

ROBERTO GIACOMELLI



GUIDA TEMATICA ALLA RIGA DI COMANDO



2013/01/04 — v.1.2

```
roberto@roberto-desktop: ~  
roberto@roberto-desktop:~$ title  
  
Titolo: Guida tematica alla riga di comando  
  
Collana: Le guide tematiche del GuIT  
Autore: Roberto Giacomelli  
E-mail: giaconet.mailbox@gmail.com
```

```
roberto@roberto-desktop: ~  
roberto@roberto-desktop:~$ copyright  
  
(C) 2012 Roberto Giacomelli  
Pubblicato in Italia con licenza Creative Commons license 2.5  
Attribuzione, Non commerciale, Condividi allo stesso modo  
  
I marchi Windows, Mac OS X, e Linux sono di proprietà riservata  
dei rispettivi detentori
```



INDICE

INDICE	III
PIANO DELLA GUIDA	V
1 LA RIGA DI COMANDO	1
1.1 Il concetto di base	1
1.2 Primi comandi	2
1.3 Avviare la riga di comando	3
1.3.1 La shell in Windows	3
1.3.2 La shell in Mac OS X	4
1.3.3 I sistemi operativi Linux	4
2 LA SHELL	5
2.1 La directory di lavoro	5
2.2 Variabili d'ambiente	6
2.3 Il path di sistema	7
2.3.1 Modificare il PATH in Windows.	8
2.3.2 Modificare il PATH in Linux.	9
2.3.3 Non modificare il PATH in Mac OS X.	9
3 ESERCITAZIONI	10
3.1 Cambiare directory di lavoro	10
3.2 Elencare i file	11
3.3 Spostare o copiare file	12
3.4 Rinominare file	12
3.5 Verificare l'installazione T _E X	13
3.6 Compilare un documento sorgente	15
3.6.1 L'opzione <code>-shell-escape</code>	15

3.7	Documentarsi con <code>texdoc</code>	16
3.8	Gestione di \TeX Live	17
4	ARGOMENTI AVANZATI	19
4.1	Espressioni regolari	19
4.1.1	Rinominare file	20
4.1.2	Correzione massiva di sorgenti	21
5	SCRIPTING	25
5.1	Sha bang#!	26
5.2	Esempi	27
5.2.1	Ancora sulle righe magiche	27
5.2.2	Upload via ftp	28
5.2.3	Uno script con Lua	29
	NOTE SU QUESTA GUIDA	31
	Licenza d'uso	31
	Colophon	31
	Collaborazione e ringraziamenti	32

PIANO DELLA GUIDA

Questa guida tematica è suddivisa in quattro sezioni principali:

1. è una presentazione concettuale della riga di comando sostenuta da accenni alla sua lunga storia (pagina [1](#));
2. è un sunto delle particolarità operative di base dell'ambiente di shell (pagina [5](#));
3. è lo svolgimento passo passo delle procedure per raggiungere le capacità e le conoscenze necessarie per lavorare autonomamente con la shell di sistema, da leggere anche indipendentemente dalle altre sezioni (pagina [10](#));
4. è un compendio di argomenti avanzati (pagina [19](#)) assieme alla presentazione delle tecniche di *scripting* (pagina [25](#)).

La modalità di consultazione dipende dalle conoscenze e dagli obiettivi del lettore. Probabilmente chi già utilizza la shell sarà interessato ad uno sguardo sugli aspetti generali ed ad un approfondimento su quelli specifici per l'utilizzatore T_EX, mentre il neofita potrebbe volersi cimentare subito con l'indirizzo pratico della terza sezione e sorvolare sui concetti che la sovrintendono.

Comunque sia, auguro a tutti una buona e spero proficua lettura. Scrivetemi senza indugio un messaggio di posta elettronica per segnalarmi errori o miglioramenti o richieste di chiarimenti, e spero mi mandate le vostre impressioni, soprattutto se non avevate mai lavorato con la riga di comando prima d'ora :-)

Roberto Giacomelli
giaconet dot mailbox at gmail dot com

Prima della comparsa delle interfacce grafiche — ideate al Palo Alto Research Center in California ed implementate da Apple prima in Lisa e poi in Macintosh nel 1983 — l'interazione con i programmi avveniva con una modalità testuale chiamata *riga di comando*, ancora oggi disponibile sui moderni elaboratori come componente essenziale e metodo efficiente di elaborare dati.

I componenti del sistema T_EX possono essere utilizzati per mezzo della sola interfaccia a riga di comando — ed è per questo che gli utilizzatori potrebbero avvantaggiarsi se la conoscessero. Ciò non toglie che componenti indipendenti come gli *shell editor*, possano costituire un'interfaccia grafica verso i programmi di composizione della famiglia T_EX.

1.1 IL CONCETTO DI BASE

Definizione di *Riga di comando*

La *riga di comando* è un ambiente testuale in cui si impartiscono istruzioni digitando nomi di programmi con eventuali argomenti

L'ambiente testuale che consente all'utente di interagire con il sistema è chiamato *shell*. La shell accetta i dati di ingresso sotto forma di *comando*, gestisce l'esecuzione ad essi corrispondente e riceve i dati di uscita destinati all'utente. Nulla vieta che informazioni di ingresso o di uscita siano memorizzati in file su disco.

La riga di comando ha una lunga storia ed è naturale che le siano stati attribuiti nomi diversi per definirla nell'ambito di un particolare sistema operativo — *linea di comando*, *terminale*, *console* sono fra questi — tuttavia i concetti che la definiscono sono ancora quelli perfezionati da

Dennis Ritchie, Ken Thompson, Brian Kernighan e da altri programmatori esperti dei laboratori Bell sul finire degli anni '60 con Unix.

Tra le innovazioni del sistema operativo Unix, la shell rappresentava l'idea che piccoli e veloci programmi specializzati in compiti precisi, potessero essere concatenati in una *pipeline* adottando l'output dell'elaborazione come input per il programma successivo. La filosofia degli strumenti tende a garantire lo sviluppo efficiente del software senza limitare la complessità dell'elaborazione.

Oggi tra i sistemi desktop più diffusi, Mac OS X e Linux, sono basati sulla struttura di Unix, mentre Windows ha seguito uno sviluppo parallelo replicando ed ampliando l'ambiente di MS-DOS con il ruolo di shell.

1.2 PRIMI COMANDI

Il formato delle istruzioni prevede in particolare la digitazione in un'unica riga del nome del programma, seguito da eventuali opzioni e da eventuali argomenti. L'esecuzione ha inizio premendo il tasto invio.

```
$ nomeprogramma [<opzioni>] [<argomenti>]
```

Solitamente le opzioni vengono distinte dai dati premettendo al loro nome uno o due trattini – oppure uno slash. Se per esempio si vuol conoscere la versione di un programma è sufficiente eseguirlo con l'opzione `-version`. Con `pdftex`, il principale programma di composizione del sistema \TeX , otteniamo:

```
$ pdftex -version
pdfTeX 3.1415926-1.40.11-2.2 (TeX Live 2010)
... eccetera
```

Nel comando precedente non vi sono argomenti ma solamente un'opzione che provoca la stampa a video di un breve testo contenente le indicazioni di versione del programma `pdftex` installato sul sistema. Altri programmi della riga di comando utilizzano invece altre chiavi per emettere il testo d'aiuto a dimostrazione che non esiste un unico standard rispettato da comandi o shell.

Se consideriamo l'opzione `-help` oppure `-?` si richiederà la stampa sintetica della sintassi prevista con l'elenco delle opzioni disponibili e brevi testi esplicativi. Proviamo al terminale:


```
$ pdftex -help
Usage: pdftex [OPTION]... [TEXNAME[.tex]] [COMMANDS]
    or: pdftex [OPTION]... \FIRST-LINE
    or: pdftex [OPTION]... &FMT ARGS
... eccetera
```

Il \$ — o > per i sistemi Windows — è chiamato *prompt* ed è il segno dopo il quale si digitano i comandi con il compito di separare informazioni utili dai comandi stessi. Nella guida faremo uso del \$ riportandolo negli esempi come primo carattere a simboleggiare la shell.

