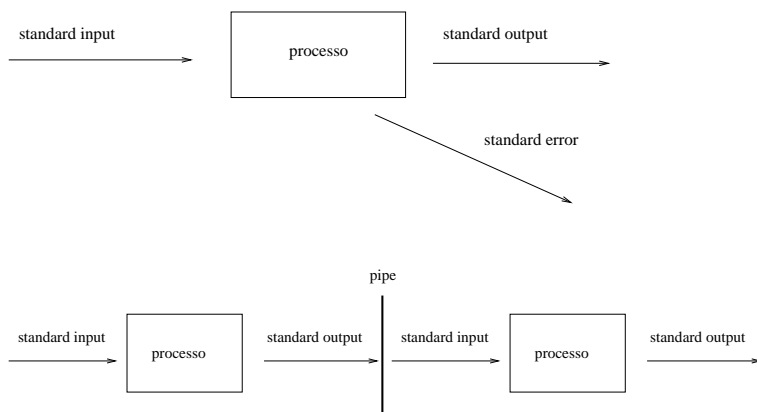


Comandi

<code>cal</code>	Mostra il calendario della settimana. Può essere seguito dall'anno o dal mese e dall'anno
<code>cat file</code>	Scriva il contenuto del file su standard output. Senza argomento legge da standard input
<code>cd nomedir</code>	Ci si sposta nella directory specificata. Senza argomenti riporta nella home. <code>..</code> è il nome della directory padre
<code>cp file destinazione</code>	Copia il file specificato nella destinazione. Se la destinazione è una directory ci crea un file con lo stesso nome di quello copiato, se la destinazione è il nome di un file allora ci scrive sopra
<code>date</code>	Stampa la data corrente. Consultare il manuale per i vari formati di data supportati
<code>echo messaggio</code>	Stampa il messaggio
<code>expr espressione</code>	Stampa il risultato. Controllare nel manuale gli operatori supportati. Esempio <code>expr 3 + 5</code> stampa 8
<code>grep espressione file</code>	Stampa le righe di file che contengono l'espressione. Nel manuale di grep sono documentate le espressioni <i>regolari</i> che accetta
<code>head file</code>	Mostra le prime righe del file. <code>-n numero</code> sceglie il numero di righe da mostrare
<code>ls dir</code>	Mostra il contenuto della directory specificata. Senza argomenti mostra il contenuto della directory corrente
<code>man comando</code>	Visualizza la pagina del manuale relativa al comando. Si esce con <code>q</code> e <code>man man</code> mostra la pagina del manuale relativa al manuale
<code>mkdir nomedir</code>	crea una directory col nome specificato
<code>mv sorgente destinazione</code>	Sposta il file/directory sorgente nel file o nella directory di destinazione
<code>pwd</code>	Stampa la directory corrente
<code>rm file</code>	Rimuove il file. NON c'è il cestino. <code>-i</code> chiede conferma. <code>-r</code> cancella la directory e tutto quello che vi è contenuto
<code>seq inizio fine</code>	Stampa una sequenza di numeri da inizio a fine. Utile nei for
<code>sleep secondi</code>	Attende il tempo specificato
<code>sort file</code>	Ordina le righe di un file. Senza argomento legge da standard input. <code>-k colonna</code> seleziona la chiave di ordinamento
<code>tail file</code>	Mostra le ultime righe del file. <code>-n numero</code> sceglie il numero di righe da mostrare
<code>test espressione</code>	Ritorna 0 se l'espressione è vera. Utile nell' <code>if</code> . es. <code>test -e pippo</code> ritorna 0 se il file pippo esiste. <code>test 3 -ge 2</code> ritorna 0 perchè $3 \geq 2$
<code>uniq</code>	Elimina i duplicati da una sequenza di righe
<code>wc file</code>	Conta il numero di caratteri/parole/linee del file. Senza argomento legge da standard input

Redirezione e piping

comando > file	Redireziona su file lo standard output
comando >> file	Redireziona su file lo standard output, ma non sovrascrive il file se esiste, fa append alla fine del file
'comando'	Sostituisce il comando con il suo risultato. Ad esempio <code>echo "oggi e'" 'date'</code>
comando comando	Collega lo standard output del primo comando allo standard input del secondo
comando ; comando	Esegue i comandi uno dopo l'altro
comando < file	Manda il file sullo standard input del comando



Esempio

```
$ echo "contenuto di pluto" > pluto
$ cat pluto
contenuto di pluto
$ echo "seconda linea" >> pluto
$ cat pluto
contenuto di pluto
seconda linea
$ echo 'date' >> pluto
$ cat pluto
contenuto di pluto
seconda linea
Thu Feb 10 01:04:16 CET 2005
$ echo "conto le parole di pluto" ; cat pluto | wc -w
conto le parole di pluto
11
$ echo "conto le parole delle prime 2 linee di pluto" ; head -n 2 pluto | wc -w
conto le parole delle prime 2 linee di pluto
5
```

Variabili, assegnamento e espansione

NOME_VARIABILE="valore"	Assegna alla variabile il valore. Le virgolette possono essere omesse se non ci sono spazi nel valore. es <code>X=3</code> oppure <code>X="3 3"</code> ma non <code>X=3 3</code>
<code>\$NOMEVARIABILE</code>	Espande la variabile, in pratica ne ritorna il valore. es <code>X=3</code> ; <code>echo X</code> stampa <code>X</code> , mentre <code>X=3</code> ; <code>echo \$X</code> stampa <code>3</code>

