname brent_and_kung\*
  ```

Finisce così questa breve carrellata di informazioni sul sistema Unix; nei prossimi paragrafi si affronteranno srgomenti quali i client per il trasferimento e il login remoto sicuro su macchine Unix ed il miglior sistema di composizione testi, uno standard *de facto* nell'ambiente scientifico.

3 ssh ed sftp

Questi sono due servizi disponibili per utilizzare in remoto macchine unix su cui si ha un *account*. Qui di seguito riporto una breve descrizione della sintassi dei comandi e di come utilizzarli per operazioni semplici. Per un utilizzo più avanzato rimando alla documentazione ufficiale dei programmi, reperibile su ogni sistema unix digitando il comando

```
man nome_comando
```

dove `nome_comando` è il nome del comando di cui si vuole conoscere la sintassi o le caratteristiche.

3.1 ssh

ssh è un programma per il login su macchine remote e per l'esecuzione di comandi su di esse. Imita il comportamento di altri sistemi, come rlogin ed rsh, ma inoltre implementa un sistema di crittografia che permette di rendere sicura la sua esecuzione su di una rete insicura.

Per accedere ad una macchina su cui sia presente un servizio ssh, è necessario digitare il seguente comando:

```
ssh nome_utente_remoto@nome.macchina.remota
```

dove `nome_utente_remoto` è il login sulla macchina in questione, mentre `nome.macchina.remota` è l'indirizzo IP oppure il nome della macchina in questione. Il programma procede richiedendo l'inserimento della password di login sul sistema remoto.

Come preannunciato il servizio ssh è un metodo di eseguire comandi su di un host come se fosse locale, quindi dopo essersi connessi alla macchina si possono esegure tutti i comandi che sul pc remoto sono disponibili.

Alla fine della sessione per uscire dal servizio è necessario utilizzare il comando `quit`.

3.2 sftp

ssh è molto comodo per la gestione remota, ma per il trasferimento di dati è più conveniente utilizzare il protocollo sftp. Questo è la copia esatta di ftp, ma in più utilizza un metodo cifrato per garantire la sicurezza attraverso le reti ethernet. L'utilizzo è molto semplice: è sufficiente connettersi alla macchina come nel caso di ssh:

```
sftp nome_utente_remoto@nome.macchina.remota
```

ed anche qui dopo avere inserito la password di autentificazione viene visualizzato un prompt dei comandi. I comandi da utilizzare su sftp si possono suddividere in tre categorie: comandi locali, comandi remoti e comandi di trasferimento.

I comandi remoti hanno la sintassi ed un significato analogo ai comandi base di unix: `mkdir, cd, ls, pwd`... che servono per creare una directory, spostarsi tra di esse, elencare il contenuto di una o più di esse, indicare il nome della directory corrente..., ma sempre con effetto sulla macchina remota.

I comandi locali sono uguali nella sintassi ai precedenti, ma per distinguerli sono caratterizzati dal prefisso `l-` che li contraddistingue appunto come *locali*. Utilizzando quindi `lmkdir` viene creata una directory *locale*, `lcd` sposta l'utente nelle directory della macchina a cui è fisicamente collegato et cetera.

I comandi per il trasferimento, infine servono appunto da interfaccia tra *il mondo locale e il mondo remoto*: permettono di spostare file da un host ad un altro. Il comando per la trasmissione di un file sulla macchina remota è `put` che spedisce il file posto come argomento via rete. La sintassi è la seguente

```
put file0 file1 file2 ...
```

È possibile utilizzare le *wildcars* (*, ?), ma non è possibile utilizzare il metodo di *completion* tipico delle shell unix (se premo `TAB` dopo l'inizio di un comando-file non ottengo l'autocompletamento del medesimo); ecco un esempio

