## ---------- Seti 3/10 ----------

La shell è un'interprete di comandi, spesso usata in maniera interattiva
Alcuni caratteri digitati nella shell vengono **espansi**:

- $var , ${var} -> La shell espande quel nome nel contenuto della variabile
- Le graffe possono essere usate anche per inserire delle stringhe all'interno di altre stringhe (esempio s1{xyz}s2 -> s1xs2 , s1ys2 , s1zs2)
    - **mv pippo pippo.txt = mv pippo{,.txt}**
    - **mv pippo{.txt,} -> pippo**
- Usando la tilde (~) o tilde + nome utente -> ti porta nella home directory (tua o dell'utente di cui si scrive il nome)
- Se io voglio identificare tutti i file che hanno per esempio estensione .c nella directory, faccio "***.c**" : l'asterisco sta per un numero arbitrario di caratteri qualsiasi, quindi nell'esempio "qualsiasi cosa che finisca in .c" (altro esempio **a***, tutti gli argomenti che iniziano con a)
- (espressioni) --> **$((5 + 2)) -> 7** per far fare conti alla shell
- $(cmd) -> l'output di quel comando (cmd) diventa l'argomento di qualcos'altro (esempio lol -> 404, ./arg $(lol) -> 404)

Se io voglio che lo standard input di un comando non sia il terminale ma un file faccio **<nomefile**; se metto **>nomefile** quello che il programma scrive verso lo standard output vada a finire dentro al file; **>>nomefile** apre il file in append, quindi se il file aveva già un contenuto, aggiunge a quello che c'era già.

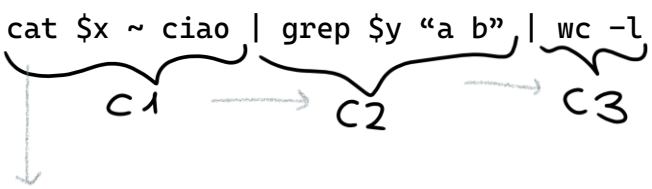
### I comandi possono essere :

- Semplici : una sequenza di parole separate da blank, dove non compare la pipe
- Pipeline : sequenza di comandi, messi "in pipe", ossia separati dalla pipe ->**|**

(esempio pipeline)

Supponiamo che :

wc -> word count




```
c1 > f1
c2 > f2 < f1
c3 < f2
```

(esempio non si fa così)

batcat -lx /home/utente ciao

**Exit status normale :**
Ogni comando (quando non viene ucciso da un segnale, quindi esce normalmente) restituisce un exit status, che finisce in $? e un **codice numerico** :

- **0** -> senza errori
- **non-0** -> errori

**Zero -> True e non-zero -> False (!!!)**
Quando un comando va a buon fine restituisce 0, quindi è l'equivalente di True

&& e || usati per comporre pipelines -> a && b oppure a || b vale la valutazione cosiddetta cortocircuitata, ossia in alcuni casi non serve valutare entrambi gli operandi per sapere i risultati degli and (&&) o degli or (||): se a è andato a buon fine, allora valuto anche b, se non è andato a buon fine non ha senso valutare b

**Exit status in un programma terminato da un segnale :**
Ctrl + C -> termina segnale, invia al comando un segnale SIGINT (2)
Gli altri segnali danno una possibilità al comando di "ignorare" il segnale (può essere catturato), il SIGKILL (9) no.
L'exit status di un processo terminato da un segnale è **128 + S (segnale)**

La Bash è un linguaggio di programmazione, quindi è possibile eseguire cicli, fare degli if, definire funzioni...

**Forkbomb** -> `:(){ :|:& };:`