

## Comandi filtro: sed

Il nome del comando `sed` sta per **S**tream **E**Ditor e la sua funzione è quella di permettere di editare il testo passato da un comando ad un altro in una pipeline. Ciò è molto utile perché tale testo non risiede fisicamente in un file su disco e quindi non è editabile con un editor tradizionale (e.g. `vi`). Tuttavia `sed` può anche prendere in input dei file, quindi la sua sintassi è la seguente:

```
sed actions files
```

- Se non si specificano azioni, `sed` stampa sullo standard output le linee in input, lasciandole inalterate.
- Se vi è più di un'azione, esse possono essere specificate sulla riga di comando precedendo ognuna con l'opzione `-e`,  
oppure possono essere lette da un file esterno specificato sulla linea di comando con l'opzione `-f`.
- Se non viene specificato un **indirizzo** o un intervallo di indirizzi di linea su cui eseguire l'azione, quest'ultima viene applicata a tutte le linee in input. Gli indirizzi di linea si possono specificare come **numeri** o **espressioni regolari**.

## Esempi d'uso di sed

- > `sed '4,$d' /etc/passwd`

stampa a video soltanto le prime 3 righe del file `/etc/passwd`: `d` è il comando di cancellazione che elimina dall'output tutte le righe a partire dalla quarta (`$` sta per l'ultima riga del file).

- > `sed 3q /etc/passwd`

stesso effetto del precedente comando: in questo caso `sed` esce dopo aver elaborato la terza riga (`3q`).

- > `sed /sh/y/:0/_%/ /etc/passwd`

sostituisce in tutte le righe che contengono la stringa `sh` il carattere `:` con il carattere `_` ed il carattere `0` con il carattere `%`.

- > `sed '/sh/!y/:0/_%/' /etc/passwd`

sostituisce in tutte le righe che **non** contengono la stringa `sh` il carattere `:` con il carattere `_` ed il carattere `0` con il carattere `%`. Si noti l'uso del quoting per impedire che la shell interpreti il metacarattere `!`.

# Sostituzione del testo con sed

Il formato dell'azione di sostituzione in `sed` è il seguente:

`s/expr/new/flags`

dove:

- `expr` è l'espressione da cercare,
- `new` è la stringa da sostituire al posto di `expr`,
- `flags` è uno degli elementi seguenti:
  - `num`: un numero da 0 a 9 che specifica quale occorrenza di `expr` deve essere sostituita (di default è la prima),
  - `g`: ogni occorrenza di `expr` viene sostituita,
  - `p`: la linea corrente viene stampata sullo standard output nel caso vi sia stata una sostituzione,
  - `w file`: la linea corrente viene accodata nel file `file` nel caso vi sia stata una sostituzione.

## Esempi di sostituzioni con sed

- `sed '/^root/,/^bin/s/:.....:/::/w disabled.txt' /etc/passwd`  
sostituisce la password criptata (lunga 13 caratteri) con la stringa vuota nelle righe in input comprese fra quella che inizia con `root` e quella che inizia con `bin`; tali righe sono poi accodate nel file `disabled.txt`.
- `cat /etc/passwd | sed 's?/bin/. *sh$/usr/local&?'`  
cerca tutte le righe in input in cui compare la stringa corrispondente all'espressione regolare `/bin/. *sh$` (ad esempio `/bin/bash`) e sostituisce quest'ultima con la stringa corrispondente a `/usr/local/bin/. *sh$` (ad esempio `/usr/local/bin/bash`). Si noti che, siccome il carattere separatore di `sed` compare nella stringa da cercare, si è usato il carattere `?` come separatore. Inoltre il carattere `&` viene rimpiazzato automaticamente da `sed` con la stringa cercata (corrispondente a `/bin/. *sh$`).

# Shell Script

Gli **shell script** sono **programmi** interpretati dalla shell, scritti in un linguaggio i cui costrutti atomici sono i **comandi Unix**. I comandi possono essere combinati in sequenza o mediante i costrutti usuali di un linguaggio di programmazione. La sintassi varia da shell a shell. Faremo riferimento alla shell `bash`. Gli shell script sono la base degli *scripting languages*, come *Perl*.

Uno shell script va scritto in un file utilizzando per esempio il comando `cat` o un editor (`vi`, `emacs`, etc). Per poter eseguire lo script, il file deve essere reso **eseguibile**. Lo script viene eseguito invocando il nome del file.