```
put file* *.vhd ...
```

Il comando per il recupero di un file sulla macchina remota è invece `get`. La sintassi è identica a quella di `put`, ma l'effetto è il contrario. Anche qui sono disponibili le wildcars.

Tutti questi comandi sono elencati dal programma se durante una sessione si digita come comando il carattere `?`.

Infine per uscire dal servizio è sufficiente digitare il comando `quit`.

4 LaTeX 2ε

LaTeX 2ε è un sistema di scrittura derivato da TeX. Al contrario dei sistemi WYSIWYG (come quelle porcherie di editor grafici tipo OpenOffice e MSWord) è un sistema del tipo WYMIWYG (what you mean is what you get). In pauca dicta è un linguaggio di programmazione ed in quanto tale ha una sintassi (piuttosto semplice), un compilatore (l'eseguibile `latex`) e

genera un file oggetto (i file `.dvi`) che linkato con le immagini creare il file di output vero e proprio (`.ps` o `.pdf`).

Ogni documento LaTeX è suddiviso in due parti il preambolo e il documento. Nel preambolo ci sono le definizioni del documento (stile, formato, lingua del sillabatore,...) i pacchetti da utilizzare (analoghe alle librerie dei programmi in C) ed eventuali comandi definiti dall'utente (le funzioni). Nel corpo del documento, invece, si trovano i file che vanno inclusi che contengono i vari capitoli del testo.

Cominciamo con un primo esempio di documento:

```
\documentclass[a4paper,10pt]{report}
\usepackage[italian]{babel}
\usepackage[latin1]{inputenc}
\usepackage[dvips]{graphicx}
% ora incomincia il documento
\begin{document}
\maketitle
\tableofcontens
\include{capitolo_1}
\include{capitolo_2}
...
\end{document}
```

dove `capitolo_n` rappresenta il nome di un file di testo con estensione `.tex` che contiene il testo del capitolo n-esimo (nb dentro il comando `\include` non si deve inserire l'estensione del file perché LaTeX cerca da sé il file corretto). Tutto ciò che precede il `\begin{document}` è da considerarsi preambolo, ciò che è preceduto da un carattere "%" è da considerarsi un commento e per questo viene ignorato dal compilatore. Il comando `\usepackage[...]{...}` serve per utilizzare i pacchetti (racchiusi tra graffe) con le opzioni definite tra le parentesi quadre. Tra i pacchetti utilizzabili ricordo

- babel, che si occupa di caricare le regole relative alla lingua usata nel documento per quanto riguarda la sillabazione delle parole;
- inputenc, che si occupa di gestire correttamente le lettere accentate (altrimennti sarebbe necessario, per ogni lettera accentata, scrivere `\'a` che in caso di necessità si può usare, ma a lungo andare stressa).
- graphicx, che si occupa di utilizzare come filtro per le immagini il programma tra quadre (al posto di dvips, ottimo per i file postscript, spesso si usa pdftex, per i file pdf).

Dopo queste dichiarazioni, inizia il documento vero e proprio grazie al comando di inizio ambiente (`\begin{}`) e la definizione dell'ambiente in questione (document per l'appunto). LaTeX utilizza molto gli ambienti nella struttura dei file, infatti ci sono ambienti per la scrittura di formule, per la gestione di elenchi, per le tabelle, le figure...

Ogni file utilizza una serie di comandi per impostare la suddivisione in capitoli, paragrafi, sottoparagrafi... Per dichiarare l'inizio di un nuovo capitolo si definisce con la seguente scrittura:

```
\chapter{Nome del Capitolo}
```

mentre per le sezioni e le sotto sezioni si usa

```
\section{nome del paragrafo}
\subsection{del sotto-paragrafo}
\subsubsection{del sotto-sotto-paragrafo}
```

Come si vede non è necessario utilizzare una numerazione in quanto ci pensa LaTeX a numerare ed a gestire questi dettagli lasciando a noi la decisione della struttura e i contenuti.

LaTeX non permette di utilizzare un livello di profondità nella struttura maggiore a 4 (non esiste qualcosa oltre il sotto-sotto-paragrafo) perché le norme dello scrivere sanciscono questo livello come massimo, inoltre di solito nella lingua italiana non si utilizza oltre il terzo livello.

4.1 Elenchi Puntati e Numerati

Spesso però è necessario creare elenchi puntati e numerati all'interno dei documenti, per questi si ricorre agli ambienti *itemize* e *enumerate*.

