Bash

Comandi

Comando	Spiegazione
`[[ condizioni ]]`	Conditional Expressions
`read A B ...`	Legge una riga dallo **stdin**. Altro
`while read A B ...`	Per leggere da file in modo comodo. Esempio
`cat file1 file2 ...`	Stampa i file uno dopo l'altro
`grep "substring"`	Cerca la substring dentro lo **stdin** e restituisce solo le righe che la contengono (`grep -- "substring"` se nella substring c'è il `-`). Altro
`wc`	"word count", conta `\n`, **numero di parole** e **bytes** (ovvero caratteri se sono tutti ASCII). Altro
`head`	Scrive le prime 10 (di default) righe dallo STDIN. Altro
`tail`	Scrive le ultime 10 (di default) righe dallo STDIN. Altro
`sort`	Sorta le righe dell'STDIN in ordine crescente alfabeticamente. Altro
`find`	Cerca qualcosa in tutta la gerarchia della cartella in cui è eseguito. Altro

Variabili

Comando	Spiegazione
`variabile=valore`	Dichiara una variabile locale di nome `variabile` e con valore `valore` **Attenzione, niente spazi!**, se vuoi che il valore abbia degli spazi, allora devi farei così `variabile=" valore con spazi "`.

Comando	Spiegazione
`export variabile=valore`	Dichiara una variabile d'ambiente (quindi ereditata dai contesti figli).

Variabili speciali già dichiarate

Variabile	Spiegazione
`$RANDOM`	Variabile speciale che ti da un numero random
`$#`	Ottieni il numero di argomenti passati allo script
`$?`	Ottieni il codice di errore dell'ultima operazione eseguita
`$$`	Ottieni il PID del processo che sta eseguendo lo script corrente
`$!`	Ottieni il PID dell'ultimo processo avviato in background dal processo corrente

Per vederle tutte:

```
MANWIDTH=150 man bash | grep -A 28 -E "^\s*Special Parameters"
```

Scripting

Comando	Spiegazione
`#!/bin/bash`	Prima riga di uno script bash. Indica con quale interprete lo script si aspetta di essere seguito. (`/bin/bash` in questo caso).
`comando1 | comando2`	È la pipe; ficca nello **stdin** della roba di destra l'output della roba di sinistra

Conditional Expressions

Sono quelle che usi in una `[[ condizione ]]`

Vedi la lista completa nel manuale

```
MANWIDTH=150 man bash | grep "CONDITIONAL EXPRESSIONS$" -A 97
```

Parameter Expansions

Substring

`${variabile:indice_inizio:lunghezza}`

Sia `indice_inizio` che `lunghezza` sono una espressione matematica (quelle con le doppie parentesi), quindi ci si possono fare i conti.

Se `indice_inizio` è negativo fa effetto pac-man, quindi l'indice è considerato dalla fine della stringa.

Se `lunghezza` è negativo, la stringa viene presa tutta fino a `lunghezza` caratteri dalla fine

Importante: Se si vuole mettere uno di questi due valori in negativo, bisogna mettere uno spazio dopo i `:`, in quanto esiste un'altra expansion che si fa con `:-` che però non abbiamo trattato.

Esempio

Script:

```
variabile="Ciao sono Andrea"
echo ${variabile: -6: 6}
echo ${variabile: 0: -7}
```

Output:

```
Andrea
Ciao sono
```

Length

Per avere la lunghezza della stringa contenuta in una variabile: `${#variabile}`

Search e Replace

Sintassi	Cosa fa
`${variabile#qualcosa}`	Cerca l'occorrenza più corta di `qualcosa` nel valore di `variabile` partendo dall'inizio e lo toglie. Se gli hashtag sono due (`##`) cerca l'occorrenza più lunga.

Ulteriori spiegazioni

`read`

Può avere dei parametri in cui mette le sottostringhe della riga che ha trovato in base ai separatori definiti in `$IFS` (**spazio**, **\t** e **\n** di default).

Esempio: `echo "Ciao-sono-andrea" | IFS="-" read A B C ; echo $C`
Variabili attese: `$A = Ciao` `$B = sono` `$C = andrea`
Output: `andrea`

`while read`

Per leggere da file in modo comodo

```
while read A B ...;
do
    ...
done < percorso-al-file
```

`WC`

Questo comando senza nessun argomento legge dallo **stdin**.
Passandogli un file come argomento `wc file` fornirà un input del tipo `Numero Nomefile`.
Per estrarre il numero quando si utilizza la modalità col parametro, si utilizzano le operazioni sulle stringhe in questo modo:

```
Appoggio=$(wc -l file)
Appoggio=${Appoggio%% *}
```

Flag utili	Cosa fa
`-c`	Restituisce il numero di **byte**
`-l`	Restituisce il numero di `\n`
`-w`	Restituisce il numero di **parole**

`head`

Stampa le **prime** 10 righe dall'alto

Flag utili	Cosa fa
`-n <numero>`	Stampa le `<numero>` righe dall'alto
`-n -<numero>`	Stampa tutto tranne le **prime** `<numero>` righe **dal basso**

`tail`

Stampa le **ultime** 10 righe (le prime 10 dal basso)

Flag utili	Cosa fa
`-n <numero>`	Stampa le `<numero>` righe dal basso
`-n +<numero>`	Stampa tutto tranne le **ultime** `<numero>` righe (le prime **all'alto**)

`sort`

Flag utilli	Cosa fa
`-r`	Sorta al contrario
`-n`	Compara numericamente

`grep`

Flag utili	Cosa fa
`-v`	Stampa tutto TRANNE le righe che hanno le occorrenze

`find`

Il comportamento del comando usato così senza parametri è quello di stampare il nome di ogni cosa che trova. Il comportamento però varia in base alle flag usate

Flag utili	Cosa fa
`-name <regex>`	Cerca tutti i file/directory con un determinato nome
`-type <tipo>`	Se `tipo` è `f` allora cerca solo file, se `tipo` è `d` cerca solo directory.
`-exec <comando>\;`	Esegue il `comando` per ogni occorrenza che trova nella gerarchia della cartella. Se si vuole usare ogni file trovato come parametro di un comando, ci si riferisce ad esso con `'{}'`.
`-print`	Stampa il percorso relativo al file, utile se usato con `-exec` per capire su quale file il `comando` viene eseguito
`-mindepth <numero>`	La profondità minima, ovvero da quante cartelle in giù deve partire a cercare (1 considera dalla cartella corrente in giù, 2 considera solo il contenuto delle directory della cartella corrente e così via)
`-maxdepth <numero>`	La profondità massima, ovvero quanto deve scendere al massimo di cartella in cartella. (1 si ferma alla cartella corrente, 2 si ferma alle cartelle della cartella corrente e via così)