A volte i comandi possono essere interattivi ovvero, una volta avviati, possono richiedere ulteriori dati in funzione dell'andamento dell'esecuzione.





1.3 AVVIARE LA RIGA DI COMANDO



Come vedremo, le modalità con cui si opera con la shell non sono poi così diverse tra i vari sistemi operativi. Molti concetti sono comuni e spesso i comandi di base hanno addirittura lo stesso nome e già questo è uno spunto di riflessione interessante.

Solitamente la shell si presenta come una normale finestra grafica — modalità detta in emulazione di terminale — al cui interno compare il cursore lampeggiante davanti al prompt in attesa di istruzioni.

Nei prossimi paragrafi impareremo ad accedere alla shell in Windows, Mac OS X, e Linux.

1.3.1 LA SHELL IN WINDOWS

In Windows la shell è chiamata *Prompt dei comandi*. Vi si accede dalla barra dei comandi    . Per comodità è possibile creare un shortcut o scorciatoia — un piccolo file puntatore ad un altro file — sul Desktop o nella barra delle applicazioni, così che basta un click per avviarla.

In Windows esiste una modalità particolare di avvio della shell che consiste nell'aprire il dialogo *Esegui...* dal menù Start e digitare il comando cmd prima di confermare su  o con il tasto .

1.3.2 LA SHELL IN MAC OS X

In Mac la riga di comando è rappresentata dall'applicazione **Terminale**, situata nella cartella *Utility* all'interno della cartella *Applicazioni*.

Potete anche trascinare l'icona Terminale dalla directory *Utility* di *Applicazioni*, alla *Dock Bar*. Per avviare la shell fate click sull'icona appena creata.

1.3.3 I SISTEMI OPERATIVI LINUX

In Linux la shell è di casa. Se il vostro Desktop Environment (DE) è Gnome utilizzerete **gnome-terminal** dal menù **Applicazioni** » **Accessori** » **Terminale**. In KDE l'emulatore di terminale è il programma **Konsole**.

Troverete molto comodo avviare il Terminale da tastiera con una combinazione di tasti, per esempio **ctrl**+**Alt**+**T**, da impostare nelle scorciatoie del DE.

In Linux esiste anche il concetto di *console virtuale*, l'insieme di 7 sessioni utente parallele, 6 con interfaccia a caratteri ed una grafica (quella che utilizziamo normalmente). Si può passare da una all'altra premendo i tasti funzione da **F1** a **F6** con la combinazione **ctrl**+**Alt**+**F1-6**, per le console a caratteri ed il tasto funzione **F7** nella combinazione **ctrl**+**Alt**+**F7**, per la sessione grafica in cui opera il Desktop Environment.

Le 6 console testuali sono ambienti a riga di comando di accesso indipendente al sistema, in cui è necessario eseguire il log-in fornendo un account valido accreditato sul sistema. Potremo così avviare fino a 7 sessioni parallele indipendenti sullo stesso computer e con le stesse credenziali.

Prima di passare alla parte operativa, esamineremo rapidamente alcune particolarità della shell in relazione ai meccanismi di individuazione e ricerca dei file, premettendo i concetti generali del file system.

I file sono organizzati in una struttura ad *albero*. Un nodo dell'albero è chiamato *directory* o *cartella* e può contenere sia file sia ulteriori nodi. Il nodo che contiene tutta la struttura del file system è chiamato *radice*.

Il *percorso* — univoca individuazione di un file — è la sequenza dei nomi delle directory a cominciare dalla radice fino ad arrivare al nome del file. Nella sequenza i nomi devono essere riconoscibili e ciò obbliga a scegliere un carattere speciale con il significato di separatore, che di conseguenza non potrà essere utilizzato nei nomi stessi.

Nei sistemi Unix il separatore è il carattere *slash* (/) che è anche il nome della directory radice, mentre in quelli Windows tale ruolo di separatore è assegnato al carattere *backslash* (\).

2.1 LA DIRECTORY DI LAVORO

Agli argomenti dei comandi di shell, spesso è necessario assegnare file l'unico modo per farlo è digitarne il percorso completo, dalla radice alla directory più profonda dell'albero del file system. Ciò è scomodo anche se si utilizza il completamento automatico della shell attivato dal tasto **Tab**.

Se volessimo ridurre il percorso del file al solo nome, una directory dovrebbe essere implicitamente aggiunta dal sistema. Effettivamente, tale nodo esiste e viene chiamato *directory di lavoro*. Per renderla nota all'utente, il prompt della riga di comando riporta in forma estesa o abbreviata la directory di lavoro.

Una nuova directory di lavoro può essere impostata dando il percorso del nuovo nodo come argomento del comando **cd** *change directory*.

Se per esempio si desidera operare su file contenuti in una particolare directory conviene impostarla come directory di lavoro così da individuare i file semplicemente con il nome e non con il percorso completo.

Ulteriori abbreviazioni facilitano la digitazione dei percorsi di file e directory: il carattere punto (.) rappresenta la directory di lavoro, l'insieme di due punti invece (..) rappresenta la directory immediatamente superiore a quella di lavoro, così che per salire di un livello, ovvero impostare la directory di lavoro al livello superiore di quella attuale, è sufficiente dare il comando `$ cd ..` (il prompt verrà immediatamente aggiornato di conseguenza).

Nei sistemi Unix inoltre il carattere tilde ~ rappresenta la directory home dell'utente in cui sono salvati tutti i file e le impostazioni personali e, come già accennato, il carattere \ rappresenta la directory radice. Nei sistemi Windows la radice è `X:\` dove X è la lettera indicatrice del disco o della periferica di memoria di massa intesa come partizione logica.

A differenza dei sistemi Unix dove l'albero del file system ha coerentemente una sola radice anche se vi sono più partizioni o dispositivi di memoria, nei sistemi Windows esiste un nodo radice per ogni partizione o disco di memoria, tanto che il comando `cd` non modifica la cartella di lavoro se ci si vuole spostare in un altro ramo a meno che non venga specificata l'opzione `/d`.

Se per esempio la directory di lavoro è `C:\Programmi` per modificarla al nuovo valore `D:\Archivio\Documenti\Tesi` possiamo dare il comando:

```
C:\Programmi> cd /d D:\Archivio\Documenti\Tesi
```

oppure possiamo procedere in due passi prima cambiando il ramo del file system a D: e poi dando il comando `cd` semplice:

```
C:\Programmi> D:  
D:\> cd Archivio\Documenti\Tesi
```

2.2 VARIABILI D'AMBIENTE

Le variabili d'ambiente memorizzano informazioni utili all'esecuzione dei comandi di shell ed in particolare percorsi. Lo scopo è quello di rendere un programma indipendente dalla reale posizione dei file su disco.

Il comando per stampare il contenuto di una variabile d'ambiente è `echo`. Sui sistemi tipo Unix la variabile è preceduta dal segno `$`, sui sistemi Windows il nome della variabile deve essere racchiuso tra `%`.

2.3 IL PATH DI SISTEMA

Poiché un programma è fisicamente un file memorizzato su disco, per eseguirlo è necessario che la shell sia in grado di conoscerne il percorso completo nell'albero del file system.

Come per la working directory anche per i programmi esiste un modo semplice di abbreviarne il percorso alla riga di comando ma questa volta, per non obbligare a memorizzare i programmi tutti in un'unica directory, la soluzione assume la forma di un elenco di directory chiamato *search path* e memorizzato nella variabile d'ambiente `PATH`. In definitiva, per eseguire un programma da riga di comando occorre o digitare il percorso completo del file eseguibile oppure occorre che la directory che lo contiene sia compresa nel `PATH`.

