Sistemi Operativi Laboratorio

Sicurezza dei Sistemi e delle Reti Informatiche
Anno Accademico 2024/2025

Nicola Iantomasi
nicola.iantomasi2020@gmail.com

Abstract

Questo documento tratta gli argomenti presenti negli esami del Prof Matteo Re', fornendo un approfondimento pratico per la preparazione al parziale. Il testo è suddiviso in più sezioni, tra cui un laboratorio sulle memorie di massa, che copre il funzionamento dei dischi magnetici, il calcolo dei blocchi, la latenza rotazionale e l'ottimizzazione dei tempi di accesso ai dati. Viene inoltre affrontata la programmazione a basso livello con esempi di system call in Assembly. Un'altra sezione è dedicata alla gestione dei file e delle directory in ambienti Unix/Linux, con dettagli su comandi della shell, permessi, processi e strumenti di manipolazione di file come grep, awk, cut, sort, e sed. Infine, il documento include esercizi pratici e simulazioni d'esame per consolidare la comprensione degli argomenti trattati. Il materiale è pensato per studenti UniMi che frequentano il corso di Sicurezza dei Sistemi e delle Reti Informatiche.

Contents

Laboratorio

Memorie di massa

Ci sono due tipi principali di memoria secondaria:

- **Direct Access Storage Devices (DASDs)**
  - Dischi magnetici:
    - Hard Disks
    - Floppy Disks
  - Dischi ottici: CD-ROM
- Serial Devices: nastri magnetici.

Unità di misura spaziali:

- byte: 8 bits
- **kilobyte (KB)**: 1024 or 2^{10} bytes
- **megabyte (MB)**: 1024 kilobytes or 2^{20} bytes
- **gigabyte (GB)**: 1024 megabytes or 2^{30} bytes

Unità di misura temporali:

- **nanosecond (ns)**: one- billionth (10-9) of a second.
- **microsecond (s)**: one- millionth (10-6) of a second.
- **millisecond (ms)**: one- thousandth (10-3) of a second.

Dischi magnetici

I dati, rappresentati da bit (0 o 1), vengono scritti su piatti circolari ricoperti di materiale ferromagnetico, chiamati **dischi**. Questi ruotano continuamente ad alta velocità. Le **testine** di lettura/scrittura registrano o leggono i dati quando i **piatti** passano sotto di esse. Nei disk drive dei PC sono presenti più piatti per aumentare la capacità di archiviazione. Il disco contiene **tracce** concentriche che sono divise in **settori**. Il **blocco** è l'unità indirizzabile più piccola di un disco. Un **cilindro** è un set di tracce posizionate ad un determinato raggio del disco (inteso come insieme di piatti).

Il disco ha un controllare che ha una cache che viene utilizzata da buffer per le richieste di lettura/scrittura.

Quando un programma legge un byte dal disco l'OS localizza la sua posizione sulla superficie (traccia/settore) e legge l'intero blocco in una speciale area di memoria che funziona da buffer. Il collo di bottiglia nell'accesso al disco è il movimento dei bracci delle testine. Ha quindi senso immagazzinare il file in tracce che occupano la medesima posizione su diversi piatti e superfici piuttosto che su diverse tracce della superficie di un singolo piatto.

Calcolo dei blocchi

- $\text{blocchiPerSuperficie} = \text{cilindriPerDisco} \times \text{settoriPerDisco}$
- $\text{blocchiPerDisco} = \text{blocchiPerSuperficie} \times \text{TestinePerDisco}$
- $\text{blocchiPerDisco} = \text{cilindriPerDisco} \times \text{settoriPerDisco} \times \text{TestinePerDisco}$
- $\text{blocchi} = \text{cilindri} \times \text{testine} \times \text{settori}$
- $\text{Offset} = C \times (H \times S) + H \times S + S$

Notazione

(C,H,S)

Esempio: (0, 0, 2)

- C = 0 → Cilindro (traccia) 0
- H = 0 → Testina (piatto) 0
- S = 2 → Settore 2

Esercizio calcolo dei blocchi

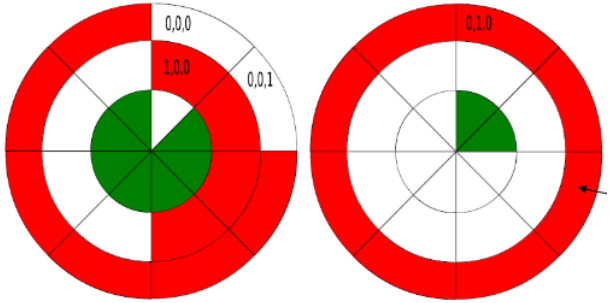
Dati:

- 3 cilindri (C)
- 2 Testine (T)
- 8 Settori (S)

Calcoli:
$\text{blocchi} = 3 \times 2 \times 8 = 48$ blocchi

Esercizio calcolo dell'Offset

Si considera la posizione iniziale (0,0,2) e la posizione finale (1,0,3).



1. Calcolo della posizione finale (1,0,3):

$$\begin{aligned} &1 \times (2 \times 8) + 0 \times 8 + 3 \times 1 \\ &= 1 \times 16 + 0 + 3 \\ &= 16 + 3 = 19 \end{aligned}$$

2. Calcolo della posizione iniziale (0,0,2)

$$\begin{aligned} &0 \times (2 \times 8) + 0 \times 8 + 2 \times 1 \\ &= 0 + 0 + 2 = 2 \end{aligned}$$

3. Differenza tra le due posizioni:

$$19 - 2 = 17$$

Tempo di lettura/scrittura

- $\text{TempoDiLettura} = \text{TempoDIRotazione} + \text{TempoDiRicerca} + \text{TempoDiAccesso}$
- $\text{RateDiLettura} = \frac{\text{DimensioneTrasferimento}}{\text{TempoDiLettura}}$

Algoritmi per ottimizzare il tempo di ricerca

- **First Come First Served (FCFS)**: Gli accessi ai blocchi del disco vengono serviti nell'ordine in cui arrivano, senza alcuna ottimizzazione. È semplice, ma può causare lunghi tempi di attesa se le richieste sono distribuite in modo casuale (effetto «starvation»).
- **Shortest Seek First (SSTF)**: L'algoritmo sceglie sempre la richiesta più vicina alla posizione attuale della testina, riducendo il tempo di ricerca medio. Tuttavia, può causare starvation per richieste lontane se il disco è molto occupato.
- **Scan / Look (Elevator)**: La testina si muove in una direzione servendo tutte le richieste fino a raggiungere il limite (Scan) o l'ultima richiesta esistente (Look), poi inverte la direzione. Questo metodo assicura un tempo di risposta più equo rispetto a SSTF e riduce la frammentazione delle richieste.

Stima capacità del disco

- $\text{CapacitàTraccia} = \text{SettoriPerTraccia} \times \text{BytePerSettore}$
- $\text{CapacitàCilindro} = \text{CapacitàTraccia} \times \text{TraccePerCilindro}$
- $\text{TraccePerCilindro} = \text{TraccePerSuperficie} \times \text{NumeroDiTestine}$
- $\text{CapacitàDisco} = \text{CapacitàCilindro} \times \text{CilindriPerDisco}$
- $\text{CilindriPerDisco} = \text{TracciePerSuperficie}$

Esercizio capacità disco

Salvare un file di 20000 record su un disco con le seguenti caratteristiche:

- 512 bytes per sector
- 40 sector per track
- 11 tracks per cylinder
- 1331 cylinder

1. Quanti cilindri sono richiesti se ogni record occupa 256 bytes?
2. Qual è la capacità totale del disco?

Calcolo dei settori necessari: Ogni settore può contenere: $\frac{512}{256} = 2$ record per settore Quindi, il numero totale di settori necessari per memorizzare 20.000 record è: $\frac{20000}{2} = 10.000$ settori

1. Calcolo dei cilindri richiesti:

   Ogni cilindro ha: $40 \times 11 = 440$ settori per cilindro

   Quindi, il numero di cilindri necessari è:

   $\frac{10000}{440} =\sim 22.73$

   Poiché un cilindro parziale non è possibile, servono 23 cilindri.

2. Capacità totale del disco

   Calcoliamo la capacità complessiva del disco considerando tutti i cilindri:

   $1331 \times 440 \times 512 = 299.499.520$ bytes

   Convertiamolo in Gigabyte (GB):

   $299.499.\frac{520}{1024\times1024\times1024}\} =\sim 0.28$ GB Capacità totale del disco ≈ 0.28 GB (circa 280 MB).

Risultati Finali:

- Cilindri richiesti: 23
- Capacità totale del disco: ∼280 MB

Se vuoi altri dettagli, fammi sapere! 😊

Esercizio

- Supponiamo di voler leggere consecutivamente i settori di una traccia ordinati dal primo all'ultimo : sectors 1, 2,...11.
- Supponiamo che due settori consecutivi non possano essere letti in assenza di interlacciamento.

Quante rivoluzioni sono necessarie per leggere l'intero disco?

- Senza interlacciamento
- Con interlacciamento 3:1

Nota: Al giorno d'oggi molti controllori dei dischi sono veloci e quindi l'interlacciamento non è più così commune.

Tempo di ricerca (seek Time)

Tempo di ricerca: tempo richiesto per muovere il braccio della testina sul cilindro desiderato. È la componente che incide di più sul tempo di accesso. Tipicamente:

- 5 ms per muoversi da una traccia alla successive (track-to-track).
- 50 ms tempo di ricerca massimo (per spostarsi al di fuori di una traccia quando ci si trova la suo interno).
- 30 ms tempo di ricerca medio (da una qualsiasi traccia ad una qualsiasi altra traccia).

