Guida essenziale al terminale Linux per la bioinformatica

Fabrizio Mafessoni, Università di Trieste, Genomica Computazionale 2025-2026

Questo piccolo manuale ha lo scopo di elencare tutti i comandi del terminale Linux necessari per la manipolazione e gestione di file (bioinformatici e non), la scrittura di piccoli programmi e pipeline bioinformatiche. La notazione segue questa convenzione. I comandi sono specificati in `Courier New`, gli output preceduti da # . Ad esempio:

```
echo 'this is a command'
#this is a command
```

A volte userò termini come *CARTELLA* o *FILE* come segnaposto per degli esempi (questi non indicano il vero nome di una cartella chiamata *CARTELLA*). Saranno scritti in *MAIUSCOLO CORSIVO*. Ad esempio:

```
echo 'UNA_PAROLA'
```

Alla guida dei comandi è seguita da una seconda sezione “Un esempio istruttivo”. L’idea è che possiate utilizzare questo manuale in due modi diversi: studiando la lista dei comandi, uno per uno, o se già siete familiari con il terminale, usandola come referenza per esplorare qualche nuovo comando; oppure, seguendo “Un esempio istruttivo” comando per comando, terminale alla mano, per apprendere in pratica come utilizzare i comandi base, e poi approfondire usando la guida. Per molti comandi esistono moltissime opzioni ed utilizzi qui non elencati e sui quali potete approfondire con manuali/chatpgt&co/internet o con `man COMANDO`!

Lista dei programmi principali

Comandi basilari e di gestione files

`echo FRASE`
echo stampa sullo schermo una FRASE. E’ possibile svolgere simboli speciali, come nuove linee (\n) e tab con l’opzione -e, ad esempio

```
echo -e 'Ora vado a capo\ne continuo'
```

`date`
Da data e ora di oggi

`whoami`
stampa sullo schermo il nome dell’utente

`pwd`
stampa sullo schermo la cartella in cui vi trovate

`touch` *FILE*
crea un file vuoto di nome *FILE*

mkdir *CARTELLA*
crea una cartella di nome *CARTELLA*

rm -r *CARTELLA*
cancella una cartella. Evitate `rm -r *` se non volete cancellare tutto quello che c'è in una cartella!

ls
elenca files e cartelle nella cartella in cui vi trovate. Esistono varie opzioni:
`-l`: da una lista piu dettagliata
`-h`: stampa la lista in una versione piu' leggibile (human)
`-t`: elenca dal contenuto piu' recente a quello piu' vecchio
`-a`: elenca tutto, anche file nascosti, visibili come *.NOMEDELFILENASCOSTO*
Possono anche essere combinate, ed ls usato per visualizzare il contenuto di cartelle specifiche (esempio ls -lt *CARTELLA* mostra dettagliatamente il contenuto di *CARTELLA* dal file piu' recente; notate come di default ls mostri il ccompia ls ./) o verificare l'esistenza di un file *FILE* (con `ls FILE`).
Può essere usato insieme al carattere speciale *, che indica "qualsiasi carattere". Ad esempio `ls *.txt` elenca tutti i file .txt, oppure `ls INIZIO*FINE` elenca tutti i file che cominciano con *INIZIO* e terminano per *FINE*

cd *CARTELLA*
entra in una cartella. Una cartella può essere specificata con path assoluto o relativo.

Un path assoluto indica il nome "preciso" di un file o cartella di modo tale che possa essere raggiunto da qualsiasi punto nel computer, e può essere riconosciuto dal fatto che comincia sempre con una /, in quanto la radice (o root) del computer è rappresentata appunto con /. Ad esempio `/mnt/` è un path assoluto che identifica dove si trovano i vari hard drives del computer, mentre `/bin/` contiene i comandi che stiamo eseguendo ora: probabilmente troverete sul vostro computer `/bin/ls`, il path per il comando `ls`; notate come quando digitate ls, linux on fa altro che eseguire il comando `/bin/ls`.