Il meccanismo di ricerca dei comandi è il seguente: digitando un nome alla riga di comando, la shell per prima cosa cerca il file corrispondente nel caso fosse un percorso di un file valido e, se non lo trova, lo cerca in tutti i percorsi memorizzati in `PATH` finché o la ricerca ha successo o viene emesso un messaggio di errore.

Nei sistemi Unix esiste il comando `which` che fa esclusivamente questo: ricercare l'eseguibile associato ad un nome. Se la ricerca ha successo viene stampato il percorso completo del programma: per esempio il percorso del programma `pdftex` utilizzato per comporre questo documento è¹:

```
$ which pdftex
/usr/local/texlive/2011/bin/i386-linux/pdftex
```

Esiste anche un terzo luogo dove la shell ricerca i comandi ma questo è vero solo per il sistema operativo Windows: la directory di lavoro. Nei sistemi Unix se la directory di lavoro corrisponde a quella del programma che si desidera eseguire non è sufficiente digitare il nome del file ma occorre dare alla shell l'informazione del percorso completo che tuttavia è immediato

¹Interessante notare che per eseguire il comando `which` la shell effettua la ricerca tra le risorse del `PATH`, infatti si potrebbe dare il comando `$ which which...`

rappresentare con i caratteri punto e slash `./`, seguiti dal nome semplice dell'eseguibile.

Più in generale, quando avviamo la shell vengono creati numerosi parametri che caratterizzano la sessione di lavoro. Tra questi ci sono il `PATH` di sistema e la directory di lavoro².

Per conoscere il `PATH` è sufficiente digitare in console il comando:

```
per Unix like OS    $ echo $PATH
per Windows OS     > echo %PATH%
```

Come modificare il `PATH` dipende fortemente dal sistema operativo e di solito, se l'operazione è necessaria, è effettuata dal programma d'installazione. Nelle prossime sezioni esamineremo in dettaglio la procedura manuale.

2.3.1 MODIFICARE IL `PATH` IN WINDOWS.

Possono essere impostati il `PATH` del singolo utente e quello di sistema. Per motivi di sicurezza è sempre consigliabile limitare le modifiche al solo `PATH` utente lasciando inalterato quello di sistema. Accedendo al sistema con altre credenziali dovremo ripetere la modifica al `PATH` se si vuole rispettare questa buona regola³.

In Windows XP occorre aprire il dialogo di proprietà del sistema con la sequenza di azioni `Start` » `Pannello di controllo` » `Sistema` » `Avanzate`, premere il pulsante `Variabili di ambiente...`, selezionare `PATH` nell'elenco delle variabili utente e modificare la lista premendo su `Modifica...` ed aggiungendo il percorso desiderato facendo attenzione a mantenere il carattere di separazione ; tra un percorso e l'altro. Se nell'elenco non compare la variabile `PATH`, createla impostandone il valore con il percorso alla directory voluta.

In Windows Vista/7 con il mouse fate un clic destro sull'icona Computer sul Desktop e attivate il dialogo `Proprietà` dal menu contestuale, poi dal collegamento presente sulla sinistra della finestra `Impostazioni di sistema avanzate` raggiungete la scheda `Avanzate`, scegliete il pulsante `Variabili di ambiente` e

²Provate a dare il comando `env` nel terminale di un sistema Unix...

³D'altra parte, l'installazione di `TEX Live` di solito viene effettuata con i diritti di amministratore e per tutti gli utenti così che il sistema `TEX` risulta disponibile per qualsiasi utente.

procedete alla modifica del contenuto della variabile `PATH` selezionandola dall'elenco inferiore.

Altro metodo per Windows Vista/7 consiste nel selezionare Computer dal menu **Start**, farci sopra un click destro del mouse e dal menù contestuale scegliere la voce **Proprietà** poi **Impostazioni di sistema avanzate** » **scheda Avanzate** » **Variabili di ambiente**. Il `PATH` compare tra le variabili di sistema, lo si seleziona così da modificarlo con l'uso dei pulsanti inferiori del dialogo, utilizzando correttamente il carattere di separazione `;` tra i percorsi.

2.3.2 MODIFICARE IL PATH IN LINUX.

In Linux la via più semplice ed efficace per aggiungere una directory alla variabile `PATH` è quella di utilizzare il comando `export`:

```
$ export PATH=/usr/local/texlive/2011/bin/i386-linux:${PATH}
```

Il comando precedente sostituisce il `PATH` attuale concatenandolo con il percorso della directory contenente i binari di `TEX Live` (è solo un esempio, `TEX Live` potrebbe essere installata in altra posizione od il percorso da inserire nella variabile potrebbe essere un altro). Il nuovo `PATH` vale solo per la sessione corrente della shell, e per rendere la modifica permanente è sufficiente inserire l'istruzione in un file chiamato `.profile` nella `home` utente, che viene letto nel momento in cui si esegue l'accesso al sistema (oppure ogni volta che si lancia il comando `source`). Possiamo editare il file con un editor oppure più rapidamente provvedere con la riga di comando (digitare il comando tutto in un'unica riga):

```
$ echo 'export PATH=/usr/local/texlive/2011/bin/i386-linux:
      ${PATH}' >> ~/.profile
```

2.3.3 NON MODIFICARE IL PATH IN MAC OS X.

Nei sistema operativo di Apple non è prevista la possibilità di modificare le variabili d'ambiente o le cartelle di sistema anche se ci si può riuscire. Ciò significa che le impostazioni devono essere correttamente eseguite dai programmi d'installazione e non dall'utente che deve sperare che non sia mai necessario compiere tali modifiche manualmente.

Le esercitazioni sono applicazioni pratiche orientate all'utilizzo del sistema \TeX dalla riga di comando, e basate sui concetti di funzionamento della shell. Mettere in pratica i comandi illustrati richiede di conoscere il modo in cui si avvia la shell sul proprio computer (in caso consulta la sezione 1.3 di pagina 3).

3.1 CAMBIARE DIRECTORY DI LAVORO

Su tutti i sistemi il comando per cambiare la directory di lavoro della shell è `cd` che sta per **change directory**:

```
$ cd <path>
```

La sintassi prevede di specificare come primo argomento il percorso della directory. Il comando dato senza argomenti, sui sistemi Unix riporta la directory di lavoro alla **home** dell'utente, su quelli Windows invece, stampa semplicemente la directory di lavoro attuale, per altro informazione già riportata nel prompt (in Unix la stampa della directory corrente si ottiene con il comando `pwd`, **print working directory**).

Nella digitazione dei path è molto comodo avvalersi del *completamento automatico*, che funziona premendo il tasto `Tab` dopo aver digitato un numero sufficiente di caratteri da individuare univocamente una directory, così da non dover più digitarne il nome per intero.

Provare ad impostare la directory di lavoro al valore di `/usr/bin` in Linux e `C:\Document and Settings` in Windows XP o `C:\Users` in Windows 7.

Per risalire al nodo superiore del file system digitare poi il comando `$ cd ..`

3.2 ELENCCARE I FILE

Una delle necessità più frequenti dell'utente è quella di stampare a video l'elenco delle cartelle e dei file presenti in un nodo del file system. In Windows il comando per farlo è `dir`, mentre negli altri sistemi è `ls` (list file). Apriamo una finestra di console e digitiamo il comando senza argomenti per mostrare il contenuto della directory di lavoro.

Windows

> dir

Linux e Mac OS X

\$ ls

Possiamo utilizzare filtri per elencare file corrispondenti a particolari criteri. Il carattere asterisco `*` sta per qualsiasi sequenza di caratteri. Il carattere `?` sta per un qualsiasi singolo carattere. Alla riga di comando proviamo quindi ad elencare i sorgenti `.tex` (questa è l'estensione standard per i nomi dei file sorgenti del sistema `TEX`):

Windows

> dir *.tex

Linux e Mac OS X

\$ ls *.tex

Naturalmente è possibile elencare i file contenuti in qualsiasi directory specificandone il percorso completo indipendentemente dalla directory di lavoro.