Tempo di ricerca medio (s)

Il tempo di ricerca dipende unicamente dalla velocità con cui si muovono i bracci delle testine e dal numero di tracce che devono essere attraversate per raggiungere l'obiettivo. Data la conoscenza delle seguenti informazioni (che sono costanti per ogni specifico modello di disco):

- Hs = tempo richiesto perché la testina inizi a muoversi.
- Ht = tempo richiesto perché la testina si muova da una traccia alla successiva.

Il tempo necessario perché la testina si muova di n tracce è:

$$\text{Seek}(n) = Hs + \text{Ht} \times n$$

Latenza rotazionale (r)

Latenza è il tempo richiesto necessario perché il disco ruoti in modo che il settore che ci interessa sia sotto la testina di lettura/scrittura. Gli hard disk ruotano a circa 5000-7000 rpm (12-8 msec per rivoluzione).

Note:
- Latenza minima = 0.
- Latenza massima = tempo per una intera rivoluzione del disco.
- Latenza media$(r) = \frac{\min + \max}{2} = \frac{\max}{2} =$ tempo $\frac{1}{2}$ rivoluzione del disco

Circa 5000 - 7000 RPM, 12/8 ms per rivoluzione RPM/60 = RPS Latenza Massima = $\frac{1}{\text{RPS}} = \frac{\text{sec}}{\text{rotazione}}$ Latenza Media = $\frac{\text{Latenza Massima}}{2}$

Tempo di trasferimento

Il tempo di trasferimento è il tempo richiesto perché una testina passi attraverso un blocco.

$$\text{Tempo di trasferimento} = \frac{\text{settori da trasferire}}{\text{settori in una traccia}} \times \text{tempo di rotazione}$$

Il tempo di trasferimento dipende unicamente dalla velocità a cui ruotano i piatti e dal numero di settori che deve essere trasferito.

St = numero totale settori per traccia.

È possibile calcolare il tempo di trasferimento per n settori contigui sulla stessa traccia come segue:

$$\text{Tempo trasferimento} = \left(\frac{n}{\text{St}}\right) \times \left(\frac{1000}{R}\right)$$

Esercizio latenza, capacità e tempo di lettura

Dati:
- 20 superfici
- 800 tracce/superficie
- 25 settori/traccia
- 512 bytes/settore
- 3600 rpm (revolutions per minute)
- 7 ms track-to-track seek time
- 28 ms avg seek time
- 50 ms max seek time

Calcolare:
- Latenza media
- Capacità del disco
- Tempo richiesto per leggere l'intero disco, un cilindro alla volta

Calcoli:
- Latenza Media = $\frac{\text{Latenza Massima}}{2} = \frac{60 \times \frac{1000}{3600}}{2} = 8.33$ ms.
- Capacità disco = $20 \times 800 \times 25 \times 512 = 193.31$ MB
- Tempo di lettura dell'intero disco = $20 \times 800 \times 7 = 112$ sec

Esercizi con le memorie di massa

- Dimensione blocchi: 512 byte
- Dimensione puntatori: 24 bit (equivalenti a 3 byte)
- Inode:
  - 5 blocchi diretti
  - 1 blocco indiretto semplice
  - 1 blocco indiretto doppio
- Primo blocco: Ha indice logico 0 1.

Numero di puntatori in un blocco indiretto

Ogni puntatore occupa 3 byte. Un blocco è di 512 byte. Quindi, il numero di puntatori in un blocco indiretto è: $\frac{512}{3} = 170$

Indirizzo logico del primo e dell'ultimo blocco con indirizzamento indiretto semplice

- Blocchi diretti: I primi 5 blocchi (indici: 0, 1, 2, 3, 4).
- Primo blocco indiretto semplice: Inizia dal blocco successivo, quindi indice 5.
- Ultimo blocco indiretto semplice: Può indirizzare 170 blocchi.
  - Ultimo blocco = 5 + 170 - 1 = 174

Indirizzo logico del primo e dell'ultimo blocco con indirizzamento indiretto doppio

- Primo blocco indiretto doppio: Inizia dopo l'ultimo blocco dell'indiretto semplice. Primo blocco = 174 + 1 = 175
- Ultimo blocco indiretto doppio: Ogni blocco di indirizzamento doppio contiene 170 puntatori e ciascun puntatore indirizza un blocco con 170 indirizzi. Numero massimo di blocchi = 170 x 170 = 28900 Ultimo blocco = 175 + 28900 - 1 = 29074

Numero di blocchi che compongono un file di 130500 byte

Ogni blocco è di 512 byte. Numero di blocchi $\frac{130500}{512} = 255$

In quale blocco fisico si trova un dato byte?

Tabella guida dei puntatori:

Puntatore	0 (D)	1 (D)	2 (D)	3 (D)	4 (D)	5 (IS)	6 (ID)
Valore	100	101	102	120	121	300	301

Tabella guida dei contenuti dei puntatori parziali:
Blocco 300:

Indice elemento	0	1	2	3	4	5
Valore	301	305	306	307	308	309

Blocco 301:

Indice elemento	0	1	2	3	4	5
Valore	800	801	802	850	851	852

Blocco 800:

Indice elemento	0	1	2	3	4	5
Valore	1200	1201	1202	1203	1204	1205

Byte 1980:
$\frac{1980}{512} = 3.87$
Il byte si trova nel quarto blocco fisico (indice 3).

Byte 3023:
$\frac{3023}{512} = 5.9$
Il byte si trova nel sesto blocco fisico, che è indirizzato dall'indirizzo indiretto semplice. Essendo 0 - 4 diretti, e 5 il nostro puntatore "Jumper" che sappiamo ci lancia al blocco, la sesta casella sarà l'indice 304.

Byte 92151:
$\frac{92151}{512} = 179.98$
Il byte si trova nell'indirizzamento indiretto doppio. Andiamo sulla prima "Jumper", che ci porta al blocco 301, ma è un ID (doppio), quindi saltiamo un altra volta il prima possibile finendo nella 800. Partendo dal primo blocco dell'indiretto doppio (indice 175, cosa che sappiamo da calcoli dell'es 2), avendo che 179 - 175 = 4, otteniamo 1024.

Assambly

%eax	Nome	%ebx	%ecx	%edx
1	sys_exit	int	-	-
2	sys_fork	struct pt_regs	-	-
3	sys_read	unsigned int	char *	size_t
4	sys_write	unsigned int	const char *	size_t
5	sys_open	const char *	int	int
6	sys_close	unsigned int	-	-

Tabella 1: Tabella delle principali syscall.

sys_exit

Registro	Descrizione
%ebx	Codice di uscita del processo

sys_fork

Registro	Descrizione
%ebx	Struttura dei registri del processo

sys_read

Registro	Descrizione
%ebx	File descriptor
%ecx	Buffer di lettura
%edx	Numero di byte da leggere

sys_write

Registro	Descrizione
%ebx	File descriptor
%ecx	Buffer da scrivere
%edx	Numero di byte da scrivere

sys_open

Registro	Descrizione
%ebx	Nome del file
%ecx	Modalità di apertura (flag)
%edx	Permessi (se necessario)

sys_close

Registro	Descrizione
%ebx	File descriptor da chiudere

Shell

Directories

Navigation

```bash
pwd                     # Print current directory path
ls                      # List directories
ls -a|--all             # List directories including hidden
ls -l                   # List directories in long form
ls -l -h|--human-readable # List directories in long form with human readable sizes
ls -t                   # List directories by modification time, newest first
stat foo.txt            # List size, created and modified timestamps for a file
stat foo                # List size, created and modified timestamps for a directory
tree                    # List directory and file tree
tree -a                 # List directory and file tree including hidden
tree -d                 # List directory tree
cd foo                  # Go to foo sub-directory
cd                      # Go to home directory
cd ~                    # Go to home directory
cd -                    # Go to last directory
pushd foo               # Go to foo sub-directory and add previous directory to stack
popd                    # Go back to directory in stack saved by `pushd`
```

2. Create directory

```bash
mkdir foo                   # Create a directory
mkdir foo bar               # Create multiple directories
mkdir -p|--parents foo/bar  # Create nested directory
mkdir -p|--parents {foo,bar}/baz # Create multiple nested directories

mktemp -d|--directory       # Create a temporary directory
```

