COMANDI SO

Permessi

user			group			others		
R 4	W 2	X 1	R 4	W 2	X 1	R 4	W 2	X 1

- **ls -l**
  - Elenca i permessi dei file nella directory corrente
- **ls -ld <directory>**
  - Elenca i permessi della <directory>
- **ls -l <directory>**
  - Elenca i permessi dei file dentro la <directory>
- **ls -alh <directory>**
  - Stampa informazioni su tutti i files contenuti nel percorso
- **chown <nuovo-proprietario> <nome-file>**
  - Cambia il proprietario del file
- **chmod <nnn> <nome-file>**
  - Cambia i permessi del file: chmod 764 miofile.txt
- **unset <nome-variabile>**
  - Elimina una variabile
- **source <nome-file>**
  - Esegue il file senza creare subshell
- **history**
  - Visualizza i comandi passati
  - !numero -> esegue il comando con l'indice 'numero'
- **set**
  - Visualizza tutte le variabili della shell, locali e d'ambiente
- **;**
  - Consente di separare diverse operazioni in una line sola
  - echo pippo ; cd Downloads -> prima esegue echo poi cd
- **echo a{bb,cc,ddd}** stampa 'abb acc addd'
  - echo ${A}**{**${B}${C},${C},${A}${B}**}**a se A=bin, B=log, C=boot
  - ottengo binlogboota binboota binbinloga
  - ***echo a{b..k}m*** stampa abm acm adm aem afm agm ahm aim ajm akm
  - ***echo a{4..7}m*** stampa a4m a5m a6m a7m
- **~ (tilde)**
  - Rappresenta la home directory
- Wildcards
  - ***** può essere sostituito da una qualunque sequenza di caratteri, anche vuota
  - **?** può essere sostituito da esattamente un singolo carattere
  - **[elenco]** può essere sostituito da un solo carattere tra quelli in elenco
    - [abk] può essere sostituito da un solo carattere tra a b oppure k.
    - [1-7] può essere sostituito da un solo carattere tra 1 2 3 4 5 6 o 7
- **rm <file>**
  - Elimina il file specificato

- **cat <file>**
  - Visualizza il contenuto del file
- **tail -n <numero> <nome-file>**
  - Stampa in stdout le ultime n righe del file
- **env | more**
  - Visualizza variabili e il loro valore
- **which <file-eseguibile>**
  - Visualizza percorso in cui si trova l'eseguibile
- **mv <percorso-file> <nuovo-percorso>**
  - Sposta il file in una nuova posizione
- **cp <percorso-file> <nuovo-percorso>**
  - Copia il file in una nuova posizione
- **ps aux**
  - Stampa info sui processi in esecuzione
- **du <directory>**
  - Visualizza l'occupazione del disco
- **kill -9 <pid-processo>**
  - Elimina il processo il cui pid è stato passato
- **killall <nome-processo>**
  - Elimina tutti i processi con quel nome
- **bg**
  - Ripristina un job fermato e messo in sottofondo
- **fg**
  - Porta il job più recente in primo piano
- **df**
  - Mostra spazio libero dei filesystem montati
- **touch <file>**
  - Crea il file se non esiste, se no ne aggiorna la data
- **more <file>**
  - Mostra il file specificato un pò alla volta
- **head <file> tail <file>**
  - Mostra le prime o le ultime 10 righe del file
- **man <nome-comando>**
  - Dà informazioni sul comando
- **wget <URL>**
  - Scarica file specificato nell'URL
- **find**
  - Cerca dei file
  - find <percorso> stampa tutti i file a partire da <percorso> ricorsivamente
  - **-maxdepth <numero>** stampa file al massimo sotto di due livelli
  - **-type <lettera>** indica quali tipi di file deve cercare (directory, file …)
    - <lettera>: **f** sta per file, **d** sta per directory
  - **-name <nome>** indica il nome del/dei file da cercare
  - -**iname "*std*"** cerca file dove nome ci sia std ignorando maiusc minusc
  - **-exec** per ciascuno dei file trovati, esegui il comando che segue
    - find usr/ -type f -iname "*std*" -exec head -n 1 **'{}' \;**
    - Per i file trovati con le specifiche, fammi vedere la prima riga dei file