Ecco una sessione di lavoro in Linux per elencare i file nella directory principale dei contenuti di questa guida. Per prima cosa si imposta la directory di lavoro e poi si chiede la stampa della lista dei file contenuti poi, rilevata la presenza di una sotto cartella `section`, si chiede l'elenco dei sorgenti `TEX` a sua volta in essa contenuti:

```
$ cd Scrivania/guidaConsole/
$ ls
guidaConsole.aux  guidaConsole.pdf      guidaConsole.toc
guidaConsole.log  guidaConsole.synctex.gz image
guidaConsole.out  guidaconsole.tex      section

$ ls section/*.tex
section/argavanz.tex  section/intro.tex      section/premessa.tex
section/eser.tex      section/notefinali.tex section/shell.tex
```

3.3 SPOSTARE O COPIARE FILE

Per spostare i file tra una directory e l'altra, si utilizza il comando `mv` (**move**), specificando come argomenti prima il percorso del file attuale poi quello di destinazione. In Windows il comando si chiama **move**.

Windows

```
> move <source path> <destination path>
```

Linux e Mac OS X

```
$ mv <source path> <destination path>
```

Se il percorso di destinazione punta alla stessa directory di partenza, il comando `mv` rinomina il file.

Per copiare anziché spostare i file si usa il comando `cp` in Linux e Mac e il comando **copy** in Windows, specificando al solito prima i file da copiare e poi la loro nuova destinazione.

3.4 RINOMINARE FILE

Cambiare nome ad un file è un'operazione concettualmente equivalente allo spostamento perché in entrambe i casi si tratti di modificarne il percorso. Torna utile specie se vi trovate ad utilizzare Windows che solitamente è impostato¹ per nascondere le estensioni dei file, rendendo impossibile cambiarla in modalità grafica.

Il comando da usare è **rename** con la stessa sintassi per tutti i sistemi:

```
$ rename <old_name> <new_name>
```

Per esempio, rinominare il file `uno.tex` — presente nella directory di lavoro —, in `due.tex`, l'istruzione da riga di comando è:

```
$ rename uno.tex due.tex
```

¹È preferibile non farlo ma per cambiare questa opzione cliccare col destro sulla finestra grafica dell'Explorer e dal dialogo delle proprietà togliere la spunta su "Nascondi l'estensione dei file".

3.5 VERIFICARE L'INSTALLAZIONE T_EX

Dopo l'installazione di una distribuzione T_EX è possibile verificare il funzionamento dei programmi con semplici comandi da console. Per prima cosa ci si chiede se il sistema riconosce gli eseguibili dei motori di composizione. In altre parole, il `PATH` di sistema dovrà contenere il percorso degli eseguibili, che appositamente per questo sono contenuti tutti in un'unica directory.

Per esempio, il seguente comando avrà successo se gli eseguibili sono correttamente installati e configurati:

```
$ pdftex -version
```

Compiliamo adesso un sorgente T_EX veramente minimo che contiene poche parole, inserendo con un editor di testi questo codice in un file, naturalmente di estensione `.tex`, che chiameremo `testtex.tex`:

```
Ciao Mondo da \TeX!  
\bye
```

Una volta impostata la directory di lavoro in modo che coincida con quella contenente il sorgente di prova, il comando di compilazione per la verifica di funzionamento è:

```
$ tex testtex
```

Otteniamo un file di uscita con estensione `dvi` *Device Independent*, il formato classico di T_EX che — l'occasione è buona per verificare il funzionamento di alcuni utili programmi compresi nelle distribuzioni e quindi già in dotazione — convertiamo in `pdf` *Portable Document Format* con il tradizionale doppio passaggio, il primo da `dvi` al formato `ps` *Postscript* con il driver `dvips` ed il secondo da *Postscript* a `pdf`:

```
$ dvips testtex      -- conversione dvi --> ps  
$ ps2pdf testtex.ps  -- conversione  ps --> pdf
```

Oggi questi passaggi di formato si utilizzano raramente perché si lavora direttamente e convenientemente con il formato di uscita `pdf`, come faremo tra poco con `pdflatex`.

A questo punto dovrete aver correttamente ottenuto i messaggi di compilazione in console e tutti i file di uscita previsti. Non rimane che

verificare il funzionamento di \LaTeX con questo piccolo sorgente da inserire in un file di testo che chiameremo `testlatex.tex`:

```
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[italian]{babel}

\begin{document}
  Ciao Mondo da \LaTeX!
\end{document}
```

Il comando di compilazione sarà quindi il seguente che produrrà in uscita il file nel formato `pdf`:

```
$ pdflatex testlatex
```

Possiamo completare le verifiche all'installazione provando il corretto funzionamento dello strumento `pdfcrop`. Si tratta di un programma a riga di comando scritto in `perl` che, appoggiandosi a `Ghostscript`, ritaglia un documento `pdf` eliminando i contorni fino a raggiungere le dimensioni minime del contenuto, operazione che facilita l'inclusione nei documenti \LaTeX dei contenuti `pdf`.

Per eseguire una prova sul file `testlatex.pdf` appena creato, impostate in una sessione di terminale la directory corrente a quella contenente il documento da scontornare e date il seguente comando al termine del quale dovrete trovare nella stessa directory del file originale un nuovo file dal nome `testlatex-crop.pdf` correttamente scontornato:

```
$ pdfcrop testlatex.pdf
```

Lo strumento utilissimo `pdfcrop` accetta molte opzioni. Le più importanti sono `-margin <valore>`, con cui si imposta un margine attorno al rettangolo ritagliato esprimendo misure in punti, `-hires` per il calcolo in alta risoluzione del *bounding box* che è appunto il rettangolo minimo che contiene gli oggetti nella pagina, e la possibilità di impostare il nome del file ritagliato semplicemente scrivendolo di seguito al nome del file originale.

Per esempio, per scontornare il file di prova precedente lasciando comunque un margine di 5 punti e sostituendo il file su disco con quello

ritagliato, il comando è il seguente (per comodità utilizzate il completamento automatico premendo il tasto `Tab` per produrre facilmente i nomi dei file):

```
$ pdfcrop --margin 5 testlatex.pdf testlatex.pdf
```

La riga di comando ci ha permesso di verificare l'installazione del sistema $\text{T}_{\text{E}}\text{X}$. Per completezza occorre aggiungere che alcuni shell editor come Kile per Linux od il multi-piattaforma TeXWorks, dispongono di un apposita voce di menù che si occupa di lanciare le compilazioni di verifica della corretta installazione dei programmi della distribuzione $\text{T}_{\text{E}}\text{X}$ informando l'utente sul funzionamento del sistema.

3.6 COMPILARE UN DOCUMENTO SORGENTE

La sintassi generale per compilare un file sorgente con uno dei programmi del sistema $\text{T}_{\text{E}}\text{X}$ è la seguente, dove il nome del file compare senza estensione:

```
$ programma-di-composizione <-opzioni> <nome del file>
```

Un sorgente $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ può essere compilato sia con il programma `latex` sia con `pdflatex`. La differenza è che con il primo si ottiene il documento nel formato `dvi` e con il secondo nel formato `pdf`. Per molte buone ragioni è preferibile utilizzare `pdflatex`:

Windows, Linux e Mac OS X

```
$ pdflatex <nomefile senza estensione>
```

3.6.1 L'OPZIONE `-SHELL-ESCAPE`

Per ragioni di sicurezza del sistema, alcuni pacchetti $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ come `pgfplots` o `gmp` richiedono compilazioni con l'opzione `-shell-escape` per abilitare il compositore all'esecuzione di programmi esterni. Coerentemente alla sintassi generale illustrata alla sezione 1, il comando di compilazione diventa:

```
$ pdflatex -shell-escape <nomefile senza estensione>
```

Questo comando è utile perché a volte si preferisce lanciarlo da riga di comando piuttosto che configurare l'editor per aggiungere nell'elenco

dei comandi di compilazione questa opzione, anche se solitamente è molto più comodo lanciare il comando all'interno dell'editor grafico piuttosto che scomodare la console.