Variabili, predefinite

<code>~</code>	Non è una vera e propria variabile, è come scrivere <code>\$HOME</code>
<code>HOME</code>	La directory home dell'utente. es. <code>/home/pippo/</code>
<code>USER</code>	Il nome dell'utente corrente
<code>PATH</code>	Lista di directory in cui cercare i comandi da eseguire. es <code>/usr/bin:~/bin/</code>
<code>PWD</code>	La directory corrente. Quindi il comando <code>pwd</code> non è altro che un <code>echo \$PWD</code>
<code>?</code>	Il valore restituito dall'ultimo comando lanciato. 0 significa successo. es. <code>false</code> ; <code>echo \$?</code> stampa un numero diverso da 0, perchè <code>false</code> è il comando che fallisce sempre. Invece <code>cd</code> ; <code>echo \$?</code> stampa 0 perchè tornare nella propria home non dovrebbe causare errori

Wildcards

<code>*</code>	Espande a tutti i files nella directory corrente. es. Siamo in una directory con files: <code>pippo</code> , <code>pluto</code> , <code>topolino</code> . <code>rm *</code> espande in <code>rm pippo pluto topolino</code> e cancella tutto. <code>rm p*</code> espande a <code>rm pippo pluto</code> (i nomi che iniziano per p).
----------------	---

Esempio

```
$ ls
paperino pluto topolino
$ X=pluto
$ echo $X
pluto
$ rm $X
$ ls
paperino topolino
$ echo $?
0
$ rm file_che_non_esiste
rm: cannot remove 'file_che_non_esiste': No such file or directory
$ echo $?
1
```

Strutture di controllo

<pre>if comando; then comando1 ; else comando2 ; fi</pre>	Esegue comando1 se comando restituisce 0, altrimenti esegue comando 2. es. <code>if test -e pippo; then rm pippo; else echo "spiacente"; fi</code> controlla se pippo è un file che esiste, se c'è lo rimuove, altrimenti stampa spiacente. <code>if false; then echo "impossibile"; fi</code> NON stampa impossibile, perchè false fallisce sempre e quindi restituisce un valore diverso da 0 e quindi non esegue il ramo then (ma farebbe il ramo else che non c'è e quindi non fa nulla).
<pre>for VAR in lst-val ; do comando ; done</pre>	Ripete comando una volta per ogni valore nella lista. es. <code>for X in 1 2 3; do echo \$X; done</code> stampa 1 2 3. <code>for X in `seq 1 30`; do echo \$X; done</code> stampa come prima, ma <code>`seq 1 30`</code> viene sostituito col risultato del comando, quindi 1 2 3 ... 30 e poi viene eseguito il for sulla lista di 30 numeri

Script

<code>#!/bin/bash</code>	È una riga magica, va messa all'inizio dei file di script. In pratica dice che il file è uno script e che è uno script per la shell bash (che è quella che stiamo imparando). È inoltre necessario marcare il file di script come eseguibile col comando <code>chmod a+x nomefile</code>
<code>#</code>	Commento, tutto quello che segue fino alla fine della riga è un commento (viene ignorato). Quindi anche la "riga magica" è un commento (ma è necessario)
<code>read NOME_VAR</code>	Quello che l'utente digita viene messo in NOME_VAR. es. <code>read X; echo \$X</code> è uno script pappagallo, che ripete quello che gli scriviamo.
<code>\$1 ... \$9</code>	Parametri passati allo script

Esercizi

- Dato il file `esempio.txt` creare `esempio_primerighe.txt` che contiene solo le prime 5 righe di `esempio.txt`
- Come prima ma che le contiene ordinate e senza redirezionarlo su un file
- Ordinate secondo la terza colonna
- Senza duplicati
- Quanti ne trova? (non contateli a mano anche se sono pochi)
- Fare uno script `junk.sh` che prende in input un file e lo sposta in `/.junk` (creandola se non c'è).
- Creare un `lsjunk.sh` che stampa i files in `/.junk`
- Creare anche un `unjunk.sh` che prende in input il nome di un file e se c'è lo toglie dal `/.junk` e lo rimette nella directory corrente, se non c'è stampa un errore.
- Modificare i tre script precedenti in modo che controllino se l'utente gli passa un argomento (sugg. basta controllare con `test` se la lunghezza di `$1` è zero, ma occhio a metterlo tra doppi apici)

Soluzioni

- ```
cat esempio.txt | head -n 5 > esempio_primerighe.txt
cat esempio.txt | head -n 5 | sort
cat esempio.txt | head -n 5 | sort -k 3 -n
cat esempio.txt | head -n 5 | sort -k 3 -n | uniq
cat esempio.txt | head -n 5 | sort -k 3 -n | uniq | wc -l
```
- ```
#!/bin/bash

NOME_DEL_CESTINO=$HOME/.junk

if test ! -d $NOME_DEL_CESTINO; then
    mkdir $NOME_DEL_CESTINO;
fi

mv $1 $NOME_DEL_CESTINO
```
- ```
#!/bin/bash

if test -d $HOME/.junk; then
 ls $HOME/.junk
else
 # se non me metto le virgolette ~ viene tradotta in $HOME e
 # e quindi viene stampato qualcosa del tipo
 # /home/pippo/.junk non esiste
 # e mi sembra piu' bellino usare ~ non espansa

 echo "~/junk" non esiste
fi
```
- ```
#!/bin/sh

# ho usato /bin/sh e non /bin/bash perche' ci sono arie shell,
# noi usiamo bash, ma impariamo solo features che fanno parte della
# shell POSIX (standard nei sistemi unix che si chiama solo sh e non
# bash) quindi possiamo dire che il nostro file deve essere
# interpretato da sh (non e' fondamentale, sono una appunta)

if test -e $HOME/.junk/$1; then
    mv $HOME/.junk/$1 .
else
    echo $1 non trovato nel cestino
fi
```