- **grep**
  - Cerca tra le righe di file quelle che contengono alcune parole
  - **grep “errore” log.txt** stampa le righe che contengono la parola errore
  - **grep -l "config" *** stampa tutti i file che contengono la parola config
  - **grep -i "avviso" log.txt** stampa le righe che contengono la parola avviso ignorando le maiuscole o minuscole
  - **grep -r "ciao" ./** stampa le righe che contengono ciao nei file a partire da path attuale e cercando ricorsivamente
- **read <nome-variabile>**
  - Legge input da standard input e lo inserisce nella variabile specificata
  - **read -n** permette di specificare il numero massimo di caratteri che devono essere letti
  - **read -N** permette di specificare il numero esatto di caratteri che devono essere letti
- **wc**
  - Conta il numero di righe, parole e caratteri in un file
- **true** **false**
  - Restituisce exit status 0 e exit status 1
- for ((i=0; i<5; i++)); do echo cane; done | **tee <file1> <file2>** …..
  - stampa cane 5 volte nei file specificati

- **cut -b 1 <file>**
  - -b consente di specificare dei sottoinsiemi di caratteri che devo no essere mandati in output, in questo caso di ”gatto” stampa solo “g”
  - -b 1-4 stampa di “gatto” “gatt”
  - -b 3- stampa dal terzo carattere in poi; -b -3 stampa i caratteri tranne gli ultimi tre
  - -b -3,5,7-9 stampa i primi 3, il quinto, dal settimo al nono carattere
- **sed ‘s/AL/CUF/g’ <file>**
  - Tra il primo / e il secondo /, va messa la stringa da sostituire e fra il secondo e il terzo / va messo quello con cui sostituire. g significa che modifica tutte le occorrenze di AL con CUF
  - sed **-i** significa che oltre a stampare le modifiche, di modificare proprio il file
- **nano <file>**
  - Puoi scrivere all’interno del file-
- **bash**
  - Crea una nuova bash interattiva NON di login
- **export <nome-variabile>**
  - Setta la variabile come d’ambiente
- Variabili d’ambiente che contengono argomenti passati allo script:
  - **$#** il numero di argomenti passati allo script
  - **$0** il nome del processo in esecuzione
  - **$1** primo argomento, **$2** secondo argomento …
  - **$*** tutti gli argomenti passati concatenati e separati da spazi
  - **$@** come $* ma se quotato gli argomenti vengono quotati separatamente

- Operatori **(( ))** e **$(( ))**
  - (( )) valuta i caratteri aritmeticamente
  - ((NUM=3+2)) assegna 5 alla variabile al posto di “3+2”
  - $(( )) quando lo si usa, per prima cosa la bash valuta quello al suo interno poi l’espressione scritta fuori
  - NOMEFILE=pippo$((${NUM}+3))
- Operatore **${!**
  - Funziona più o meno come un puntatore
  - varA=pippo nomevar=varA echo ${!nomevar} stampa a video pippo
- Operatori **||** e **&&**
  - **||** esegue due comandi in sequenza, ma il secondo viene eseguito solo se il primo termina con exit code diverso da 0 (**failure**)
  - **&&** esegue due comandi in sequenza, ma il secondo viene eseguito solo se il primo termina con exit code uguale a 0 (**success**)
- Per estrarre numero da 1 a 5: NUM=$((1+(${RANDOM}%5)))
- OUT=` ./prova.exe `
  - Esegue l’eseguibile e assegna alla variabile OUT l’output prodotto dal programma (backtick, alt+96)

```
NUM=5
echo  "output dello script: "   ` cat ~/$1-$((  ${NUM} +1 )).txt   |  wc -l  `
```

supponiamo di eseguire quello script invocandolo con i seguenti argomenti:
./esempio_command_substitution.sh prova 5

- Tutta la parte dopo “output dello script” è all’interno di backtick, cioè prima esegue quello che c’è dentro, cattura l’output e lo sostituisce a quell’espressione
- wc (wordcount) serve a dire quante righe, parole e caratteri sono contenute in un file. Usare il comando man per sapere cosa fa tipo -l davanti
- cat primo.c | wc, i due comandi vengono lanciati insieme e in output da il numero di righe, parole e caratteri scritti nel file primo.c
- ~ sostituisce il percorso della home directory
- $1 diventa prova
- Poi c’è un - che rimane tale
- $((${NUM} + 1)) diventa 6 poiché NUM=5
- Ossia dopo cat si ha: /home/vic/prova-6.txt
- Quindi si stampa il numero di righe nel file prova-6.txt

- Command Substitution:
  - “ “ Double quote: impedisce la sostituzione di wildcard
  - ‘ ‘ Single quote: Impedisce sostituzione di tutto