3.7 DOCUMENTARSI CON `texdoc`

Uno dei programmi più utili di una distribuzione \TeX è senza dubbio l'utility `texdoc` che permette di accedere rapidamente alla documentazione di un pacchetto o di un programma di composizione, generalmente in formato pdf.

Per visualizzare la guida di un pacchetto particolare il comando è:

```
$ texdoc <nome pacchetto>
```

Invece, se non si è sicuri del nome del pacchetto o se si cerca documentazione per un dato argomento, si utilizza l'opzione `-l` che sta per *list*, per ottenere l'elenco dei documenti riguardanti una particolare chiave di ricerca:

```
$ texdoc -l chiave
```

Per esempio provate a cercare con `texdoc` la documentazione riguardante Metapost, un linguaggio per il disegno programmato. Per la distribuzione \TeX Live 2012 l'utility troverà i seguenti documenti visualizzabili dando il numero corrispondente — per questo `texdoc` è un programma con un minimo di funzionalità interattive:

```
$ texdoc -l metapost
1 /usr/local/texlive/2012/texmf-dist/doc/metapost/base/mpman.pdf
2 /usr/local/texlive/2012/texmf-dist/doc/latex/pdfslide/mpgraph.pdf
3 /usr/local/texlive/2012/texmf-dist/doc/metapost/base/mpgraph.pdf
4 /usr/local/texlive/2012/texmf-dist/doc/metapost/base/mpintro.pdf
Please enter the number of the file to view, anything else to skip:
```

Oppure, per consultare il manuale del programma `pdflatex` con l'illustrazione della sintassi del comando e di tutte le opzioni possibili, come l'impostazione del nome del file di uscita (`-jobname <name>`), l'impostazione di un formato precompilato per velocizzare la compilazione (`-fmt <format>`), l'impostazione del modo di interazione durante la compilazione (`-interaction <mode>`) e molto altro ancora, digitate:

```
$ texdoc pdflatex
```

3.8 GESTIONE DI T_EX LIVE

La distribuzione T_EX Live scaricabile da CTAN è dotata dell'utility `tlmgr` (T_EX Live manager)². Si tratta di uno script in Perl solitamente utilizzato da riga di comando ma di cui è disponibile anche una versione grafica.

Nei sistemi Mac OS X `tlmgr` è sostituito da un programma ad interfaccia grafica distribuito con MacT_EX, la versione di T_EX Live per i sistemi Apple, chiamata T_EX Live Utility.

Ecco una selezione dei comandi più utili per la gestione della distribuzione con T_EX Live manager (lanciate il comando `$ texdoc tlmgr` per consultare tutte le opzioni disponibili e le corrispondenti sintassi). Alcuni dei comandi potrebbero richiedere la modifica di directory di sistema e pertanto è necessario acquisire i diritti di amministratore:

Aggiornamento di tutti i pacchetti installati:

```
$ tlmgr update --all
```

Elencare i pacchetti non aggiornati:

```
$ tlmgr update --list
```

Aggiornare l'utility `tlmgr` stessa:

```
$ tlmgr update --self
```

Lanciare `tlmgr` in modalità grafica:

```
$ tlmgr gui
```

²Alcune distribuzioni Linux come Debian e quindi Ubuntu, a causa delle politiche di sicurezza e di stabilità del sistema, non distribuiscono nei propri repository T_EX Live con `tlmgr`. Per questa ed altre ragioni conviene installare T_EX Live direttamente dai mirror di CTAN. Leggendo una buona guida come quella di Enrico Gregorio scaricabile all'indirizzo profs.sci.univr.it/~gregorio/texlive-ubuntu.pdf si possono evitare anche i problemi dovute alle dipendenze di alcuni programmi come Kile verso T_EX Live dei repository della distribuzione Linux. Alcune spiegazioni aggiuntive per gli utenti del pinguino si trovano nei post <http://robitex.wordpress.com/2010/09/15/installare-tex-live-2010-in-ubuntu-lucid-lynx/> e <http://robitex.wordpress.com/2010/10/12/kile-e-tex-live-2010-su-sistemi-ubuntu/>.

Consultare lo stato di un pacchetto con alcune informazioni di base:

```
$ tlmgr show nome_pacchetto
```

Impostare un repository CTAN particolare (digitare il comando su un unica riga sostituendo a `<mirror>` il percorso di un server CTAN):

```
$ tlmgr option repository <mirror>/tex/systems/texlive/tlnet/
```

Per esempio un *mirror* piuttosto veloce almeno per la mia connessione è quello di un'università olandese, ed allora il comando diventa (da digitare su una stessa riga):

```
$ tlmgr option repository  
ftp://ftp.snt.utwente.nl/pub/software/tex/systems/texlive/tlnet/
```


In questo capitolo presenterò alcune procedure avanzate che sfruttano la riga di comando per compiere operazioni altrimenti noiose, ripetitive o troppo lunghe da svolgere con altri strumenti. Si tratta di una collezione di casi pratici risolti con l'uso avanzato della riga di comando che si dimostra essere un sistema molto potente.

4.1 ESPRESSIONI REGOLARI

Di solito le chiavi delle *espressioni regolari* sono piuttosto indecifrabili. . . devono infatti rispondere a formati e caratteri di controllo poco amichevoli ma hanno il vantaggio di non richiedere programmazione.

Negli esempi, useremo il formato RE¹ previsto dal linguaggio Perl, poiché il compilatore `perl` è disponibile anche per Windows.

Per disporre di Perl, linguaggio nato appositamente anche per manipolare file di testo, gli utenti Windows possono o eseguire una installazione dal sito www.perl.org oppure, se dispongono di T_EX Live, possono tentare in via sperimentale di utilizzare il compilatore `perl` minimale già compreso nella distribuzione inserendo nel PATH utente il percorso `$tlhome\%year\tpkg\perl\bin` dove `$tlhome` va sostituito con la cartella contenente T_EX Live, solitamente `C:\texlive` ed `%year` va sostituito con il numero di versione che corrisponde all'anno di uscita.

La chiave RE per sostituire un testo con un altro è `'s/<old>/<new>/'` composta dal carattere `s` che sta per *singola linea* ed il carattere slash `/` che fa da separatore. In coda alla chiave possono essere aggiunti *modificatori* per esempio `i` che significa che la ricerca dell'occorrenza non deve tener conto di maiuscolo/minuscolo (*ignore*), e `g` che indica che devono essere

¹RE è l'acronimo di *regular expression*.

trovate tutte le occorrenze nella riga (*global*) e non solo la prima. Per maggiori approfondimenti visitare la pagina web [Perl RE Quick Guide](#) della documentazione Perl.

4.1.1 RINOMINARE FILE

Diversamente dall'implementazione in Windows, in Linux il comando **rename** accetta una chiave nel formato [Perl Regular Expression](#), rendendolo molto flessibile e potente. In Windows rimane la possibilità di utilizzare un comando scritto in Perl che si comporti come il **rename** di Linux anche se questa soluzione richiede un po' di lavoro in più.