3. Moving Directories

```bash
cp -R|--recursive foo bar                       # Copy directory
cp file.txt /path/to/destination/          # Copy `file.txt` nella directory di destinazione
cp file.txt newfile.txt                    # Crea una copia di `file.txt` con un nuovo nome
cp -i file.txt /path/to/destination/       # Chiede conferma prima di sovrascrivere file esistenti
cp -f file.txt /path/to/destination/       # Sovrascrive file esistenti senza chiedere conferma
cp -n file.txt /path/to/destination/       # Non sovrascrive mai file esistenti
cp -u file.txt /path/to/destination/       # Copia solo se il file sorgente è più recente o non esis
cp -v file.txt /path/to/destination/       # Mostra ogni file mentre viene copiato
cp -r dir/ /path/to/destination/           # Copia una directory e tutto il suo contenuto ricorsivam

# Esempi pratici
cp *.txt /path/to/destination/             # Copia tutti i file `.txt` nella directory di destinazio
cp -r dir1/ /path/to/destination/          # Copia ricorsivamente `dir1` e tutti i file e subdirecto
cp -i *.txt /path/to/destination/          # Copia tutti i `.txt` chiedendo conferma per eventuali
duplicati

mv foo bar                                      # Move directory
mv file.txt /path/to/destination/          # Sposta `file.txt` nella directory specificata
mv file.txt newname.txt                    # Rinomina `file.txt` a `newname.txt`
mv dir1 dir2                               # Rinomina `dir1` in `dir2`
mv -i file.txt /path/to/destination/       # Chiede conferma se il file esiste già nella destinazion
mv -f file.txt /path/to/destination/       # Sovrascrive il file di destinazione senza conferma
mv -n file.txt /path/to/destination/       # Non sovrascrive mai file esistenti nella destinazione
mv -u file.txt /path/to/destination/       # Muove solo se il file sorgente è più recente di quello
di destinazione

# Esempi pratici
mv *.txt /path/to/destination/             # Sposta tutti i file `.txt` nella directory specificata
mv -i dir1/* /path/to/destination/         # Sposta tutti i contenuti di `dir1` nella destinazione
chiedendo conferma per i duplicati
```

4. Deleting Directories

```bash
rmdir foo                       # Delete non-empty directory
rm -r|--recursive foo           # Delete directory including contents
rm -r|--recursive -f|--force foo # Delete directory including contents, ignore nonexistent files and
never prompt
rm file.txt                                # Rimuove un singolo file
rm -i file.txt                             # Chiede conferma prima di rimuovere
rm -f file.txt                             # Forza la rimozione senza chiedere conferma (utile per
file con permessi di sola lettura)
rm -r dir/                                 # Rimuove una directory e tutto il suo contenuto ricorsiv
rm -rf dir/                                # Rimuove forzatamente una directory e tutto il suo
contenuto ricorsivamente
rm -v file.txt dir/                        # Mostra i file e le directory mentre vengono rimossi
rm *.txt                                   # Rimuove tutti i file con estensione `.txt` nella
directory corrente

# Esempi pratici
rm -i *.txt                                # Rimuove tutti i file `.txt` chiedendo conferma per cias
rm -rf /path/to/dir                        # Rimuove forzatamente la directory specificata e tutto
il contenuto
```

Files

1. Creating Files

```bash
touch foo.txt           # Create file or update existing files modified timestamp
touch foo.txt bar.txt   # Create multiple files
touch {foo,bar}.txt     # Create multiple files
touch test{1..3}        # Create test1, test2 and test3 files
touch test{a..c}        # Create testa, testb and testc files

mktemp                  # Create a temporary file
```

2. Standard Output, Standard Error and Standard Input

```bash
echo "foo" > bar.txt        # Overwrite file with content
echo "foo" >> bar.txt       # Append to file with content

```

```bash
ls exists 1> stdout.txt     # Redirect the standard output to a file
ls noexist 2> stderror.txt  # Redirect the standard error output to a file
ls 2>&1 out.txt             # Redirect standard output and error to a file
ls > /dev/null              # Discard standard output and error
ls -lah                     # -l (Use a long listing format)
                            # -a (List all entries including those starting with a dot)
                            # -h (Print sizes in human readable format)

read foo                    # Read from standard input and write to the variable foo
```

3. Moving Files

```bash
cp foo.txt bar.txt                          # Copy file
mv foo.txt bar.txt                          # Move file

cat file1 file2                             # Copy text files
cat file1 file2 > newcombinedfile           # Combinate text files
cat < file1 > file2                         # Copy file1 to file2
```

4. Deleting Files

```bash
rm foo.txt           # Delete file
rm -f|--force foo.txt # Delete file, ignore nonexistent files and never prompt
```

5. Reading Files

```bash
cat foo.txt          # Print all contents
less foo.txt         # Print some contents at a time (g - go to top of file, SHIFT+g, go to bottom of file, /foo to search for 'foo')
head foo.txt         # Print top 10 lines of file
tail foo.txt         # Print bottom 10 lines of file
open foo.txt         # Open file in the default editor
wc foo.txt           # List number of lines words and characters in the file
```

6. File Permissions

#	Permission	rwx	Binary
7	read, write and execute	rwx	111
6	read and write	rw-	110
5	read and execute	r-x	101
4	read only	r–	100
3	write and execute	-wx	011
2	write only	-w-	010
1	execute only	–x	001
0	none	—	000

For a directory, execute means you can enter a directory.

User	Group	Others	Description
6	4	4	User can read and write, everyone else can read (Default file permissions)
7	5	5	User can read, write and execute, everyone else can read and execute (Default directory permissions)

- u - User
- g - Group
- o - Others
- a - All of the above

```bash
ls -l /foo.sh           # List file permissions
chmod +100 foo.sh       # Add 1 to the user permission
chmod -100 foo.sh       # Subtract 1 from the user permission
chmod u+x foo.sh        # Give the user execute permission
chmod g+x foo.sh        # Give the group execute permission
chmod u-x,g-x foo.sh    # Take away the user and group execute permission
chmod u+x,g+x,o+x foo.sh # Give everybody execute permission
chmod a+x foo.sh        # Give everybody execute permission
chmod +x foo.sh         # Give everybody execute permission
chown [user][:nameoffile] nameoffile      # Edit proprietario o il gruppo di un file/directory.
chgrp group nameoffile  # Permette di modificare il gruppo di un file/directory
```

7. Finding Files

Find binary files for a command.

```bash
type wget                           # Find the binary
which wget                          # Find the binary
whereis wget                        # Find the binary, source, and manual page files
```

`locate` uses an index and is fast.

```bash
updatedb                            # Update the index

locate foo.txt                      # Find a file
locate --ignore-case                # Find a file and ignore case
locate f*.txt                       # Find a text file starting with 'f'
```

`find` doesn't use an index and is slow.

```bash
find /path -name foo.txt                 # Find a file
find /path -iname foo.txt                # Find a file with case insensitive search
find /path -name "*.txt"                 # Find all text files
find /path -name foo.txt -delete         # Find a file and delete it
find /path -name "*.png" -exec pngquant {} # Find all .png files and execute pngquant on it
find /path -type f -name foo.txt         # Find a file
find /path -type d -name foo             # Find a directory
find /path -type l -name foo.txt         # Find a symbolic link
find /path -type f -mtime +30            # Find files that haven't been modified in 30 days
```

```bash
find /path -type f -mtime +30 -delete      # Delete files that haven't been modified in 30 days
```

8. Find in Files

```bash
grep 'foo' /bar.txt                        # Search for 'foo' in file 'bar.txt'
grep 'foo' /bar -r|--recursive             # Search for 'foo' in directory 'bar'
grep 'foo' /bar -R|--dereference-recursive # Search for 'foo' in directory 'bar' and follow symbolic links
grep 'foo' /bar -l|--files-with-matches    # Show only files that match
grep 'foo' /bar -L|--files-without-match   # Show only files that don't match
grep 'Foo' /bar -i|--ignore-case           # Case insensitive search
grep 'foo' /bar -x|--line-regexp           # Match the entire line
grep 'foo' /bar -C|--context 1             # Add N line of context above and below each search resu
grep 'foo' /bar -v|--invert-match          # Show only lines that don't match
grep 'foo' /bar -c|--count                 # Count the number lines that match
grep 'foo' /bar -n|--line-number           # Add line numbers
grep 'foo' /bar --colour                   # Add colour to output
grep 'foo\|bar' /baz -R                    # Search for 'foo' or 'bar' in directory 'baz'
grep --extended-regexp|-E 'foo|bar' /baz -R # Use regular expressions
egrep 'foo|bar' /baz -R                    # Use regular expressions
grep '^s.[aeiou]'                          # Show all lines that have a vowel as the third letter and start with s
grep "pattern" file.txt                    # Cerca "pattern" in file.txt e stampa le righe corrispon
grep -i "pattern" file.txt                 # Ricerca case-insensitive (ignora maiuscole/minuscole)
grep -v "pattern" file.txt                 # Mostra le righe che *non* contengono "pattern"
grep -r "pattern" /path/to/dir/            # Ricerca ricorsiva in tutte le directory e file
grep -l "pattern" *.txt                    # Elenca solo i nomi dei file che contengono il pattern
grep -c "pattern" file.txt                 # Conta le righe che contengono il pattern
grep -n "pattern" file.txt                 # Mostra le righe con numero di riga
grep -H "pattern" file.txt                 # Mostra il nome del file nelle corrispondenze (utile per più file)
grep -o "pattern" file.txt                 # Mostra solo le porzioni che corrispondono al pattern

# Regex avanzate
grep "^pattern" file.txt                   # Cerca righe che iniziano con "pattern"
grep "pattern$" file.txt                   # Cerca righe che terminano con "pattern"
grep "[0-9]\{3\}" file.txt                 # Cerca tre cifre consecutive (usando sintassi POSIX)
grep -E "(one|two)" file.txt               # Cerca "one" o "two" (regex estesa)

# Esempi pratici
grep -i "error" /var/log/syslog            # Cerca "error" nei log di sistema, ignorando maiuscole/minuscole
grep -r "TODO" ~/projects/                 # Cerca "TODO" in modo ricorsivo nella directory `project
grep -c "^#" script.sh                     # Conta le righe che iniziano con `#` (commenti)
```