Un path relativo indica la posizione di un file o cartella relativamente alla nostra posizione attuale nel computer, specificata come `./`. Pertanto, se all'interno della cartella in cui vi trovate esiste una sottocartella *CARTELLA*, potete eseguire `cd CARTELLA` per entrarvi (o `cd ./CARTELLA`). Per uscirne (o andare nella cartella madre contenente quella attuale) eseguite cd .., mentre per tornare nella cartella iniziale, la vostra home, potete usare `cd ~`, o `cd ${HOME}`.

du -h
elenca quanto occupano file e cartelle nella vostra cartella. Utile per capire quali cartelle occupano molto e devono essere cancellate

df -h
Elenca lo spazio disponibile

cat *FILE*
mostra l'intero contenuto di uno o *piu'* files, uno dopo l'altro. Se il vostro file è compresso (spesso con estenzione .gz) potete usare `zcat FILE` per vederne il contenuto senza doverlo prima decomprimere

head *FILE*
mostra le prime 10 righe del file. Per vederne piu' o meno (esempio 3) `head -3 FILE`.

tail *FILE*
come head ma nostra la fine del file

less *FILE*
mostra il contenuto del file dandoci la possibilita' di esplorarlo su e giu'. Utile per file con righe molto lunghe è `less -S FILE`, che non forza le righe ad andare a capo.

wc
Conta righe (con `-l`), parole (con `-w`) o caratteri (con `-c`) di un FILE o STRINGA.

cp *FILE DESTINAZIONE*
Copia un file di origine FILE in una destinazione. Se *DESTINAZIONE* è una cartella, *FILE* è copiato dentro la cartella. Se *DESTINAZIONE* è il nome di un file, *FILE* verrà copiato con quel nome. Possono anche essere specificati piu' file di origine usando i caratteri speciali o wildcards (ad esempo `cp *.txt DESTINAZIONE`) o copiano una cartella intera con sintassi `cp -r CARTELLA DESTINAZIONE`.

mv *FILE DESTINAZIONE*
come `cp` ma spostando (move) quindi cancellando il file di origine. Di fatto questo rinomina file e cartelle.

paste FILE1 FILE2
stampa *FILE1* e *FILE2* di fianco uno all'altro

man COMANDO
Da un manuale (se c'è) per *COMANDO*

Variabili, loop e condizionali

Il terminale e XX ha tutti gli elementi fondanti dei linguaggi di programmazione, potendo definire **variabili** ed usarle per effettuare operazioni. Possiamo definire una variabile con:
`NOME_VARIABILE=VALORE_VARIABILE`
Per richiamarne il valore serve usare il dollaro (o il dollaro con parentesi graffa, `$NOME_VARIABILE` o `${NOME_VARIABILE}`). Per vederne il valore possiamo usare `echo $NOME_VARIABILE`
Per assegnare il risultato di un comando ad una variabile usiamo `$( COMANDO )`, ad esempio:

```
NOME_VARIABILE=$( whoami )
```

if [CONDIZIONE];then
COMANDO
fi
Esegue un *COMANDO* quando la *CONDIZIONE* è vera. then e if fungono da parentesi delimitando il o i comandi da eseguire. Possono anche essere definite delle condizioni piu' complesse con la sintassi:

```
if [ CONDIZIONE1 ];then
COMANDO1
elif [ CONDIZIONE2 ];then
COMANDO2
else
COMANDO3
fi
```

Le condizioni possono essere numeriche (`$VARIABILE -gt 5` , il valore di *VARIABILE* è maggiore di 5), coinvolgere stringhe (`$VARIABILE == "ROSSO"`), o file (`-f FILE` , file esiste).

for
Esegue un COMANDO ripetutamente, ogni volta assegnando ad una VARIABILE un valore diverso elencato come VALORE1, VALORE2, etc, seguendo la sintassi:

```
for VARIABILE in VALORE1 VALORE2 ..;do
COMANDO
done
```

Il comando può non essere uno solo, e le espressioni do e done fungono da "parentesi", racchiudendo tutti i comandi da operare ogni volta.