Espressioni Condizionali

- Condizioni non possono essere comandi
- Restituiscono 0 per indicare true
- Si utilizza **[[ ]]**, si usano anche ! && || e parentesi tonde per l'ordine
- Se uso **[ ]**, non si possono raggruppare op. con le tonde e si usano -a, -o e ! per and.
- Se uso **test <condizione>**
- SI [[$NUM -lt $(($VAR * (3 + $MUL)))]]
- Per effettuare confronti aritmetici:
  - <arg1> **-eq** <arg2> True se arg1 is equal to arg2
  - <arg1> **-ne** <arg2> True se arg1 is not equal to arg2
  - <arg1> **-lt** <arg2> True se arg1 is less than arg2
  - <arg1> **-le** <arg2> True se arg1 is less or equal to arg2
  - <arg1> **-gt** <arg2> True se arg1 is greater than arg2
- Per effettuare confronti lessicografici tra stringhe:
  - <stringa1> **==** <stringa2> True se le stringhe sono uguali
  - <stringa1> **!=** <stringa2> True se le stringhe sono diverse
  - <stringa1> **<** <stringa2> True se <stringa1> viene prima alfabeticamente
  - <stringa1> **>** <stringa2> True se <stringa1> viene dopo alfabeticamente
  - **-z <stringa>** True se la lunghezza della <stringa> è zero
  - **-n <stringa>** True se la lunghezza della <stringa> è diverso da zero
- Operatori per verificare condizioni su file:
  - **-d <file>** True se il file esiste ed è una directory
  - **-e <file>** True se il file esiste
  - **-f <file>** True se il file esiste ed è un file regolare
  - **-h <file>** True se il file esiste ed è un link
  - **-r <file>** True se il file esiste ed è leggibile
  - **-s <file>** True se il file esiste e non è vuoto
  - **-t <file>** True se il file descriptor fd è aperto e si riferisce a un terminale
  - **-w <file>** True se il file esiste ed è scrivibile
  - **-x <file>** True se il file esiste ed è eseguibile
  - **-O <file>** True se il file esiste ed è posseduto dall'effective user id
  - **-G <file>** True se il file esiste ed è posseduto dall'effective group id
  - **-L <file>** True se il file esiste ed è un link
  - **<file1> -nt <file2>**
    - True se <file1> è più recente del <file2> o se <file1> esiste e <file2> non esiste
  - **<file1> -ot <file2>**
    - True se <file1> è più vecchio del <file2> o se <file2> esiste e <file1> non esiste

- **read <variabile>**
  - Legge da standard input sequenze di caratteri fino a che non si preme Invio
  - Se legge un file, legge una riga fino all'andata a capo
  - Restituisce 0 se non arriva a fine file e ha letto qualcosa
  - Restituisce >0 se arriva a fine file
- **${#**VAR**}**
  - Se VAR=ciao, allora dà in output 4, numero di caratteri del suo valore
- **$?**
  - Indica l'exit status. 0 se andato a buon fine se no un altro numero

File

- Leggere file:
  - while read RIGA ; [[$? == 0]] || [[-n ${RIGA}]] ; do echo "read ${RIGA}" ; done

- Aprire file:

Modo Apertura	**Utente sceglie fd (n è il numero scelto dall'utente)**	**Sistema sceglie fd libero e lo inserisce in variabile**
Solo Lettura	**exec** n**<** PercorsoFile	**exec** {NomeVar}**<** PercorsoFile
Scrittura	**exec** n**>** PercorsoFile	**exec** {NomeVar}**>** PercorsoFile
Aggiunta in coda	**exec** n**>>** PercorsoFile	**exec** {NomeVar}**>>** PercorsoFile
Lettura e Scrittura	**exec** n**<>** PercorsoFile	**exec** {NomeVar}**<>** PercorsoFile

Aggiunta in coda: Apre il file in scrittura senza però eliminare quello che c'era scritto prima
n è il file descriptor (solitamente è 0, 1 e 2 rispettivamente stdin, stdout e 1e)

- Codice per aprire in lettura un file:

```
exec  {FD}<  /home/vittorio/mioinput.txt
while read  -u  ${FD}  StringaLetta ;
do
        echo  "ho letto: ${StringaLetta}"
done
```

- Codice per aprire in scrittura un file:

```
exec {FD}>  /home/vittorio/miooutput.txt
for name in pippo pippa pippi ; do
        echo  "inserisco  ${name}"   1>&${FD}
done
```

1>&${FD} dice che echo non deve scrivere a stdout ma nel file
METTERE SEMPRE UN CONTROLLO CON IF DOPO EXEC
if $? == 0