9. Replace in Files

```bash
sed 's/fox/bear/g' foo.txt                 # Replace fox with bear in foo.txt and output to consc
sed 's/fox/bear/gi' foo.txt                # Replace fox (case insensitive) with bear in foo.txt and output to console
sed 's/red fox/blue bear/g' foo.txt        # Replace red with blue and fox with bear in foo.txt and output to console
sed 's/fox/bear/g' foo.txt > bar.txt       # Replace fox with bear in foo.txt and save in bar.txt
sed 's/fox/bear/g' foo.txt -i|--in-place   # Replace fox with bear and overwrite foo.txt

sed 's/fox/bear/g' foo.txt                 # Replace "fox" with "bear" in foo.txt and output to consol
sed 's/fox/bear/gi' foo.txt                # Replace "fox" (case insensitive) with "bear" in foo.txt and output to console
sed 's/red fox/blue bear/g' foo.txt        # Replace "red fox" with "blue bear" in foo.txt and output to console
sed 's/fox/bear/g' foo.txt > bar.txt       # Replace "fox" with "bear" in foo.txt and save output to bar.txt
sed -i 's/fox/bear/g' foo.txt              # Replace "fox" with "bear" and overwrite foo.txt

# Caratteri speciali
sed 's/a./X/g' foo.txt                     # Replace "a" followed by any character with "X"
sed 's/fo*/X/g' foo.txt                    # Replace "f" followed by zero or more "o"s with "X"
sed 's/f[ao]x/bear/g' foo.txt              # Replace "fax" or "fox" with "bear"
sed 's/[aeiou]/X/g' foo.txt                # Replace any vowel with "X"

# Classi e posizioni specifiche
sed 's/fox$/bear/g' foo.txt                # Replace "fox" only if it's at the end of the line
sed 's/^fox/bear/g' foo.txt                # Replace "fox" only if it's at the beginning of the line
sed 's/[^0-9]/X/g' foo.txt                 # Replace any non-digit character with "X"
sed 's/[A-Z]/X/g' foo.txt                  # Replace any uppercase letter with "X"

# Quantificatori
sed -E 's/a{3}/X/g' foo.txt                # Replace exactly "aaa" with "X"
sed -E 's/a{2,}/X/g' foo.txt               # Replace "aa", "aaa", etc., with "X"
sed -E 's/a{2,4}/X/g' foo.txt              # Replace "aa", "aaa", or "aaaa" with "X"

# Metacaratteri utili
sed -E 's/\w/X/g' foo.txt                  # Replace any alphanumeric character with "X"
sed -E 's/\W/X/g' foo.txt                  # Replace any non-alphanumeric character with "X"
sed -E 's/\s/X/g' foo.txt                  # Replace any whitespace with "X"
sed -E 's/\S/X/g' foo.txt                  # Replace any non-whitespace character with "X"

# Gruppi di cattura e riferimenti
sed -E 's/(fox)/bear/g' foo.txt            # Replace "fox" with "bear" using capture group
sed -E 's/(fox)(bear)/\2\1/g' foo.txt      # Swap "fox" and "bear" in matching patterns
sed -E 's/(f)(o)(x)/\3\2\1/g' foo.txt      # Reverse characters in "fox" to "xof"

# Sostituzioni condizionali
sed '/pattern/s/fox/bear/' foo.txt         # Replace "fox" with "bear" only in lines containing "patte
sed '5s/fox/bear/' foo.txt                 # Replace "fox" with "bear" only in the 5th line
sed '5,10s/fox/bear/' foo.txt              # Replace "fox" with "bear" only from lines 5 to 10

# Aggiungere testo alla fine o all'inizio
sed 's/$/ END/' foo.txt                    # Append "END" at the end of each line
sed 's/^/START /' foo.txt                  # Add "START" at the beginning of each line

# Sostituire e duplicare con gruppi
sed -E 's/(word)/\1\1/g' foo.txt           # Duplicate "word" into "wordword"
sed -E 's/(fox) (bear)/\2 \1/g' foo.txt    # Replace "fox bear" with "bear fox"
```

```
53
54 # Eliminare righe e parole
55 sed '/pattern/d' foo.txt                # Delete lines that contain "pattern"
56 sed 's/[0-9]//g' foo.txt                # Remove all digits
57
58 # Modifiche In-Place con backup
59 sed -i.bak 's/fox/bear/g' foo.txt       # Replace "fox" with "bear" and create a backup file foo.tx
```

Simbolo	Significato
`.`	Corrisponde a qualsiasi singolo carattere
`*`	Corrisponde a zero o più occorrenze del carattere precedente
`^`	Inizio della riga
`$`	Fine della riga
`[]`	Corrisponde a qualsiasi carattere all'interno delle parentesi quadre
`[^]`	Corrisponde a qualsiasi carattere non presente nelle parentesi quadre
`\`	Escapes il carattere successivo
`{n}`	Corrisponde esattamente a `n` occorrenze del carattere precedente
`{n,}`	Corrisponde a `n` o più occorrenze del carattere precedente
`{n,m}`	Corrisponde tra `n` e `m` occorrenze del carattere precedente
`()`	Gruppi di cattura per le espressioni regolari, consentendo di trattare porzioni di testo come un'unica entità
`` ` ``	Indica un'espressione regolare letterale in alcune shell, utilizzata per racchiudere comandi o espressioni da eseguire (tuttavia, non è direttamente un metacarattere in `sed`)
`?`	Corrisponde a zero o una occorrenza del carattere precedente
`+`	Corrisponde a una o più occorrenze del carattere precedente (richiede l'uso di `-E` o `\+` in `sed`)
`\b`	Corrisponde a un confine di parola (inizio o fine di una parola)
`\B`	Corrisponde a un punto che non è un confine di parola

Redirection Input/Ouput

```bash
1 command > nameoffile                  # Redirect output on file
2 command >> nameoffile                 # Redirect output on file with append
3 command < nameoffile                  # Body of file passed to command (on stdin)
4 command1 | command2                   # stdout (command1) -> stdin (command 2)
5 # Redirezioni di Output (Output Standard - STDOUT)
6
7 echo "Hello World" > output.txt            # Scrive "Hello World" in output.txt, sovrascrivendo il
  contenuto del file
8 echo "Hello Again" >> output.txt          # Aggiunge "Hello Again" alla fine di output.txt
```

```
9
10 # Redirezioni di Input (Input Standard - STDIN)
11
12 sort < input.txt                        # Usa input.txt come input per il comando sort
13 wc -l < input.txt                       # Conta le righe di input.txt e mostra il risultato
14
15 # Redirezione dell'Errore Standard (STDERR)
16
17 ls /nonexistent 2> error.txt           # Redirige solo l'errore (stderr) in error.txt
18 ls /nonexistent > output.txt 2> error.txt  # Redirige stdout in output.txt e stderr in error.txt
19 ls /nonexistent &> all_output.txt          # Redirige sia stdout che stderr in all_output.txt
   (shorthand per `> output.txt 2>&1`)
20
21 # Redirezioni Complesse con `2>&1`
22
23 ls /nonexistent > output.txt 2>&1           # Redirige stderr (2) allo stesso file di stdout
   (1), output.txt
24 echo "Text" > output.txt 2>&1           # Scrive sia stdout che stderr in output.txt
25 command > output.txt 2>&1 | tee log.txt   # Redirige stdout e stderr in output.txt e contemporaneament
   su log.txt
26
27 # Append Output e Errore
28
29 echo "Another line" >> output.txt 2>> error.txt  # Aggiunge stdout a output.txt e stderr a error.txt
30 ls /nonexistent &>> all_output.txt          # Aggiunge sia stdout che stderr in all_output.txt
31
32 # /dev/null - Ignorare Output o Errori
33
34 command > /dev/null                      # Scarta l'output standard (stdout)
35 command 2> /dev/null                     # Scarta l'errore standard (stderr)
36 command &> /dev/null                     # Scarta sia stdout che stderr
37
38 # Pipe e Subshell con Redirezioni
39
40 (echo "stdout"; echo "stderr" >&2) > output.txt 2> error.txt  # Redirige separatamente stdout e stderr
   in subshell
41 { echo "stdout"; echo "stderr" >&2; } > all_output.txt 2>&1   # Usa blocco con {} per redirigere
   stdout e stderr in un unico file
42
43 # `tee` per Redirezione Multipla e Logging
44
45 echo "Logging info" | tee output.txt                  # Mostra l'output e lo scrive in output.tx
46 echo "Append log" | tee -a output.txt                 # Mostra l'output e lo aggiunge a output.t
47 command | tee output.txt | grep "pattern"              # Mostra, salva in output.txt e filtra
   con grep
48
49 # Here Document (Input Multilinea)
50
51 cat << EOF > file.txt
52 Prima linea
53 Seconda linea
54 EOF                                          # Scrive il testo in file.txt
```

```

# Here Document con Esecuzione Comando

cat << EOF | grep "pattern"                              # Passa il contenuto come input a grep
Prima linea
Seconda linea con pattern
EOF

# Here String (Passare Stringa come Input a un Comando)

grep "pattern" <<< "Test string"                         # Cerca "pattern" in "Test string"
awk '{print $1}' <<< "campo1 campo2 campo3"              # Stampa il primo campo di una stringa

# Redirezioni in File Descriptor Personalizzati

exec 3> custom_output.txt                                # Apre il file descriptor 3 per la scrittura su custom_output.txt
echo "Test" >&3                                          # Scrive "Test" su custom_output.txt tramite fd 3
exec 3>&-                                                # Chiude il file descriptor 3

exec 4< input.txt                                        # Apre il file descriptor 4 per la lettura da input.txt
read line <&4                                            # Legge una riga da fd 4 (input.txt)
exec 4<&-                                                # Chiude il file descriptor 4

# Redirezioni Complesse con File Descriptor e `exec`

exec 3> out.log 4> err.log                               # Apre fd 3 per stdout e fd 4 per stderr
echo "stdout" >&3                                        # Scrive su out.log tramite fd 3
echo "stderr" >&4                                        # Scrive su err.log tramite fd 4
exec 3>&- 4>&-                                           # Chiude i file descriptor 3 e 4
```

