Riassunto comandi SO

Ecco i comandi ordinati per categorie e un ordine logico che privilegia la frequenza e l'importanza d'uso in uno scenario generale:

Navigazione e gestione delle directory:

1. `pwd` : mostra la directory di lavoro corrente.
2. `cd percorso_directory` : cambia la directory di lavoro corrente.
3. `ls -alh percorso` : stampa informazioni su tutti i file contenuti nel percorso.
4. `mkdir percorso_directory` : crea una nuova directory nel percorso specificato.
5. `rmdir percorso_directory` : elimina la directory specificata, se è vuota.

Gestione dei file:

6. `touch percorso_file` : crea il file specificato se non esiste, oppure ne aggiorna la data.
7. `rm percorso_file` : elimina il file specificato.
8. `mv percorso_file percorso_nuovo` : sposta il file specificato in una nuova posizione.
9. `cat percorso_file` : visualizza in output il contenuto del file specificato.
10. `more percorso_file` : mostra il file specificato un poco alla volta.
11. `head percorso_file` : mostra le prime 10 linee del file specificato.
12. `tail percorso_file` : mostra le ultime 10 linee del file specificato.

Visualizzazione e gestione del sistema:

13. `ps aux` : stampa informazioni sui processi in esecuzione.
14. `kill -9 pid_processo` : elimina il processo avente identificativo `pid_processo` .
15. `killall nome_processo` : elimina tutti i processi con quel nome.
16. `df` : mostra lo spazio libero dei filesystem montati.
17. `du percorso_directory` : visualizza l'occupazione del disco.

Lavoro con i job:

- Per sapere l'id (non è il pid) dei job usa il comando jobs

1. `bg %jobID` : ripristina un job fermato e lo mette in sottofondo.
2. `fg` %jobID: porta il job più recente in primo pian

- se non metti il jobID viene considerato l'ultimo job avviato

Variabili e ambiente:

20. `env` : visualizza le variabili ed il loro valore.
21. `which nomefileeseguibile` : visualizza il percorso in cui si trova (solo se nella `PATH`)

l’eseguibile.

Ricerca e analisi:

22. `find (percorso iniziale)` : definisci meglio, guarda -type (f file, d direcotry) e -name (puoi usare le wildcards metti tuto fra virgolette: -name " "), usare -mindepth (numero) e -maxdepth per indicare nel primo caso la profondità delle sottodirectory da dove iniziare, nel secondo caso la massimo profondità dove fermarsi
23. `grep` : cerca tra le righe di file quelle che contengono alcune parole. (usare -c per contare)
24. `wc` : conta il numero di parole o di caratteri di un file.

Strumenti di documentazione e output:

25. `man nomecomando` : mostra il manuale e fornisce informazioni sul comando specificato.
26. `echo sequenza_di_caratteri` : visualizza in output la sequenza di caratteri specificata.
27. `read nomevariabile` : legge input da standard input e lo inserisce nella variabile specificata. fare read -n (numero carattere) numerVar per leggere n caratteri alla volta

Comandi di stato e logica:

28. `true` : restituisce exit status 0 (vero).
29. `false` : restituisce exit status 1 (non vero).

Altri Comandi Usati

30. source file_da_eseguire: fa in modo che la shell che chiama prende le varibili della shell chiamata
31. cat .* per vedere tutti i file nascosti
32. set +o history (disabilita la memoria history) set-o history (la abilita)
33. set -a per creare solo varbili d'ambiente
34. se metto ls nome_file (anche con wildcards) cercherà solo quei file
35. cut -b indici, altri indici volendo gli indici possono essere:
36. -numero: tieni dall'inizio fino a quell'indice
37. numer1-numero2: tieni ciò che sta fra quegli indici, essi compresi
38. nuemr-: tieni dal numero fino alla fine
39. Le stringhe iniziano con indice 1
40. Esempio cut -b -1,3-4
41. La varaibile RANDOM, si chiama con $RANDOM e da un valore casuale è possibile settare un seed con RANDOM=seed, in base al seed la varibile assumerà determinati valori
42. se voglio che wc -w(conta le parole e dai il numero di un file) dia come output solo il numero bisogna metterlo in pipe con un cat del file: cat file | wc -w , (usare wc -l per contare le linee), (wc -m per contare i caratteri)
43. $$ variabile che da il pid del processo
44. Il comando sed: per mettere delle variabili usare " " al posto di ' '
45. Per lanciare uno script in background fare ./nomeScript &

46. usare set -x all'inizio del file per avere del debugging
47. exit valore per finire impostando la variavile $? al valore
48. uniq per rendere le righe uguali un unica riga
    - `-c` **(count)**: Conta quante volte ogni linea appare nell'input e riporta il conteggio accanto a ciascuna linea.
      Esempio: `uniq -c file.txt`
    - `-d` **(duplicate)**: Mostra solo le linee duplicate (quelle che appaiono più di una volta).
      Esempio: `uniq -d file.txt`
    - `-u` **(unique)**: Mostra solo le linee uniche (quelle che appaiono una sola volta).
      Esempio: `uniq -u file.txt`
    - `-i` **(ignore case)**: Ignora la distinzione tra maiuscole e minuscole durante il confronto.
      Esempio: `uniq -i file.txt`
    - `-f N` **(skip fields)**: Salta i primi `N` campi (parole separate da spazi o tab) quando confronta le linee.
      Esempio: `uniq -f 2 file.txt` (ignora i primi 2 campi nelle righe).
    - `-s N` **(skip characters)**: Salta i primi `N` caratteri di ogni riga durante il confronto.
      Esempio: `uniq -s 3 file.txt` (ignora i primi 3 caratteri).
    - `-w N` **(compare N characters)**: Confronta solo i primi `N` caratteri di ogni riga.
      Esempio: `uniq -w 5 file.txt` (confronta solo i primi 5 caratteri).