Ammettiamo che in una sotto cartella **image** i nomi delle immagini contengano degli spazi. Questo non permette il regolare funzionamento delle macro \LaTeX per l'inclusione dei file. Con un unico comando decidiamo di sostituire tutti gli spazi con il carattere trattino -. Formiamo la chiave RE che inizia con **s** con il carattere da sostituire (lo spazio), il nuovo carattere (il trattino -), ed aggiungiamo il modificatore **g** (global) per sostituire *tutte* le occorrenze:

```
$ rename 's/ /-/g' image/*
```

Ancora per evitare problemi con le immagini in un progetto \LaTeX , vogliamo rendere minuscole le lettere che compongono i nomi dei file compreso quelli dell'estensione. Il modello RE indicherà stavolta una traslitterazione specificando non un unico carattere ma il set di caratteri **A-Z** e quelli corrispondenti da sostituire con il set **a-z**. La manipolazione in corrispondenza di posizione richiede le lettere iniziali **tr**, *transliterate* (che possono essere scritte con il sinonimo **y**):

```
$ rename 'y/A-Z/a-z/' image/*
```

La trasformazione dei caratteri spazio in trattini del primo esempio, può essere ottenuta anche con il modello RE di traslitterazione **'y/ /-/'**.

Potremo voler rinominare i file premettendo ai nomi un prefisso, per esempio **123**, con il modello **'s/^/123/'** oppure, viceversa, potremo voler eliminarlo con il modello **'s/^{\w{3}}//'**, oppure ancora eliminare escludendo l'estensione, l'ultimo carattere del nome, qualsiasi esso sia, con il modello **'s/\w\.\./.'**

4.1.2 CORREZIONE MASSIVA DI SORGENTI

Si tratta del problema della sostituzione di occorrenze in un testo che possiamo risolvere avvalendoci dei comandi di cerca/sostituisci presenti negli editor ma se sono coinvolti molti file è più conveniente l'uso del terminale. La sintassi generale del comando Perl da riga di comando è la seguente:

```
$ perl -p -i -e 's/<RegExpr>/<nuovo>/modificatori' <files>
```

Esaminiamo le tre opzioni iniziali: `-p` indica che dovranno essere esaminate tutte le righe di testo nei file, `-i` indica che saranno modificati direttamente i file originali (opzione *inline*) e `-e` indica che dovrà essere eseguito il codice specificato di seguito (opzione *execute*).

Abbiamo poi l'argomento che specifica il pattern di sostituzione con campi separati da uno slash, nel formato già incontrato per il comando `rename`.

Nell'eseguire questi comandi si deve prestare attenzione verificando la corretta esecuzione. È consigliabile fare una copia di backup dei file perché potrebbe presentarsi la necessità di ripristinare i file originali, e ragionare sulla carta elaborando tutti i casi possibili.

SOSTITUIRE UNA PAROLA

Il nostro revisore ci ha segnalato gentilmente che nella tesi anziché scrivere “perché” abbiamo invece utilizzato l'accento sbagliato scrivendo “perchè”. Dopo aver compreso l'errore² eseguiamo una copia di sicurezza dei sorgenti e lanciamo il comando seguente una volta impostata la working directory a quella che contiene la cartella `chapter` con i file da correggere:

```
$ perl -p -i -e 's/perchè/perché/g' chapter/*.tex
```

Nel testo potrebbero esserci dei “Perchè” ed il nostro comando precedente non li sostituirebbe. Se avessimo messo il modificatore `i` di case-insensitive il termine “Perchè” verrebbe sostituito con “perché” perdendo la maiuscola iniziale. Una prima soluzione è limitarsi alla sostituzione di “erchè”:

²Leggendo per esempio l'articolo “Norme tipografiche per l'italiano in L^AT_EX” di Gustavo Cevolani disponibile all'indirizzo web sul [sito del G_UT_EX](#).

```
$ perl -p -i -e 's/erchê/erché/g' chapter/*.tex
```

Oppure possiamo utilizzare un *gruppo* creato con le parentesi tonde:

```
$ perl -p -i -e 's/(P|p)erchê/$1erché/g' chapter/*.tex
```

In quest'ultimo comando l'espressione regolare `/(P|p)erchê/` comprende un carattere iniziale che può essere `P` oppure `p` grazie al carattere `|`. Poiché la corrispondenza si trova tra parentesi tonde, il carattere trovato viene restituito nella variabile `$1` che andrà a comporre correttamente la parola da sostituire.

Come avrete notato i comandi sono potenti ma occorre fare molta attenzione perché potremo non accorgerci di aver sostituito occorrenze che in realtà non avrebbero dovuto esserlo. Per esempio la parola da sostituire potrebbe essere parte di altre parole più lunghe.

I NUMERI DECIMALI IN AMBIENTE MATEMATICO

Nell'ambiente matematico di \TeX la virgola produce l'aggiunta di un piccolo spazio. Questo si può evitare inserendo la virgola in un gruppo per rappresentare correttamente i numeri decimali: scrivendo `\(v = 12,00 \)` si ottiene $v = 12, 00$ mentre con `\(v = 12\{, \}00 \)` si ottiene $v = 12,00$.

Racchiudere la virgola di separazione decimale in un gruppo, fu la soluzione adottata in un vecchio progetto. Ora invece vogliamo riutilizzare quei file sorgenti risolvendo con il pacchetto *siunitx* ed inserendo i numeri decimali in una macro `\num`.

Il pattern da ricercare consiste in due sequenze di numeri separate da `{,}`, ovvero `\d+{,}\d+` dove il simbolo `\d` significa una singola cifra (e sta per decimal), l'indicatore `+` indica una o più ripetizioni. Il comando completo è dunque:

```
$ perl -p -i -e 's/(\d+){,}(\d+)/\num{${1}.${2}}/g' *.tex
```

SOSTITUIRE UNA PAROLA CON UNA MACRO

In un progetto suddiviso in molti file sorgenti abbiamo digitato molte volte il termine *Metapost* — il programma di grafica vettoriale — ma solo adesso ci siamo accorti che esiste il pacchetto *mflgo* che mette a disposizione la macro `\MP`.

Un comando potrebbe essere:

```
$ perl -p -i -e 's/metapost/\\MP{}/ig' *.tex
```

che sostituirebbe nei sorgenti tutte le occorrenze di “Metapost” oppure “METAPOST” oppure ancora di “metapost” con la macro `\MP{}`.

Qualcosa di più si potrebbe fare inserendo le doppie graffe solo quando necessario per salvaguardare lo spazio che altrimenti `TEX` assorbirebbe, con questi due comandi:

```
$ perl -p -i -e 's/metapost(\\s+|\\w)/\\MP{$1}/ig' *.tex
$ perl -p -i -e 's/metapost(\\W)/\\MP$1/ig' *.tex
```

Il primo pattern individua solamente le occorrenze in cui al termine “metapost” segue uno o più caratteri spazio oppure un carattere alfabetico, il secondo individua solamente le occorrenze in cui al termine “metapost” segue un carattere non alfabetico.

Una tabella delle sostituzioni è la seguente:

...Metapost è un programma	-> ...\\MP{} è un programma...
...MetapostMetapostMetapost<invio>	-> ...\\MP{}\\MP\\MP{}<invio>
...lo sviluppo di Metapost.	-> ...lo sviluppo di \\MP.
\\section{Viva Metapost}	-> \\section{Viva \\MP}

I comandi si possono assemblare in un unico comando separando le espressioni di sostituzione con punti e virgola (il carattere di slash può essere usato per spezzare su più righe un comando molto lungo per renderlo più leggibile):

```
$ perl -pi -e \
's/metapost(\\s+|\\w)/\\MP{$1}/ig; s/metapost(\\W)/\\MP$1/ig' \
*.tex
```

RIGHE MAGICHE

Le righe magiche iniziano con il carattere `%` perciò sono ignorate durante la compilazione, mentre invece sono elaborate da alcuni shell editor come `TeX Works` per impostare la codifica del testo, selezionare il programma di composizione e dichiarare il nome del file sorgente che dovrà essere compilato effettivamente. Ciò risulta utilissimo per lavorare a progetti complessi in cui ci sono più file sorgenti separati in diverse cartelle.