```
\begin{itemize}
\item pippo;
\item pluto;
\end{itemize}
```

che crea il seguente risultato

- pippo;
- pluto.

Analogamente si opera per l'ambiente enumerate (si cambia solo il nome dell'abiente, ma la struttura è la stessa)

```
\begin{enumerate}
\item pippo;
\item pluto;
\end{enumerate}
```

e questo è il risultato

1. pippo;
2. pluto.

4.2 Formule Matematiche

Ora veniamo al piatto forte di LaTeX: le formule matematiche. Ci sono vari ambienti per la gestione della matematica, ma il più usato è *displaymath*. Questo ambiente si occupa di visualizzare le formule inserite senza numerarle . Ora vediamo qualche esempio:

```
\begin{displaymath}
F(\omega) = \int_{-\infty}^{+infty} f(x) \cdot \e^{-j\omega x}dx
\end{displaymath}
```

da cui si ottiente (correttezza della formula a parte)

$$F(\omega) = \int_{-\infty}^{+\infty} f(x) \cdot e^{-j\omega x} dx$$

Come si può notare, per mettere un carattere in *superscript* è sufficiente usare `^` prima del carattere stesso; qualora, invece, si voglia mettere più caratteri ad apice, è necessario dire a LaTeX dove inizi e dove finisca la stringa a cui il comando `^` va applicato: questo si realizza racchiudendo il set di caratteri tra parentesi graffe, come nell'esempio successivo

```
x^{a quello che mi pare, ad es \frac{\int x dx}{\sum_{j=1}^n}=
y_{con il pedice che voglio}
```

$$x^{aquellochemipare,ades\frac{\int xdx}{\sum_{j=1}^{n}}} = y_{conilpedicechevoglio}$$

Nota bene che lo stesso comportamento ce l'hai anche con i pedici delle lettere. Come si è visto, nonostante io abbia messo degli spazi nelle diciture precedenti il modo matematico me li ha rimossi tutti. Infatti per inserire degli spazi tra le formule è necessario utilizzare i comandi `\,`, `\quad`, `\qquad` che sono riportati in ordine crescente per spazio.

Scrivere all'interno di un documento 150 formule e per 150 volte scrivere `\begin[displaymath]` è da mentecatti, per questo si può utilizzare un'abbreviazione `$$`. Infatti

```
$$
\int \cos(x) dx = \sin(x) +c
$$
```

dà luogo a

$$\int \cos(x)dx = \sin(x) + c$$

4.3 Inserimento di Tabelle e di Figure

Le tabelle in LaTeX si distinguono in due sottogruppi: fisse e mobili. Le prime sono fatte per essere messe in un luogo ben preciso, interrompono il testo e quindi non supportano le *caption*; le seconde sono oggetti che vengono gestiti in maniera abbastanza trasparente al programmatore dal compilatore, purtroppo (e dopo vi spiegherò il perché), e quindi necessitano di avere dei riferimenti incrociati e una descrizione (la caption appunto).

Vediamo ora una ripica tabella fissa il codice per la sua creazione è