AWK

```bash
awk 'pattern { action }' file.txt   # Struttura base: esegue l'azione su righe che corrispondono al pattern
awk -F, '{ print $1 }' file.csv     # Usa la virgola come delimitatore; stampa la prima colonna
awk '/pattern/' file.txt            # Stampa le righe che contengono "pattern"
awk 'NR==1' file.txt                # Stampa solo la prima riga del file
awk 'END { print NR }' file.txt     # Stampa il numero totale di righe nel file
awk '{ sum += $1 } END { print sum }' file.txt  # Somma i valori nella prima colonna
awk '{ if ($1 > 10) print $0 }' file.txt  # Stampa le righe dove il primo campo è maggiore di 10
awk '{ print $NF }' file.txt         # Stampa l'ultimo campo di ogni riga
awk 'length($0) > 80' file.txt       # Stampa le righe con più di 80 caratteri
awk 'BEGIN { FS=":"; OFS="," } { print $1, $2 }' file.txt  # Cambia delimitatore da ':' a ',' e stampa le prime due colonne

awk 'BEGIN { print "Inizio" }'      # Esegue l'azione all'inizio dell'esecuzione
awk 'END { print "Fine" }'          # Esegue l'azione alla fine dell'esecuzione
awk '{ print NR, $0 }' file.txt     # Stampa il numero di riga e il contenuto di ogni riga
awk -v var=10 '{ if ($1 > var) print $0 }' file.txt  # Usa variabili per confronti
awk '{ $1 = $1 * 2; print }' file.txt  # Modifica il primo campo e lo stampa
awk 'gsub(/pattern/, "replacement")' file.txt  # Sostituisce "pattern" con "replacement" in ogni rig
awk 'NF == 0' file.txt              # Stampa le righe vuote
```

Cut

```bash
cut -c 1-5 file.txt                      # Estrae i primi 5 caratteri di ogni riga
cut -c 3,7 file.txt                      # Estrae solo i caratteri 3 e 7 di ogni riga
cut -d ',' -f 2 file.txt                 # Estrae la seconda colonna in file delimitato da virgole
cut -d ',' -f 1,3 file.txt               # Estrae la prima e terza colonna
cut -f 2-4 file.txt                      # Estrae colonne da 2 a 4 (tab-separated di default)
cut -d ':' -f 1,3 --output-delimiter='-' /etc/passwd # Estrae e unisce con delimitatore custom

# Usare `cut` con Pipe
echo "campo1,campo2,campo3" | cut -d ',' -f 2    # Estrae "campo2"
ps aux | cut -d ' ' -f 1                         # Estrae il nome dell'utente
df -h | cut -d ' ' -f 1,5                        # Estrae filesystem e uso percentuale
cat data.tsv | cut -f 1,3 --output-delimiter='|' # Estrae colonne e cambia delimitatore a '|'
```

Sort

```bash
sort file.txt                            # Ordina alfabeticamente
sort -r file.txt                         # Ordina in ordine inverso
sort -n file.txt                         # Ordina numericamente
sort -u file.txt                         # Ordina e rimuove le righe duplicate
sort -o sorted.txt file.txt              # Ordina file.txt e salva il risultato in sorted.txt
sort -t ',' -k 2 file.txt                # Ordina per seconda colonna usando "," come delimitatore
sort -k 3,3n file.txt                    # Ordina numericamente solo la terza colonna

# Ordinare con opzioni avanzate
sort -k 2,2 -k 3,3n file.txt             # Ordina per seconda colonna, poi terza colonna numericame
sort -f file.txt                         # Ordina ignorando maiuscole e minuscole
sort --parallel=4 file.txt               # Usa 4 thread per l'ordinamento

# Usare `sort` con pipe
ps aux | sort -k 3 -r                    # Ordina i processi per utilizzo CPU (decrescente)
ls -l | sort -k 5 -n                     # Ordina i file per dimensione
cat data.csv | sort -t ',' -k 1,1 -u     # Ordina CSV per prima colonna e rimuove duplicati
```

Head

```bash
head file.txt                            # Mostra le prime 10 righe (default)
head -n 5 file.txt                       # Mostra le prime 5 righe
head -c 20 file.txt                      # Mostra i primi 20 byte
head -v file1.txt file2.txt              # Mostra le prime 10 righe di più file con nomi visibili
head -q file1.txt file2.txt              # Mostra i primi 10 righe di più file senza intestazioni

# Usare `head` con pipe
ls -l | head -n 3                        # Mostra i dettagli dei primi 3 file
ps aux | head                            # Mostra le prime 10 righe della lista dei processi
```

Tail

```bash
tail file.txt                          # Mostra le ultime 10 righe (default)
tail -n 5 file.txt                     # Mostra le ultime 5 righe
tail -c 20 file.txt                    # Mostra gli ultimi 20 byte di file.txt
tail -v file1.txt file2.txt            # Mostra le ultime 10 righe di più file con nomi visibili

# Modalità di monitoraggio in tempo reale (molto utile per log)
tail -f logfile.log                    # Mostra le nuove righe aggiunte a logfile.log in tempo re
tail -n 20 -f logfile.log                # Mostra le ultime 20 righe e poi continua a monitorare il file
tail -F logfile.log                    # Monitora anche dopo riavvii o ricreazioni del file

# Usare `tail` con pipe
dmesg | tail -n 20                     # Mostra le ultime 20 righe del log del kernel
ls -lt | tail                              # Mostra i file più vecchi nella directory (ordinati per data)
```

Word Count

```bash
wc file.txt                            # Conta righe, parole, e byte in file.txt
wc -l file.txt                         # Conta solo le righe
wc -w file.txt                         # Conta solo le parole
wc -c file.txt                         # Conta solo i byte
wc -m file.txt                         # Conta solo i caratteri (multibyte-safe)
wc -L file.txt                         # Lunghezza della riga più lunga in file.txt

# Contare file multipli
wc file1.txt file2.txt                 # Conta righe, parole, byte per ciascun file e totale comb

# Usare `wc` con pipe
echo "Hello World" | wc -w             # Conta parole nell'output di echo
ls | wc -l                             # Conta il numero di file nella directory corrente
```

Compression/Decompression

1a. zip

Compresses one or more files into *.zip files.

```bash
zip foo.zip /bar.txt               # Compress bar.txt into foo.zip
zip foo.zip /bar.txt /baz.txt      # Compress bar.txt and baz.txt into foo.zip
zip foo.zip /{bar,baz}.txt         # Compress bar.txt and baz.txt into foo.zip
zip -r|--recurse-paths foo.zip /bar # Compress directory bar into foo.zip
```

2a. gzip

Compresses a single file into *.gz files.

```bash
gzip /bar.txt foo.gz           # Compress bar.txt into foo.gz and then delete bar.txt
gzip -k|--keep /bar.txt foo.gz # Compress bar.txt into foo.gz
```

3a. tar -c

Compresses (optionally) and combines one or more files into a single **.tar,** .tar.gz, **.tpz or** .tgz file.

```bash
tar -c|--create -z|--gzip -f|--file=foo.tgz /bar.txt /baz.txt # Compress bar.txt and baz.txt into foo.tgz
tar -c|--create -z|--gzip -f|--file=foo.tgz /{bar,baz}.txt     # Compress bar.txt and baz.txt into foo.tgz
tar -c|--create -z|--gzip -f|--file=foo.tgz /bar              # Compress directory bar into foo.tgz
```

1b. unzip

```bash
unzip foo.zip          # Unzip foo.zip into current directory
```

3b. tar -x

```bash
tar -x|--extract -z|--gzip -f|--file=foo.tar.gz # Un-compress foo.tar.gz into current directory
tar -x|--extract -f|--file=foo.tar              # Un-combine foo.tar into current directory
tar -xv                                         # Verbosely list files processed
```

Identifying Processes

```bash
top                    # List all processes interactively
htop                   # List all processes interactively
ps all                 # List all processes
pidof foo              # Return the PID of all foo processes

CTRL+Z                 # Suspend a process running in the foreground
bg                     # Resume a suspended process and run in the background
fg                     # Bring the last background process to the foreground
fg 1                   # Bring the background process with the PID to the foreground

sleep 30 &             # Sleep for 30 seconds and move the process into the background
jobs                   # List all background jobs
jobs -p                # List all background jobs with their PID

lsof                   # List all open files and the process using them
lsof -itcp:4000        # Return the process listening on port 4000
```

Process Priority

Process priorities go from −20 (highest) to 19 (lowest).

```bash
nice -n -20 foo        # Change process priority by name
renice 20 PID          # Change process priority by PID
ps -o ni PID           # Return the process priority of PID
```