Le righe magiche di questo stesso sorgente sono le seguenti:

```
% !TEX encoding = UTF-8
% !TEX program = pdflatex
% !TEX root = ../guidaconsole.tex
```

L'editor lancerà la compilazione con `pdflatex` non del file attualmente caricato ma del sorgente principale chiamato `guidaconsole.tex` salvato nella cartella superiore (sono presenti infatti i due punti `..`).

Ci proponiamo di modificare il nome del sorgente principale in tutti i file presenti nella cartella `chapter` con un comando `RE`. Se supponiamo come in questo caso che il nome del file contenga solo caratteri alfanumerici il pattern che lo rappresenta è `\w+`. Dovremo inoltre scrivere `\.` al posto del punto semplice `.` e `\/` al posto di `/` perché sono entrambi metacaratteri:

```
perl -p -i -e 's/(\.\.\.|\.)/\w+\.tex/$1\main.tex/' chapter/*.tex
```

Ora, invece di correggere una riga magica già presente nei file, vorremmo che siano aggiunte ad una collezione di sorgenti `.tex` posizionati nella cartella `cap` della nostra tesi.

Poiché il carattere `^` simboleggia l'inizio della linea potremo impartire dalla cartella principale i tre comandi seguenti ciascuno col il compito di inserire una riga magica come prima riga:

```
perl -0777 -pi -e 's/^/% !TEX root = ../tesi.tex\n\n/' cap/*.tex
perl -0777 -pi -e 's/^/% !TEX program = pdflatex\n/' cap/*.tex
perl -0777 -pi -e 's/^/% !TEX encoding = UTF-8\n/' cap/*.tex
```

L'opzione `-0777` fa in modo che l'intero file sia trattato come un'unica linea impostando un carattere di separazione inesistente altrimenti i caratteri di sostituzione verrebbero aggiunti per ogni inizio di riga. Inoltre dobbiamo dare le stringhe di sostituzione nell'ordine inverso a quello desiderato perché le successive si inseriranno in testa — non che l'ordine importi nel caso delle righe magiche.

Nel prossimo capitolo vedremo un altro metodo per inserire le righe magiche all'inizio dei file sorgenti basato sullo scripting di sistema.

In questa sezione ci addentreremo nella programmazione di shell a cui daremo la seguente definizione:

Definizione di *Scripting*

Lo Scripting è la scrittura di codice secondo un linguaggio dinamico che verrà eseguito direttamente da un interprete

Quello che caratterizza lo scripting rispetto all'attività alla riga di comando descritta fino ad ora è dunque la scrittura di un *programma* vero e proprio la cui sintassi dovrà essere conforme ad uno specifico linguaggio che non prevede la dichiarazione preventiva del tipo di dato, che viene invece stabilito *dinamicamente* durante l'esecuzione.

Per esempio, l'assegnazione di un intero e di una stringa è qualcosa di simile al seguente frammento di codice:

```
n = 10           % invece di:  int n = 10
s = "dieci"      % invece di:  string s = "dieci"
```

L'esecuzione del codice viene affidata ad un *interprete* che lo elabora direttamente secondo una procedura anche molto sofisticata che rende davvero minima la differenza dei linguaggi di scripting più moderni dai linguaggi compilati il cui codice invece tradotto in una forma binaria memorizzata in un file oggetto che sarà poi quello eseguito effettivamente. Questa sempre più sfumata distinzione tra linguaggi interpretati e compilati è dovuta al fatto che molto spesso anche i primi traducono il sorgente in una forma binaria intermedia per ambienti di esecuzione detti *macchine virtuali*.

La shell dei più diffusi sistemi operativi offre un linguaggio di scripting predefinito. Non vi è alcuna differenza concettuale se il codice viene digitato

direttamente al prompt dei comandi oppure letto da un file di testo che assume il nome di *script*.

I linguaggi di scripting più conosciuti sono [Python](#), [Lua](#), [Perl](#), [Ruby](#), [Falcon](#), [Javascript](#), eccetera. Quasi sempre si tratta di software libero disponibile per le piattaforme più diffuse tra cui ovviamente i sistemi Windows, Linux ed OS X.

5.1 SHA BANG#!

Abbiamo lasciato in sospeso la risposta ad una domanda che forse è balenata al lettore: visto che esistono molti linguaggi di scripting, come fa la shell al momento dell'esecuzione di uno script a chiamare l'interprete giusto?

Per tutti i sistemi ed in particolare per quelli Windows, l'informazione che lega lo script all'interprete per cui è stato scritto è fornita dall'utente in modo esplicito, ovvero lo script deve essere lanciato da riga di comando dando il nome del file che lo contiene come argomento dell'interprete.

Oppure, può essere sufficiente un doppio click sull'icona del file di script se l'estensione è stata associata al corrispondente eseguibile dell'interprete tramite il riconoscimento automatico degli ambienti ad interfaccia grafica di Linux e Mac od il registro di sistema in Windows.

Sui sistemi di tipo Unix — come Linux ed OS X — la risposta alla domanda iniziale può tuttavia essere molto semplice: inserire il riferimento all'interprete corretto all'interno dello script stesso — generalmente nella prima riga — ed è proprio quello che rappresenta la così detta *sha bang*, formata dalla coppia di caratteri `#!` seguiti dal percorso dell'eseguibile. Quasi tutti i linguaggi di scripting supportano la sha bang nel senso che si tratta di una riga contenente informazioni per la shell che non deve essere interpretata ma perfettamente ignorata come se fosse un commento.

Con questo trucco il nome del file dello script diventa un comando, esattamente come quelli di sistema compilati. Per di più nella maggior parte dei linguaggi è prevista la possibilità di elaborare gli argomenti digitati dall'utente al terminale.

Immaginiamo uno script in linguaggio Lua contenuto nel file `compila` che accetti un argomento. Per eseguirlo possiamo dare il comando:

```
$ lua compila dir % se il sistema non supporta la sha bang
$ compila dir      % se invece si
```


5.2 ESEMPI

Terminata la presentazione dei concetti basilari dello scripting, possiamo scrivere i nostri script personali utili ad automatizzare processi o a produrre vere e proprie applicazioni. L'argomento è vastissimo: possiamo pensare di scrivere semplici sequenze di comandi oppure creare complessi programmi ad interfaccia grafica, scegliendo tra molti ed efficienti linguaggi e librerie.

5.2.1 ANCORA SULLE RIGHE MAGICHE

Supponiamo di aver scritto diverso tempo fa dei sorgenti \LaTeX , quando ancora non conosceamo l'editor TeX Works. In quei file ovviamente non sono presenti le righe magiche che TeX Works riesce ad interpretare impostando codifica e programma di composizione, ed oggi, a distanza di tempo, vorremmo aggiungerle a ciascuno di quei file `.tex`.

Per questo inseriamo in una cartella i sorgenti ed un file che contenga le righe magiche di cui abbiamo bisogno, per esempio queste:

```
% !TEX encoding = UTF-8
% !TEX program = pdflatex
```

e salviamo questo file con il nome `header` (possiamo anche non assegnare nessuna estensione). A questo punto apriamo la shell e scriviamo la riga¹

```
for i in *.tex; do \
  cp header header_temp; \
  cat $i >> header_temp; \
  mv header_temp $i; \
done
```

e vedremo come, premendo il tasto , a tutti i file sarà stato aggiunto quanto scritto nel file `header`.

Si tratta di codice per Bash che risolve in altro modo un problema già incontrato tra i comandi avanzati del capitolo precedente. Il codice è molto semplice: con `cp header header_temp;` si crea una copia del file che contiene le righe magiche; con `cat $i >> header_temp;` uniamo il contenuto del file `.tex` che la variabile `$i` sta scorrendo, con il file

¹Per una questione di praticità in questo documento il codice è stato spezzato su più righe.