## Esempio

```
> cat >dirsize          # il file dirsize viene editato
ls /usr/bin | wc -w
Ctrl-d                  # fine dell'editing
> chmod 700 dirsize     # dirsize viene reso (in particolare) eseguibile
> dirsize               # viene invocato il comando dirsize
459                     # risultato dell'esecuzione
```

## Esecuzione di script

Lo script viene eseguito in una **sottoshell** della shell corrente.

Il comando `set -v/set -x` fa sì che durante l'esecuzione di uno script la shell visualizzi i comandi nel momento in cui li legge/segue (`set -` annulla l'effetto di `set -v/set -x`). Ciò è utile per il debugging.

```
> cat >data
```

```
set -x
```

```
echo the date today is:
```

```
date
```

```
Ctrl-d
```

```
> chmod u+x data
```

```
> data
```

```
++ echo the date today is:
```

```
the date today is:
```

```
++ date
```

```
Tue Oct 25 17:37:52 CEST 2005
```

```
# contenuto dello script data
```

```
# ...
```

```
# lo script viene invocato
```

```
# ... la shell visualizza
```

```
# i comandi mentre
```

```
# li esegue
```

```
# ...
```

## ... esecuzione di script

```
> cat >sost
```

```
set -v # contenuto dello script sost
```

```
cd TEXT # ...
```

```
ls *.txt
```

```
sed s/'#'/';;;'/ file.txt
```

```
Ctrl-d
```

```
> more TEXT/file.txt
```

```
# questo e' un commento
```

```
# in un programma
```

```
> chmod u+x sost
```

```
> sost # lo script viene invocato
```

```
cd TEXT # ... la shell visualizza
```

```
ls *.txt # i comandi
```

```
file.txt # mentre
```

```
sed s/'#'/';;;'/ file.txt # li legge
```

```
;;; questo e' un commento # output del comando sed
```

```
;;; in un programma # ...
```

# Variabili

Le **variabili** della shell sono stringhe di caratteri a cui è associato un certo spazio in memoria. Il **valore** di una variabile è una **stringa di caratteri**. Le variabili della shell possono essere utilizzate sia sulla linea di comando che negli script.

Non c'è dichiarazione esplicita delle variabili.

**Assegnamento** di una variabile (eventualmente nuova): `variabile=valore`  
(Importante: non lasciare spazi a sinistra ed a destra dell'operatore =)

```
> x=variabile
```

```
> y='y e' una variabile'
```

Per **accedere** al valore di una variabile si utilizza il \$:

```
> echo il valore di x: $x
```

```
il valore di x: variabile
```

```
> echo il valore di y: $y
```

```
il valore di y: y e' una variabile
```

```
> echo y
```

```
y
```

Le variabili sono **locali** alla shell o allo script in cui sono definite. Per rendere globale una variabile (**variabile d'ambiente**) si usa il comando `export`:

```
> export x          # promuove x a variabile di ambiente
```



# Variabili di ambiente

Le **variabili di ambiente** sono variabili globali. Esiste un insieme di variabili di ambiente speciali riconosciute dalla shell e definite al momento del login:

VARIABILE	SIGNIFICATO
PS1	prompt della shell
PS2	secondo prompt della shell; utilizzato per esempio in caso di ridirezione dell'input dalla linea di comando
PWD	pathname assoluto della directory corrente
UID	ID dello user corrente
PATH	lista di pathname di directory in cui la shell cerca i comandi
HOME	pathanme assoluto della home directory

## Esempi d'uso:

```
> echo il valore di PATH: $PATH
```

```
il valore di PATH: /usr/bin:/usr/openwin/bin:/usr/local/bin
```

Le variabili di ambiente possono essere modificate:

```
> PS1='salve: '
```

```
salve:
```

```
> PATH=./bin:$PATH
```

```
> echo il valore di PATH: $PATH
```

```
il valore di PATH: ./bin:/usr/bin:/usr/openwin/bin:/usr/local/bin
```

# Parametri

Le variabili \$1, \$2, ..., \$9 sono variabili speciali associate al primo, secondo, ..., nono parametro passato sulla linea di comando quando viene invocato uno script:

```
> cat >copia
mkdir $1
mv $2 $1/$2
Ctrl-d
```

```
> copia nuovadir testo
> ls nuovadir
testo
```

Se uno script ha più di 9 parametri, si utilizza il comando `shift` per fare lo shift a sinistra dei parametri e poter accedere ai parametri oltre il nono:

```
> cat >stampa_decimo
shift
echo decimo parametro: $9
Ctrl-d
> stampa_decimo 1 2 3 4 5 6 7 8 9 10
decimo parametro: 10
```

# Variabili di stato automatiche (I)

Sono variabili speciali che servono per gestire lo **stato** e sono aggiornate **automaticamente** dalla shell. L'utente può accedervi solo in lettura.