- **exec {FD}>&- exec n>&-**
  - Chiude un file aperto in qualsiasi modo
- Variabile **$$**
  - Dice il PID della shell corrente
- Variabile **IFS**:
  - Contiene i caratteri che fungono da separatori delle parole negli elenchi
  - IFS=$' \t\n': di default contiene uno spazio bianco, un tab e un newline
  - Per esempio se ho stringhe: ciao,5,domani, posso impostare IFS=$',\n'
- Redirezionamenti:
  - **<** ricevere input da file.
  - **>** mandare std output verso file eliminando il vecchio contenuto del file
  - **>>** mandare std output verso file aggiungendolo al vecchio contenuto del file
  - **|** redirigere output di un programma nell' input di un altro programma
- **programma < nome_file_input**
  - Programma vede il contenuto del file come se venisse digitato da tastiera
- **programma > nome_file**
  - Programma scrive nel file al posto di stdout
  - echo ciao > prova.txt
    - in prova.txt ci sarà scritto ciao
- **programma > nome_file_output < nome_file_input**
  - Scrive ciò che è scritto nel file_input dentro il file_output
- **program &> nome_file_error_and_output**
  - Scrive su un file sia lo stderr che lo stdout
- **program 1>&2**
  - Scrive nello stderr
- **(ls; pwd; whoami) > out.txt**
  - Eseguo quei tre comandi come fosse unico e trovo nel file out.txt le stampe
  - Crea una bash figlia ed esegue quei comandi in sequenza

SLIDE 156 PUO' USCIRE ALL'ESAME

Processi

- **<istruzione> &**
  - Lancia un processo in background. In **$!** troverò il PID
  - **(sleep 10; echo ciao) &** aspetta 10 sec e stampa ciao, ma nel frattempo posso usare la shell, se non usassi &, non posso usare la shell per 10 sec
  - Viene creato un job con un identificatore **[n]**
- **jobs**
  - Elenca i job in esecuzione, i job a sinistra hanno un identificatore **[n]**,a fianco può esserci un + per i più recenti e - per i meno recenti
- **CTRL Z**
  - Sospende un processo in foreground
- **CTRL C**
  - Termina un processo in foreground
- **bg %n**
  - Riprende l'esecuzione in background di un processo sospeso
  - Es. bg %2 mette in background il job 2

- **fg %n**
  - Porta in foreground il processo sospeso con identificatore n
- **kill <pid> kill %n**
  - Elimina il processo specificato dal pid o dall'identificatore del job
  - Es. kill 6152 oppure kill %2
- **disown**
  - Sgancia dalla shell l'ultimo job messo in background
  - **disown -r** sgancia dalla shell tutti i job running
  - **disown -a** sgancia dalla shell tutti i job running e sospesi
  - **disown %jobid** sgancia dalla shell il job specificato

- **nohup <comando arg1 arg2 … argN> &**
  - Uguale a scrivere:
    - <comando arg1 arg2 … argN> &
    - disown
      - sgancio dalla shell l'ultimo processo mandato in background
- **wait**
  - Attende la terminazione dei job/processi in esecuzione
  - Come argomenti può prendere: Elenco di PID, Elenco di jobs, nessuno
  - Es: wait $! wait ${pid1} ${pid2} ${pid3} restituisce exit status di pid3

Manipolazione di Stringhe

- VAR="[13] qualcosa con [o] fine".suffisso=che sta alla fine. prefisso=che sta all'inizio
- ${VAR%%pattern}
  - echo ${VAR%%]*} Rimuovo il piu' lungo suffisso che fa match con stringa orig
  - [13
- ${VAR%pattern}
  - echo ${VAR%]*} Rimuovo il piu' corto suffisso che fa match con stringa orig
  - [13] qualcosa con [o
- ${VAR##pattern}
  - echo ${VAR##*[} Rimuovo il piu' lungo prefisso che fa match con stringa orig
  - o] fine
- ${VAR#pattern}
  - echo ${VAR#*[} Rimuovo il piu' corto prefisso che fa match con stringa orig
  - 13] qualcosa con [o] fine
- ${VAR/pattern/string}
  - Cerca nel contenuto di VAR la sottostringa più lunga che fa match con il pattern specificato (con wildcard) e lo sostituisce con string.
- ${VAR:offset}
  - Sottostringa che parte dal offset-esimo carattere del contenuto di VAR
  - L'offset del primo carattere e' zero
- ${VAR:offset:length}
  - Sottostringa lunga length che parte dal offset-esimo carattere nella var. VAR
  - VAR="babilonia". echo ${VAR:2:5} stampa: bilon