`header_temp`; infine con `mv header_temp $i`; rinominiamo `header_temp` con il nome del file `.tex` a cui abbiamo aggiunto le righe sovrascrivendolo.

Per affinamenti successivi si può sicuramente ottenere un codice più efficiente (e questo forse non lo è, ma sicuramente raggiunge il suo scopo), da salvare in un file di estensione `.sh` che possa, per esempio, lavorare su file presenti in sotto-cartelle, oppure verificare che tali righe siano già presenti o meno.

Quanto scritto può essere fatto ovviamente su sistemi operativi Linux e Mac OS X con la shell di sistema; per i sistemi operativi Windows tale procedura è possibile a patto di installare [Cygwin](#) o un altro emulatore Unix.

5.2.2 UPLOAD VIA FTP

Una caso che riguarda proprio le guide tematiche: vorremmo mettere a punto uno script che esegua la connessione sicura al sito del [GfIt](#) e provveda a caricarvi uno o più documenti `pdf` in modo da renderli disponibili per il download.

Operativamente, l'esecuzione di uno script siffatto è molto più immediata che non utilizzare un programma ad interfaccia grafica che supporti il protocollo `ftp` (*file transfert protocol*). Definiamo dapprima la sintassi del comando `curl` a cui ci affidiamo per le transazioni `ftp` e poi creiamo lo script con il linguaggio di shell: Bash.

```
$ curl -u <user>:<password> -T <file> <url destination path>
```

Il programma [curl](#) supporta un notevole numero di protocolli di rete. L'opzione `-u` inserisce le credenziali dell'utente presso il server, l'opzione `-T` indica che desideriamo procedere al caricamento del file verso la destinazione remota. Il formato della destinazione definisce anche il tipo di connessione che desideriamo stabilire che nel caso in esame sarà una connessione `ftp`.

Dopo aver indicato che si tratta di uno script per Bash² per mezzo della Sha Bang, definiamo le variabili per il percorso di destinazione sul server remoto così da facilitare eventuali modifiche e, per mezzo di un ciclo

²Nei moderni sistemi Unix il programma `sh` non è altro che un collegamento al programma `bash`, la Bourne Again Shell divenuta lo standard tra le shell disponibili per i sistemi aperti ed anche per quelli Mac.

esteso a tutti i file **pdf** contenuti nella cartella di lavoro, eseguiamo una volta l'upload dei documenti delle guide.

Di seguito è riportato il codice dello script. Si noti come l'uso delle variabili avviene con una modalità di espansione premettendo il carattere dollaro al nome.

```
#!/bin/sh
guit_ftp='ftp://ftp.guitex.org/'
guit_dir='httpdocs/home/images/doc/GuideGuIT/'
for f in *.pdf
do
    echo "Upload del file :$f"
    curl -T $f $guit_ftp$guit_path$f
    echo "Fatto!"
done
```

Abbiamo trascurato di inserire le informazioni delle credenziali perché immaginiamo che esse siano reperibili automaticamente nell'opportuno formato nel file di configurazione di **curl** di nome **.curlrc** che si trova nella directory **home** dell'utente. Si sono tralasciati i problemi di sicurezza relativi alla conservazione e alla trasmissione sicura delle credenziali a favore della semplicità.

Salvato il codice in un file chiamato **upguide** assegnamogli i diritti di esecuzione con il comando:

```
$ chmod +x upguide
```

A questo punto il suo utilizzo è semplice: basterà avviarlo una volta impostata la cartella di lavoro a quella contenente i documenti da caricare sul sito.

Potremo voler aggiungere allo script l'invio di una e-mail ad un elenco di interessati per comunicare la disponibilità di aggiornamenti...

5.2.3 UNO SCRIPT CON LUA

Come esempio di semplice script in un linguaggio diverso da Bash, consideriamo il codice Lua seguente. Lo scopo è stampare a video alcune informazioni centrate sulla riga di 80 caratteri, considerando la possibilità

che l'utente specifichi quale argomento opzionale una lunghezza di riga diversa. I valori degli argomenti sono disponibili all'interno dello script nella tabella predefinita di nome `arg`:

```
#!/usr/bin/lua
ll = arg[1] or 80

local function prt( s )
    local n = (ll - #s)/2
    local sp = string.rep(' ', n)
    print(sp..s)
end

prt "Titolo: Guida tematica alla riga di comando"
prt "Collana: Le guide tematiche del GuIT"
prt "eccetera..."
```


Al solito, per costruire lo script, salviamo il codice nel file `centermsg` rendendolo eseguibile con `chmod` se siamo in ambiente Unix, e digitiamo alla riga di comando:

```
$ ./centermsg    # in Linux o Mac
> lua centermsg  # in Windows
```




Ricordo che in Lua se una funzione, come in questo caso `prt`, prevede un unico argomento di tipo stringa o tabella, possiamo omettere le parentesi tonde.

NOTE SU QUESTA GUIDA


LICENZA D'USO

Questo documento è rilasciato con licenza  [Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 2.5 Italia License](https://creativecommons.org/licenses/by-nc-sa/2.5/it/).

Tu sei libero di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera e di modificare quest'opera alle seguenti condizioni:

-  **ATTRIBUZIONE:** Devi attribuire la paternità dell'opera nei modi indicati dall'autore o da chi ti ha dato l'opera in licenza e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
-  **NON COMMERCIALE:** Non puoi usare quest'opera per fini commerciali.
-  **CONDIVIDI ALLO STESSO MODO:** Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.

COLOPHON

Questa guida è stata composta con \LaTeX , utilizzando il compositore pdfTeX con la classe `guidatematica` appositamente predisposta dal  e rilasciata sotto la licenza LaTeX Project Public Licence (LPPL) come enunciata in <http://www.latex-project.org/lppl.txt>.

Per presentare *graficamente* procedure ed avvertimenti si è utilizzato il pacchetto `tcolorbox` e per le combinazioni di tasti e simili il pacchetto `menukeys`. I loghi per la licenza Creative Commons sono composti con il pacchetto `cclicenses` di Gianluca Pignalberi.

Dal punto di vista della gestione delle revisioni, i sorgenti della guida sono stati memorizzati con `git` — naturalmente con comandi dalla console — prima verso un repository privato su <http://bitbucket.org> e poi nel nuovo repository del `GUTeX`, così da risolvere allo stesso tempo le problematiche di sicurezza dei dati e quelle relative all'evoluzione nel tempo dei contenuti.

COLLABORAZIONE E RINGRAZIAMENTI

I sorgenti sono stati resi pubblici per uno sviluppo completamente collaborativo ed aperto nel [repository GuTeX su github.com](#). In particolare è la sezione avanzata della guida che potrebbe ricevere i contributi maggiori e più interessanti.

Qualsiasi contributo o suggerimento può essere inviato al mio indirizzo di posta elettronica `giaconet dot mailbox at gmail dot com`, oppure si può clonare il repository e proporre un *merge* con `git`, oppure ancora si può aprire una discussione sul [sito del progetto](#).

Ringrazio i lettori che vorranno migliorare questa guida segnalando errori o contribuendo ad estenderne i contenuti. Intanto l'hanno già fatto ansys, OldClaudio, Elrond e Marco Rocco. Grazie mille!

Un ringraziamento particolare va a Claudio Beccari che ha messo a punto la classe per le guide tematiche condividendone lo sviluppo con noi ed ha revisionato frase per frase le varie versioni di questa guida.

Inoltre ringrazio Agostino De Marco per avermi segnalato il sito web <http://bitbucket.org> che offre repository `git` pubblici ma anche privati (spero che un servizio del genere tornerà utile quando scriveremo insieme il prossimo articolo per `ArSTeX`nica, la rivista del `GUTeX`), Massimiliano Dominici e tutti i [membri del progetto GuTeX](#) per aver creato questo nuovo fantastico spazio *virtuale* di collaborazione e diffusione.