```
\begin{tabular}{|c|c|c|}
\hline
Colonna 1 & Colonna 2 & Ultima colonna \\
\hline
11 & 12 & 13 \\
21 & 22 & 23 \\
31 & 32 & 33 \\
\hline
\end{tabular}
```

Colonna 1	Colonna 2	Ultima colonna
11	12	13
21	22	23
31	32	33

La sintassi, come sempre, è elementare[3]

- devo dichiarare l'ambiente con le istruzioni `\begin{.}` e `\end{.}`;
- devo definire il numero di colonne necessarie, tramite l'opzione passata all'ambiente itemize, `{|c|c|c|}` allo stesso tempo con questa opzione si definisce il numero di margini presenti (i caratteri *pipe* "|");
- metto i miei dati, separando le colonne con un carattere "&" e le righe con "\\";

[3]Sì, lo so che mi state maledicendo, ma io la trovo meravigliosamente semplice e potente allo stesso tempo, *de gustibus*...

- interpongo eventualmente delle linee orizzontali di separazione tra le righe, tramite il comando `\hline`.

Ok, mi direte, bello, ma... la tabella non è centrata: fa pietà all'interno di un documento! Avete ragione, ma a quasi tutto c'è un rimedio, in questo caso arriva in nostro aiuto l'ambiente *center*. Infatti è sufficiente inserire prima e dopo la dichiarazione della tabella

```
\begin{center}
\end{center}
```

ed otterrete questo risultato

Colonna 1	Colonna 2	Ultima colonna
11	12	13
21	22	23
31	32	33

Ora è proprio stupenda!

Mancano ancora le tabelle mobili (o flottanti), queste sono in realtà tabelle come le precedenti, ma si comportano come le figure (infatti sono anch'esse oggetti mobili).

```
\begin{table}[!hbtp]      % luogo per l'inserimento delle regole di
                          % posizionamento
\begin{center}
\begin{tabular}{|c|c|c|}
\hline
foo & bar & foobar\\
\hline
\end{tabular}
\caption{Questa è una tabella particolarmente riuscita}
\label{tab:tabella_che_mi_piace_particolarmente}
\end{center}
\end{table}
```

foo	bar	foobar

Tabella 1: Questa è una tabella particolarmente riuscita

In sostanza sono tabelle con una caption ed una label: la caption è la didascalia che appare al di sotto della tabella,mentre la label è un riferimentoe, un puntatore di C-iana memoria, che permette di identificare in maniera univoca l'oggetto cui punta per permettere di referenziarlo tramite il comando

```
\ref{tab:tabella_che_mi_piace_particolarmente}
```

il risultato lo potete vedere nella tabella 1.

Gli oggetti mobili sono impaginati autonomamente dal compilatore secondo delle regole ben definite dal corretto scrivere, ad esempio un'immagine non vi taglia una frase a metà, non viene messa a metà di una pagina, ma o all'inizio, o alla fine o in una pagina a parte. Tutto ciò è molto bello se devo fare un testo piuttosto lungo, ma può causare problemi se il testo è molto corto e ricco di figure (vedi relazioni di laboratorio). Infatti a volte capita che LaTeX sposti una figura un po' più in là per rispettare le sue regole e questo sposta tutte le altre in successione, ma allora quella successiva essendo stata spostata, può trovarsi ancora fuori luogo e quindi LaTeX la sposta nuovamente... In definitiva si ottiene un effetto domino che porta alla fine di ogni capitolo ad avere un po' di immagini/tabelle accalcate. In realtà visto che TUTTE devono avere una caption ed una label[4] non si hanno grossi problemi di comprensione, ma leggere un documento in cui ci si riferisce ad un'immagine che arriva solo dopo 20 pagine, non è bello. Per questo è possibile by-passare le regole di LaTeX così utilizzando come parametro all'ambiente table uno di questi switch: `h t p b`, eventualmente preceduto dal carattere `!`. Questi vogliono dire: *"Senti, o complilatore, io sono un buon ingegnere di animo nobile che conosce le regole del corretto scrivere, indi per cui ti chiedo se, per favore, puoi mettermi la presente tabella ove io bramo"* (h=here, t=top, b=bottom e p=page (la mette all'interno di una pagina con sole figure)). Inoltre, come preannunciato, c'è il flag `!` che può essere aggiunto ai precedenti. Questo ha un significato leggermente differente dal precedente, infatti significa: *"Senti, o complilatore, io sono un buon ingegnere di animo nobile che conosce le regole del corretto scrivere, indi per cui mettimi la presente tabella ove io bramo e non rompere le siffatte ciufole!"*

A questo punto il compilatore farà quanto richiestogli ad ogni costo nel senso che può produrre degli effetti anche pietosi, come interrompere una pagina a metà, mettere la tabella (che non ci sarebbe stata) nella pagina successiva e, di qui, riprendere a scrivere; per questo è preferibile non utilizzare questa forzatura, ma essere un po' più soft (vedi non usare il punto esclamativo) e magari ridurre la dimensione della tabella in questione (se ovviamente possibile).

Per la gestione delle figure la cosa è oltremodo semplice: sono oggetti mobili e vanno inclusi; se si sa questo si sa già quasi tutto sulle figure. La sintassi per l'inserimento delle figure è