Killing Processes

```bash
CTRL+C                 # Kill a process running in the foreground
kill PID               # Shut down process by PID gracefully. Sends TERM signal.
kill -9 PID            # Force shut down of process by PID. Sends SIGKILL signal.
pkill foo              # Shut down process by name gracefully. Sends TERM signal.
```

```bash
pkill -9 foo          # force shut down process by name. Sends SIGKILL signal.
killall foo           # Kill all process with the specified name gracefully.
```

Secure Shell Protocol (SSH)

```bash
ssh hostname              # Connect to hostname using your current user name over the default SSH port 22
ssh -i foo.pem hostname   # Connect to hostname using the identity file
ssh user@hostname         # Connect to hostname using the user over the default SSH port 22
ssh user@hostname -p 8765 # Connect to hostname using the user over a custom port
ssh ssh://user@hostname:8765 # Connect to hostname using the user over a custom port
```

Set default user and port in `~/.ssh/config`, so you can just enter the name next time:

```bash
$ cat ~/.ssh/config
Host name
  User foo
  Hostname 127.0.0.1
  Port 8765
$ ssh name
```

Process management

```bash
ps                    # Process status, information about processes running in memory
top                   # Process viewer, find the CPU-intensive programs currently running (real-tim
ulimit -u 300         # Process limit for single user
./program1 ; ./program2 # Sequential execution
./program1 & ./program2 # Parallel execution

```

Command History

```bash
!!          # Run the last command

touch foo.sh
chmod +x !$   # !$ is the last argument of the last command i.e. foo.sh
```

Navigating Directories

```bash
pwd                   # Print current directory path
ls                    # List directories
ls -a|--all           # List directories including hidden
ls -l                 # List directories in long form
ls -l -h|--human-readable # List directories in long form with human readable sizes
ls -t                 # List directories by modification time, newest first
stat foo.txt          # List size, created and modified timestamps for a file
stat foo              # List size, created and modified timestamps for a directory
tree                  # List directory and file tree
tree -a               # List directory and file tree including hidden
tree -d               # List directory tree
cd foo                # Go to foo sub-directory
cd                    # Go to home directory
cd ~                  # Go to home directory
cd -                  # Go to last directory
pushd foo             # Go to foo sub-directory and add previous directory to stack
popd                  # Go back to directory in stack saved by `pushd`
```

Creating Directories

```bash
mkdir foo                     # Create a directory
mkdir foo bar                 # Create multiple directories
mkdir -p|--parents foo/bar    # Create nested directory
mkdir -p|--parents {foo,bar}/baz # Create multiple nested directories

mktemp -d|--directory         # Create a temporary directory
```

Moving Directories

```bash
cp -R|--recursive foo bar                       # Copy directory
mv foo bar                                      # Move directory

rsync -z|--compress -v|--verbose /foo /bar      # Copy directory, overwrites destination
rsync -a|--archive -z|--compress -v|--verbose /foo /bar # Copy directory, without overwriting destination
rsync -avz /foo username@hostname:/bar          # Copy local directory to remote directory
rsync -avz username@hostname:/foo /bar          # Copy remote directory to local directory
```

Deleting Directories

```bash
rmdir foo                     # Delete empty directory
rm -r|--recursive foo         # Delete directory including contents
rm -r|--recursive -f|--force foo # Delete directory including contents, ignore nonexistent files and never prompt
```

Creating Files

```bash
touch foo.txt          # Create file or update existing files modified timestamp
touch foo.txt bar.txt  # Create multiple files
touch {foo,bar}.txt    # Create multiple files
touch test{1..3}       # Create test1, test2 and test3 files
touch test{a..c}       # Create testa, testb and testc files

mktemp                 # Create a temporary file
```

stdin, stdout and stderr

```bash
echo "foo" > bar.txt       # Overwrite file with content
echo "foo" >> bar.txt      # Append to file with content

ls exists 1> stdout.txt    # Redirect the standard output to a file
ls noexist 2> stderror.txt # Redirect the standard error output to a file
ls 2>&1 > out.txt          # Redirect standard output and error to a file
ls > /dev/null             # Discard standard output and error

read foo                   # Read from standard input and write to the variable foo
```

Moving Files

```bash
cp foo.txt bar.txt                                  # Copy file
mv foo.txt bar.txt                                  # Move file

rsync -z|--compress -v|--verbose /foo.txt /bar      # Copy file quickly if not changed
rsync z|--compress -v|--verbose /foo.txt /bar.txt # Copy and rename file quickly if not changed
```

Deleting Files

```bash
rm foo.txt            # Delete file
rm -f|--force foo.txt # Delete file, ignore nonexistent files and never prompt
```

Reading Files

```bash
cat foo.txt           # Print all contents
less foo.txt          # Print some contents at a time (g - go to top of file, SHIFT+g, go to bottom of file, /foo to search for 'foo')
head foo.txt          # Print top 10 lines of file
tail foo.txt          # Print bottom 10 lines of file
open foo.txt          # Open file in the default editor
wc foo.txt            # List number of lines words and characters in the file
```

File Permissions

| # | Permission | rwx | Binary | | — | — | — | — | | 7 | read, write and execute | rwx | 111 | | 6 | read and write | rw- | 110 | | 5 | read and execute | r-x | 101 | | 4 | read only | r– | 100 | | 3 | write and execute | -wx | 011 | | 2 | write only | -w- | 010 | | 1 | execute only | –x | 001 | | 0 | none | — | 000 |

For a directory, execute means you can enter a directory.

| User | Group | Others | Description | | — | — | — | — | | 6 | 4 | 4 | User can read and write, everyone else can read (Default file permissions) | | 7 | 5 | 5 | User can read, write and execute, everyone else can read and execute (Default directory permissions) |

- u - User
- g - Group
- o - Others
- a - All of the above

```bash
ls -l /foo.sh           # List file permissions
chmod +100 foo.sh       # Add 1 to the user permission
chmod -100 foo.sh       # Subtract 1 from the user permission
chmod u+x foo.sh        # Give the user execute permission
chmod g+x foo.sh        # Give the group execute permission
chmod u-x,g-x foo.sh    # Take away the user and group execute permission
chmod u+x,g+x,o+x foo.sh # Give everybody execute permission
chmod a+x foo.sh        # Give everybody execute permission
chmod +x foo.sh         # Give everybody execute permission
```

Finding Files

Find binary files for a command.

```bash
type wget                                   # Find the binary
which wget                                  # Find the binary
whereis wget                                # Find the binary, source, and manual page files
```

`locate` uses an index and is fast.

```bash
updatedb                                    # Update the index

locate foo.txt                              # Find a file
locate --ignore-case                        # Find a file and ignore case
locate f*.txt                               # Find a text file starting with 'f'
```

`find` doesn't use an index and is slow.

```bash
find /path -name foo.txt                    # Find a file
find /path -iname foo.txt                   # Find a file with case insensitive search
find /path -name "*.txt"                    # Find all text files
find /path -name foo.txt -delete            # Find a file and delete it
find /path -name "*.png" -exec pngquant {} # Find all .png files and execute pngquant on it
find /path -type f -name foo.txt            # Find a file
find /path -type d -name foo                # Find a directory
find /path -type l -name foo.txt            # Find a symbolic link
find /path -type f -mtime +30               # Find files that haven't been modified in 30 days
find /path -type f -mtime +30 -delete       # Delete files that haven't been modified in 30 days
```

Find in Files

```bash
grep 'foo' /bar.txt                         # Search for 'foo' in file 'bar.txt'
grep 'foo' /bar -r|--recursive              # Search for 'foo' in directory 'bar'
grep 'foo' /bar -R|--dereference-recursive  # Search for 'foo' in directory 'bar' and follow symbolic links
grep 'foo' /bar -l|--files-with-matches     # Show only files that match
grep 'foo' /bar -L|--files-without-match    # Show only files that don't match
grep 'Foo' /bar -i|--ignore-case            # Case insensitive search
grep 'foo' /bar -x|--line-regexp            # Match the entire line
grep 'foo' /bar -C|--context 1              # Add N line of context above and below each search resu
grep 'foo' /bar -v|--invert-match           # Show only lines that don't match
grep 'foo' /bar -c|--count                  # Count the number lines that match
grep 'foo' /bar -n|--line-number            # Add line numbers
grep 'foo' /bar --colour                    # Add colour to output
grep 'foo\\|bar' /baz -R                     # Search for 'foo' or 'bar' in directory 'baz'
grep --extended-regexp|-E 'foo|bar' /baz -R # Use regular expressions
egrep 'foo|bar' /baz -R                     # Use regular expressions
```

Replace in Files

```bash
sed 's/fox/bear/g' foo.txt              # Replace fox with bear in foo.txt and output to conso
sed 's/fox/bear/gi' foo.txt             # Replace fox (case insensitive) with bear in foo.txt and output to console
sed 's/red fox/blue bear/g' foo.txt     # Replace red with blue and fox with bear in foo.txt and output to console
sed 's/fox/bear/g' foo.txt > bar.txt    # Replace fox with bear in foo.txt and save in bar.txt
```

```bash
sed 's/fox/bear/g' foo.txt -i|--in-place # Replace fox with bear and overwrite foo.txt
```

Symbolic Links

```bash
ln -s|--symbolic foo bar           # Create a link 'bar' to the 'foo' folder
ln -s|--symbolic -f|--force foo bar # Overwrite an existing symbolic link 'bar'
ls -l                              # Show where symbolic links are pointing
```