Al termine dell'esecuzione di ogni comando unix, viene restituito un valore di uscita, **exit status**, uguale a 0, se l'esecuzione è terminata con successo, diverso da 0, altrimenti (codice di errore).

La variabile speciale `$?` contiene il valore di uscita dell'ultimo comando eseguito.

```
> cd
> echo $?
0

> copia nuovadir testo
> echo $?
0

> copia nuovadir testo
mkdir: Failed to make directory "nuovadir"; File exists
mv: cannot access testo
> echo $?
2
```

Il comando `exit n`, dove `n` è un numero, usato all'interno di uno script, serve per terminare l'esecuzione e assegnare alla variabile di stato il valore `n`.

## Variabili di stato automatiche (II)

La shell `bash` mette a disposizione numerose variabili di stato; le principali sono:

<b>Variabile</b>	<b>Contenuto</b>
<code>\$?</code>	<i>exit status</i> dell'ultimo comando eseguito dalla shell
<code>\$\$</code>	PID della shell corrente
<code>\$!</code>	il PID dell'ultimo comando eseguito in background
<code>\$-</code>	le opzioni della shell corrente
<code>\$#</code>	numero dei parametri forniti allo script sulla linea di comando
<code>*\$</code> , <code>\$@</code>	lista di tutti i parametri passati allo script sulla linea di comando

In particolare `$$` viene usata per generare nomi di file temporanei che siano unici fra utenti diversi e sessioni di shell diverse, e.g., `/tmp/tmp$$`.

# Login script

Il **login script** è uno script speciale eseguito **automaticamente** al momento del login. In (alcune versioni di) Unix/Linux tale script è contenuto in uno dei file `.bash_profile`, `.bash_login`, `.bashrc`, `.profile`, memorizzati nella home directory degli utenti.

Il login script contiene alcuni comandi che è utile eseguire al momento del login, come la definizione di alcune variabili di ambiente.

Ciascun utente può modificare il proprio login script, ad esempio (ri)definendo variabili di ambiente e alias “permanenti”.

Esiste anche uno script di login *globale* contenuto nel file `/etc/profile` in cui l'amministratore di sistema può memorizzare dei comandi di configurazione che valgano per tutti (tale script è infatti eseguito prima di quelli dei singoli utenti).

Lo script di logout eseguito al momento dell'uscita dalla shell, si chiama solitamente `.bash_logout`.

# Esercizi

- Creare una sottodirectory `bin` all'interno della propria home directory in cui mettere gli script. Fare in modo che gli script contenuti in `bin` possano essere invocati da qualunque directory con il nome del file, senza dover specificare l'intero pathname.

- Qual è l'effetto della seguente sequenza di comandi? Perché?

```
> cat >chdir
cd ..
Ctrl-d
> chmod 700 chdir
> chdir
> pwd
```

- Creare un alias permanente `lo` per il comando `exit`.
- Progettare uno script che prende come parametro una stringa e un file di testo e controlla se la stringa compare nel file.

- Il comando `read` assegna alla variabile speciale `REPLY` un testo acquisito da standard input. Qual è l'effetto dello script `words` contenente i seguenti comandi?

```
echo -n 'Enter some text: '  
read one two restofline  
echo 'The first word was: $one'  
echo 'The second word was: $two'  
echo 'The rest of the line was: $restofline'  
exit 0
```

- Qual è l'effetto della seguente sequenza di comandi? Perché?

```
> cat >data  
echo -n the date today is:  
date  
Ctrl-d  
> chmod 700 data  
> data
```

- Scrivere uno script che estragga soltanto i commenti dal file con estensione `java` fornito come primo argomento, sostituendo `//` con la stringa `linea di commento del file <nome del file>:.` Inoltre i commenti estratti devono essere salvati nel file fornito come secondo argomento.