```
\begin{figure}[h]
```

[4]Questo non è imposto dal compilatore, ma se mi voglio riferire a qualcosa di non fisso è meglio (leggi, si deve) mettere un riferimento univoco.

```
\begin{center}
\includegraphics[width=.7\textwidth]{./immagine_meravigliosa}
\caption{Questa è un'immagine meravigliosa.}
\label{fig:img}
\end{center}
\end{figure}
```

da cui si ottiene ciò che è riportato nella figura 1. Il comando per l'inclusione del file è `\includegraphics[]{}` che si occupa di caricare la figura relativa e immetterla secondo i parametri indicari tra quadre; in questo caso la scala in modo da rendere la sua larghezza pari a 0.7 volte la larghezza della pagina.

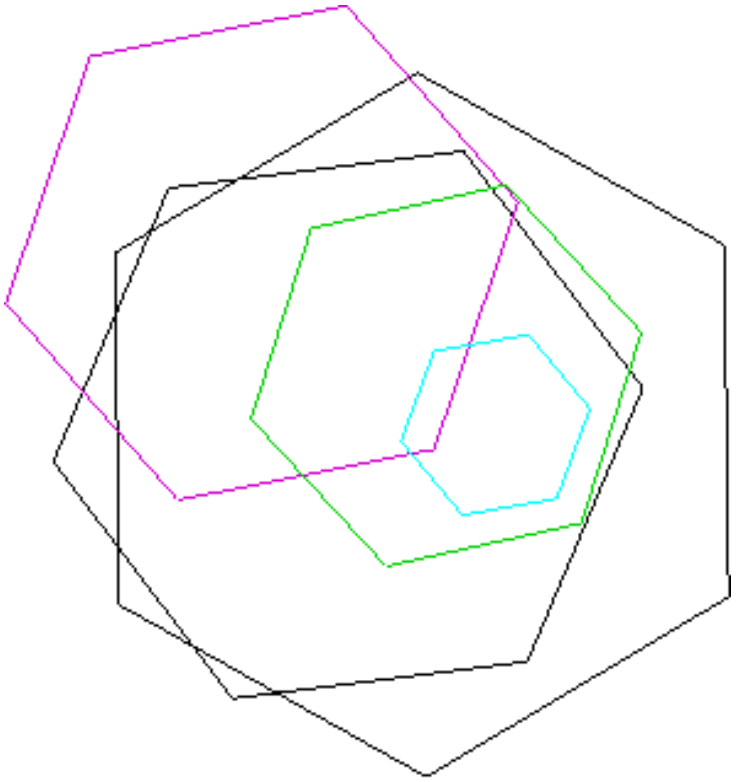

Figura 1: Questa è un'immagine meravigliosa.

Riferimenti bibliografici

[1] . Giacomini, *Appunti di informatica libera*, reperibile presso `http://a2.swlibero.it/`.

[2] . J. Volkerding, *The Slackware linux installation guide*, reperibile presso `http://www.slackware.com/book/`.

[3] *Red Hat Linux Essentials*, Apogeo.

[4] . Oetiker, *Una (mica tanto) breve introduzione a LaTeX 2ε*, reperibile presso il sito `http://www.ctan.org/`.

[5] *Impara LaTeX e mettilo da parte*, reperibile presso il sito `http://www.ctan.org/`