Compressing Files

zip

Compresses one or more files into *.zip files.

```bash
zip foo.zip /bar.txt               # Compress bar.txt into foo.zip
zip foo.zip /bar.txt /baz.txt      # Compress bar.txt and baz.txt into foo.zip
zip foo.zip /{bar,baz}.txt         # Compress bar.txt and baz.txt into foo.zip
zip -r|--recurse-paths foo.zip /bar # Compress directory bar into foo.zip
```

gzip

Compresses a single file into *.gz files.

```bash
gzip /bar.txt foo.gz           # Compress bar.txt into foo.gz and then delete bar.txt
gzip -k|--keep /bar.txt foo.gz # Compress bar.txt into foo.gz
```

tar -c

Compresses (optionally) and combines one or more files into a single **.tar,** .tar.gz, **.tpz or** .tgz file.

```bash
tar -c|--create -z|--gzip -f|--file=foo.tgz /bar.txt /baz.txt # Compress bar.txt and baz.txt into foo.tgz
tar -c|--create -z|--gzip -f|--file=foo.tgz /{bar,baz}.txt    # Compress bar.txt and baz.txt into foo.tgz
tar -c|--create -z|--gzip -f|--file=foo.tgz /bar              # Compress directory bar into foo.tgz
```

Decompressing Files

unzip

```bash
unzip foo.zip          # Unzip foo.zip into current directory
```

gunzip

```bash
gunzip foo.gz          # Unzip foo.gz into current directory and delete foo.gz
gunzip -k|--keep foo.gz # Unzip foo.gz into current directory
```

tar -x

```bash
tar -x|--extract -z|--gzip -f|--file=foo.tar.gz # Un-compress foo.tar.gz into current directory
tar -x|--extract -f|--file=foo.tar              # Un-combine foo.tar into current directory
```

Disk Usage

```bash
df                     # List disks, size, used and available space
df -h|--human-readable # List disks, size, used and available space in a human readable format

du                     # List current directory, subdirectories and file sizes
du /foo/bar            # List specified directory, subdirectories and file sizes
du -h|--human-readable # List current directory, subdirectories and file sizes in a human readable format
du -d|--max-depth      # List current directory, subdirectories and file sizes within the max depth
du -d 0                # List current directory size
```

Memory Usage

```bash
free                   # Show memory usage
free -h|--human        # Show human readable memory usage
free -h|--human --si   # Show human readable memory usage in power of 1000 instead of 1024
free -s|--seconds 5    # Show memory usage and update continuously every five seconds
```

Packages

```bash
apt update                  # Refreshes repository index
apt search wget             # Search for a package
apt show wget               # List information about the wget package
apt list --all-versions wget # List all versions of the package
apt install wget            # Install the latest version of the wget package
apt install wget=1.2.3      # Install a specific version of the wget package
apt remove wget             # Removes the wget package
apt upgrade                 # Upgrades all upgradable packages
```

Shutdown and Reboot

```bash
shutdown                    # Shutdown in 1 minute
shutdown now "Cya later"    # Immediately shut down
shutdown +5 "Cya later"     # Shutdown in 5 minutes

shutdown --reboot           # Reboot in 1 minute
shutdown -r now "Cya later" # Immediately reboot
shutdown -r +5 "Cya later"  # Reboot in 5 minutes

shutdown -c                 # Cancel a shutdown or reboot

reboot                      # Reboot now
reboot -f                   # Force a reboot
directory
bg                    # Resume a suspended process and run in the background
fg                    # Bring the last background process to the foreground
fg 1                  # Bring the background process with the PID to the foreground

sleep 30 &            # Sleep for 30 seconds and move the process into the background
jobs                  # List all background jobs
jobs -p               # List all background jobs with their PID
```

```bash
lsof                    # List all open files and the process using them
lsof -itcp:4000         # Return the process listening on port 4000
```

Process Priority

Process priorities go from −20 (highest) to 19 (lowest).

```bash
nice -n -20 foo         # Change process priority by name
renice 20 PID           # Change process priority by PID
ps -o ni PID            # Return the process priority of PID
```

Killing Processes

```bash
CTRL+C                  # Kill a process running in the foreground
kill PID                # Shut down process by PID gracefully. Sends TERM signal.
kill -9 PID             # Force shut down of process by PID. Sends SIGKILL signal.
pkill foo               # Shut down process by name gracefully. Sends TERM signal.
pkill -9 foo            # force shut down process by name. Sends SIGKILL signal.
killall foo             # Kill all process with the specified name gracefully.
```

Date & Time

```bash
date                    # Print the date and time
date --iso-8601         # Print the ISO8601 date
date --iso-8601=ns      # Print the ISO8601 date and time

time tree               # Time how long the tree command takes to execute
```

Scheduled Tasks

```
    *       *         *         *           *
Minute, Hour, Day of month, Month, Day of the week
```

```bash
 crontab -l             # List cron tab
 crontab -e             # Edit cron tab in Vim
 crontab /path/crontab  # Load cron tab from a file
 crontab -l > /path/crontab # Save cron tab to a file

 * * * * * foo          # Run foo every minute
 */15 * * * * foo       # Run foo every 15 minutes
 0 * * * * foo          # Run foo every hour
 15 6 * * * foo         # Run foo daily at 6:15 AM
 44 4 * * 5 foo         # Run foo every Friday at 4:44 AM
 0 0 1 * * foo          # Run foo at midnight on the first of the month
 0 0 1 1 * foo          # Run foo at midnight on the first of the year

 at -l                  # List scheduled tasks
 at -c 1                # Show task with ID 1
 at -r 1                # Remove task with ID 1
 at now + 2 minutes     # Create a task in Vim to execute in 2 minutes
 at 12:34 PM next month # Create a task in Vim to execute at 12:34 PM next month
 at tomorrow            # Create a task in Vim to execute tomorrow
```

HTTP Requests

```bash
curl <https://example.com>                                  # Return response body
curl -i|--include <https://example.com>                     # Include status code and HTTP headers
curl -L|--location <https://example.com>                    # Follow redirects
curl -o|--remote-name foo.txt <https://example.com>         # Output to a text file
curl -H|--header "User-Agent: Foo" <https://example.com>    # Add a HTTP header
curl -X|--request POST -H "Content-Type: application/json" -d|--data '{"foo":"bar"}' <https://example.com> # POST JSON
curl -X POST -H --data-urlencode foo="bar" <http://example.com>                    # POST URL Form Encoded

wget <https://example.com/file.txt> .                       # Download a file to the current directory
wget -O|--output-document foo.txt <https://example.com/file.txt> # Output to a file with the specified name
```

Network Troubleshooting

```bash
ping example.com            # Send multiple ping requests using the ICMP protocol
ping -c 10 -i 5 example.com # Make 10 attempts, 5 seconds apart

ip addr                     # List IP addresses on the system
ip route show               # Show IP addresses to router

netstat -i|--interfaces     # List all network interfaces and in/out usage
netstat -l|--listening      # List all open ports

traceroute example.com      # List all servers the network traffic goes through

mtr -w|--report-wide example.com                            # Continually list all servers the network traffic goes through
mtr -r|--report -w|--report-wide -c|--report-cycles 100 example.com # Output a report that lists network traffic 100 times

nmap 0.0.0.0                # Scan for the 1000 most common open ports on localhost
nmap 0.0.0.0 -p1-65535      # Scan for open ports on localhost between 1 and 65535
nmap 192.168.4.3            # Scan for the 1000 most common open ports on a remote IP address
nmap -sP 192.168.1.1/24     # Discover all machines on the network by ping'ing them
```

Hardware

```bash
lsusb                   # List USB devices
lspci                   # List PCI hardware
lshw                    # List all hardware
```

Secure Shell Protocol (SSH)

```bash
ssh hostname            # Connect to hostname using your current user name over the default SSH port 22
ssh -i foo.pem hostname     # Connect to hostname using the identity file
ssh user@hostname           # Connect to hostname using the user over the default SSH port 22
ssh user@hostname -p 8765   # Connect to hostname using the user over a custom port
```

```
ssh ssh://user@hostname:8765 # Connect to hostname using the user over a custom port
```

Set default user and port in `~/.ssh/config`, so you can just enter the name next time:

```bash
$ cat ~/.ssh/config
Host name
  User foo
  Hostname 127.0.0.1
  Port 8765
$ ssh name
```

Bash Script

Variables

```bash
foo=123               # Initialize variable foo with 123
declare -i foo=123    # Initialize an integer foo with 123
declare -r foo=123    # Initialize readonly variable foo with 123
echo $foo             # Print variable foo
echo ${foo}_'bar'     # Print variable foo followed by _bar
echo ${foo:-'default'} # Print variable foo if it exists otherwise print default

export foo            # Make foo available to child processes
unset foo             # Make foo unavailable to child processes
```

Environment Variables

```bash
env            # List all environment variables
echo $PATH     # Print PATH environment variable
export FOO=Bar # Set an environment variable
```

Functions

```bash
greet() {
  local world = "World"
  echo "$1 $world"
  return "$1 $world"
}
greet "Hello"
greeting=$(greet "Hello")
```

Exit Codes

```bash
exit 0   # Exit the script successfully
exit 1   # Exit the script unsuccessfully
echo $?  # Print the last exit code
```

Conditional Statements

Boolean Operators

- `$foo` - Is true
- `!$foo` - Is false

Numeric Operators

- `eq` - Equals
- `ne` - Not equals
- `gt` - Greater than
- `ge` - Greater than or equal to
- `lt` - Less than
- `le` - Less than or equal to
- `e` foo.txt - Check file exists
- `z` foo - Check if variable exists