-exec nel find

L'opzione `-exec` del comando `find` in Bash permette di eseguire un comando su ogni file o directory che corrisponde ai criteri di ricerca specificati. È molto utile per automatizzare operazioni come spostare, eliminare, modificare o analizzare file trovati.
`find [percorso] [criteri] -exec [comando] '{}' \;`

- `[percorso]`: il percorso della directory in cui cercare (ad esempio, `.` per la directory corrente).
- `[criteri]`: i criteri di ricerca (es. `-name`, `-type`, `-size`, ecc.).
- `-exec [comando]`: il comando da eseguire per ogni file o directory trovato.
- `{}`: un segnaposto che rappresenta ogni file trovato.
- `\;`: indica la fine del comando (necessario).

1. Sostituzione di una stringa (operazione base)

```
sed 's/old_text/new_text/' file.txt
```

Questo comando sostituisce la prima occorrenza di `old_text` con `new_text` su ogni riga del file `file.txt`.

2. Sostituzione globale di tutte le occorrenze su ogni riga

```
sed 's/old_text/new_text/g' file.txt
```

Aggiungendo il flag `g` alla fine del comando, tutte le occorrenze di `old_text` vengono sostituite con `new_text` in ogni riga del file.

3. Sostituire una stringa solo sulla riga specificata

```
sed '3s/old_text/new_text/' file.txt
```

Questo comando sostituisce `old_text` con `new_text` solo sulla **terza** riga di `file.txt`.

4. Eliminare una riga specifica

```
sed '3d' file.txt
```

Questo comando elimina la **terza** riga del file `file.txt`.

5. Eliminare tutte le righe vuote

```
sed '/^$/d' file.txt
```

Questo comando rimuove tutte le righe vuote dal file `file.txt`. La regex `^$` corrisponde a righe che non contengono caratteri.

6. Aggiungere una riga dopo una riga specifica

```
sed '2a New line of text' file.txt
```

Aggiunge "New line of text" **dopo** la seconda riga di `file.txt`. Nota che il comando `a` inserisce il testo **dopo** la riga indicata.

7. Inserire una riga prima di una riga specificata

```
sed '2i New line of text' file.txt
```

Aggiunge "New line of text" **prima** della seconda riga di `file.txt`. Il comando `i` inserisce una riga **prima** della riga specificata.

8. Modifica in-place (sostituisce direttamente il file)

```
sed -i 's/old_text/new_text/g' file.txt
```

Aggiungendo l'opzione `-i`, il comando modifica il file direttamente (in-place), senza dover usare un file di output.

9. Uso di espressioni regolari

```
sed 's/[0-9]\{3\}/XXX/' file.txt
```

In questo esempio, `sed` sostituisce qualsiasi sequenza di **tre cifre consecutive** con "XXX". Le parentesi graffe `{3}` sono usate per specificare quante volte deve comparire un pattern.

10. Utilizzare `sed` su input diretto (senza file)

```
echo "Hello World" | sed 's/World/Universe/'
```

In questo caso, il comando `sed` sostituisce "World" con "Universe" nella stringa passata come input attraverso il pipe (`|`).

11. Filtrare righe che corrispondono a un pattern

```
sed -n '/pattern/p' file.txt
```

Con l'opzione `-n`, `sed` non stampa automaticamente ogni riga. Il comando `p` indica di stampare solo le righe che corrispondono al pattern specificato.

12. Sostituire in più righe con un file di input (pattern multiplo)

```
sed -e 's/old_text/new_text/' -e 's/another_old_text/another_new_text/' file.txt
```

Sintassi di base

```
sort [opzioni] [file]
```

Se non viene specificato un file, `sort` legge dall'input standard.

Funzionamento e opzioni principali

1. **Ordinamento alfanumerico predefinito**
   - Ordina le righe alfabeticamente (A-Z) o numericamente se contengono solo numeri.

   ```
   sort file.txt
   ```

2. **Ordinamento inverso**
   - Usa l'opzione `-r` per ordinare in ordine decrescente.

```
sort -r file.txt
```

3. **Ordinamento numerico**
   - L'opzione `-n` ordina i numeri correttamente (es. 2, 10, 100 invece di 10, 100, 2).

```
sort -n file.txt
```

4. **Ordinamento per colonna specifica**
   - Usa l'opzione `-k` per ordinare in base a una determinata colonna.

```
sort -k 2 file.txt
```

   Questo ordina le righe in base alla seconda colonna.
5. **Ignorare la differenza tra maiuscole e minuscole**
   - Usa l'opzione `-f` per ignorare le maiuscole.

```
sort -f file.txt
```

6. **Eliminare le righe duplicate**
   - Usa l'opzione `-u` per rimuovere le righe duplicate.

```
sort -u file.txt
```