String Operators

- `=` - Equals
- `==` - Equals
- `z` - Is null
- `n` - Is not null
- `<` - Is less than in ASCII alphabetical order
- `>` - Is greater than in ASCII alphabetical order

If Statements

```bash
if [[$foo = 'bar']]; then
  echo 'one'
elif [[$foo = 'bar']] || [[$foo = 'baz']]; then
  echo 'two'
elif [[$foo = 'ban']] && [[$USER = 'bat']]; then
  echo 'three'
else
  echo 'four'
fi
```

Inline If Statements

```bash
[[ $USER = 'rehan' ]] && echo 'yes' || echo 'no'
```

While Loops

```bash
declare -i counter
counter=10
while [$counter -gt 2]; do
  echo The counter is $counter
  counter=counter-1
done
```

For Loops

```bash
 for i in {0..10..2}
   do
     echo "Index: $i"
   done

```

```
for filename in file1 file2 file3
  do
    echo "Content: " >> $filename
  done

for filename in *;
  do
    echo "Content: " >> $filename
  done
```

Case Statements

```bash
echo "What's the weather like tomorrow?"
read weather

case $weather in
  sunny | warm ) echo "Nice weather: " $weather
  ;;
  cloudy | cool ) echo "Not bad weather: " $weather
  ;;
  rainy | cold ) echo "Terrible weather: " $weather
  ;;
  * ) echo "Don't understand"
  ;;
esac
```

Esercizi con pipeline

Esercizio 1

Trovare il file più grande in un ramo.

```bash
find /var -type f -print0 | xargs -0 du | sort -n | tail -1 | cut -f2 | xargs -I{} du -sh {}
```

1. `find /var -type f -print0`: Trova tutti i file sotto la directory /var.
   - -type f: Considera solo i file.
   - -print0: Stampa i percorsi dei file separandoli con un carattere nullo (0), utile per evitare problemi con spazi nei nomi.
2. `xargs -0 du`: Passa l'elenco dei file a `du` per calcolarne la dimensione.
   - `-0`: Usa il carattere nullo come separatore (compatibile con -print0).
   - `du`: Calcola lo spazio occupato dai file.
3. `sort -n`: Ordina l'output numericamente (dimensioni crescenti).
4. `tail -1`: Prende l'ultima riga dell'output, corrispondente al file più grande.
5. `cut -f2`: Estrae il secondo campo della riga (il percorso del file).
6. `xargs -I{} du -sh {}`: Passa il percorso del file a du per ottenere la dimensione in formato leggibile (-sh).

Esercizio 2

Copiare file mantenendo la gerarchia

```bash
find /usr/include/ -name 's\*.h' | xargs -I{} cp --parents {} ./localinclude
```

1. `find /usr/include/ -name 's\*.h`: Trova tutti i file con nomi che iniziano con s e terminano con .h.
2. `xargs -I{} cp --parents {} ./localinclude`: Copia i file mantenendo la struttura delle directory.
   - `--parents`: Preserva la gerarchia delle directory.

Esercizio 3

Calcolare lo spazio occupato dai file di un utente

```bash
find /home -user user -type f -exec du -k {} \; | awk '{ s = s+\$1 } END {print " Total used: "
```

1. `find /home -user user -type f -exec du -k {} \;`: Trova tutti i file di proprietà di user nella directory /home e calcola la loro dimensione:
   - `-user user`: Seleziona i file di un certo utente.
   - `-exec du -k {} \;`: Esegue du per calcolare la dimensione in KB.
2. `awk '{ s = s+$1 } END {print " Total used: ",s}'`: somma tutte le dimensioni calcolate da du e stampa il totale.

Esercizio 4

Contare il numero di file in un ramo (ramo home)

```bash
find /home -type f | wc -l
```

1. `find /home -type f`: Trova tutti i file nella directory /home.
2. `wc -l`: Conta il numero di righe nell'output di find, che corrisponde al numero di file.

Esercizio 5

Creare un archivio tar.gz contenente tutti i file minori di 50 KB

```bash
find / -type f -size -50k | tar --exclude "/dev/\*" --exclude "/sys/\*" --exclude "/proc/\*"
-cz -f test.tgz -T -
```

1. `find / -type f -size -50k`: Trova tutti i file di dimensione inferiore a 50 KB.
   - `-size -50k`: Seleziona i file con dimensione < 50 KB.
   - `tar --exclude "/dev/\*" --exclude "/sys/\*" --exclude "/proc/\*" -cz -f test.tgz -T -`: Crea un archivio compresso (.tar.gz) con i file trovati:

- `--exclude`: Esclude alcune directory critiche (/dev, /sys, /proc).
  - `-cz`: Crea (c) un archivio compresso con gzip (z).
  - `-f test.tgz`: Specifica il nome dell'archivio.
  - `-T -`: Usa l'elenco dei file dalla stdin (fornito da find).

Esercizio 6

Rinominare tutti i file .png in .jpg

```bash
find . -name '\*.png' | sed -e 's/.\*/mv & &/' -e 's/png$\/jpg/' | bash
```

1. `find . -name '\*.png'`: Trova tutti i file con estensione .png nella directory corrente.

2. `sed -e 's/.\*/mv & &/' -e 's/png\$/jpg/'`: Trasforma i nomi dei file in comandi mv:
   - `s/.\*/mv & &/`: Genera un comando mv per ogni file trovato.
   - `s/png\$/jpg/`: Cambia l'estensione .png in .jpg.
3. `bash`

Esegue i comandi mv generati.

Simulazione esame

Esercizio assembly

```nasm
; ---------------------------------------------
; functions.asm -----------------------------
; ---------------------------------------------

;---------------------------------------------
; int slen(String message)
; String length calculation function
slen:
    push    ebx
    mov     ebx, eax

nextchar:
    cmp     byte [eax], 0
    jz      finished
    inc     eax
    jmp     nextchar

finished:
    sub     eax, ebx
    pop     ebx
    ret

;---------------------------------------------
; void sprint(String message)
; String printing function
sprint:
    push    edx
    push    ecx
    push    ebx
    push    eax
    call    slen

    mov     edx, eax
    pop     eax

    mov     ecx, eax
    mov     ebx, 1
    mov     eax, 4
    int     80h

    pop     ebx
    pop     ecx
    pop     edx
    ret


;---------------------------------------------
; void exit()
```

```
; Exit program and restore resources
quit:
    mov     ebx, 0
    mov     eax, 1
    int     80h
    ret
```

```nasm
; ------------------------------------------------------------
; test.asm ----------------------------------------------------
; ------------------------------------------------------------

%include        'functions.asm'

SECTION .data
aMsg        db      'This is message a', 0h
bMsg        db      'This is message b', 0h

SECTION .text
global  _start

_start:

mov     eax, 2
int     80h

cmp     eax, 0
jz      a

b:
mov     eax, bMsg
call    sprint

call    quit

a:
mov     eax, aMsg
call    sprint

call    quit
```

Rispondere alle seguenti domande riguardanti test.asm, commentando le righe corrispondenti ed iniziando il commento con un `;`.

1. Cosa stiamo caricando in `eax` a riga 16 di test.asm?
2. Commentate le righe 19 e 20 di test.asm
3. Commentate le righe da 29 a 32 (incluse) di test.asm

Risposte:

1. Nellara riga 16 in `eax` viene caricato il valore `2`. Su Linux, ogni chiamata di sistema ha un codice identificativo specifico; in questo caso, il codice `2` identifica la chiamata di sistema `fork()`. La funzione `fork()` è una chiamata di sistema che crea un nuovo processo come copia esatta di quello corrente. Dopo aver impostato `eax` a `2`, il programma utilizza l'istruzione `int 80h` per eseguire la chiamata di sistema: `int 80h` è un'interruzione software che avvisa il kernel di eseguire l'operazione richiesta, qui identificata da `eax = 2`.
2. Commento righe 19-20 di test.asm

```nasm
cmp eax, 0  ; Verifica se il processo è figlio (eax = 0) o genitore (eax > 0)
jz  a ; Se eax è 0 (siamo nel processo figlio), salta all'etichetta "a"
```

3. Commento righe 29-232 di test.asm:

```nasm
a:  ; Etichetta "a", punto d'ingresso per il processo figlio
mov eax, aMsg ; Carica l'indirizzo di "aMsg" (messaggio a) in eax
call  sprint  ; Chiama la funzione sprint per stampare il messaggio "aMsg"
call  quit  ; Chiama la funzione quit per terminare il processo
```

Esercizio pipeline

Utilizzare `strace` per listare tutte le chiamate di sistema effettuate durante l'esecuzione del comando `df -h` , il cui nome inizi con la lettera `s` ed il cui nome abbia, in terza posizione, una vocale. Ordinarle per frequenza di chiamata ed estrarre la chiamata di sistema effettuata più frequentemente. Stampare in output "n syscallname" dove n è il numero di occorrenze della chiamata a syscallname. Risolvete l'esercizio utilizzando una pipeline.

```nasm
strace -f df -h 2>&1 | grep '^s.[aeiou]' | cut -d '(' -f 1 | sort | uniq -c | sed 's/^ *//' | sort -nr | head -n 1
```

Oppure

```nasm
strace -f df -h 2>&1 | grep '^s.[aeiou]' | cut -d '(' -f 1 | sort | uniq -c | sort -nr | head -n 1 | awk '{print $1, $2}'
```