while
Esegue un COMANDO finché condizione è vera. Attenzione perché questo può generare dei loop infiniti se una variabile è sempre vera e non è mai modificata affinché non diventi falsa ad un certo punto:

```
while [ CONDIZIONE ];do
COMANDO
done
```

Redirezioni e manipolazione di stringhe

COMANDO > FILE
Redirige l'output (sullo schermo) del comando, all'interno di FILE. Da notare che FILE viene creato (quindi se già esisteva FILE, questo viene sovrascritto)

COMANDO >> FILE
Come > ma invece di sovrascrivere FILE se già esistente, appende alla fine di FILE

COMANDO1 | COMANDO2
L'output del primo comando COMANDO1 viene rediretto come input del secondo comando COMANDO2. Il simbolo | viene indicato come pipe, e rappresenta ciò che è

una pipeline (anche in bioinformatica): una sequenza di comandi in serie il cui output del primo diventa input del secondo.

grep
grep trattiene le righe contenenti un certo pattern, che nel piu' semplice caso è semplicemente una stringa di caratteri. Può essere utilizzato come `grep PATTERN FILE` oppure piu' frequentemente con una pipe: `cat FILE | grep PATTERN`. Può essere anche utilizzato per escludere stringhe con `-v PATTERN`, o per cercare stringhe complesse con l'utilizzo dei caratteri speciali. Ad esempio grep A.A cattura tutte le stringhe AXA, dove X è qualsiasi carattere (al massimo in una copia, come ATA), o qualsiasi carattere tra due A, con A.*A.

sed
sed si usa in modo simile a grep ed esegue semplici operazioni di sostituzione e manipolazione su una stringa. I molteplici utilizzi di grep possono essere esplorati con man sed. Tra gli usi piu' comuni c'è la sostituzione (s/) di un *PATTERN* con una nuova stringa *STRINGA_NUOVA* con
`cat FILE | sed 's/PATTERN/STRINGA_NUOVA/'`
Quando è intesa come "tutte le occorrenze del PATTERN" e non solo la prima in ogni linea si usa `sed 's/PATTERN/STRINGA_NUOVA/g'` per "g" di global.

tr
translates è simile a sed, ma in permette delle operazioni con classi di caratteri. Ad esempio può convertire minuscole in maiuscole (`echo "hello world" | tr 'a-z' 'A-Z'`), rimuovere (-d per delete) nuove linee (`tr -d '\n'`) o punteggiatura (`tr -d '[:punct:]'`) e altre.

awk
awk è un linguaggio di programmazione e manipolazione di testi, particolarmente utile per i dati tabulari. Lavora riga per riga, avendo quindi il vantaggio di poter lavorare su file grandissimi che altrimenti non potrebbero essere processati con altri programmi per limiti di memoria. La sintassi tipica `awk` è
`cat FILE | awk '{print $3,$5}'` che stampa colonna 3 e 5 del file *FILE*. Con awk potete anche definire variabili con `-v`, ed usare variabili predefinite come NF (numero delle colonne), NR (numero della riga corrente), FS (separatore di colonne di input) e OFS (separatore di colonne di output):
`cat FILE | awk -v FS=',' -v OFS='\t' '{print "linea n."NR,$1, $NF}'` che stampa, numerando ogni riga, la prima ed ultima colonna di un file comma-separated (,) come file tab-separated (\t). Awk permette anche l'utilizzo di for, if e while loops. Ad esempio, potremmo voler stampare solo le prime 6 colonne per le righe che non iniziano con ## (a volte headers in formati bioinformatici come vcf) usando la funzione printf (che sostituisce %s con una stringa, nel nostro caso $i):

```
cat FILEVCF | awk '{if (substr($1,1,2)!="##"){for (i=1;i<6;i++){printf "%s\t",$i};printf "\n"}'
```

Installazione e monitoraggio di programmi

sudo COMANDO

Esegue un qualsiasi comando come amministratore. Richiede la password.

apt
Comando di base su ubuntu per installare altri programmi. Spesso si lamenterà che non avete i permissi: necessita che lo usiate con sudo. Ha vari sottocomandi:
`apt update`: fa un update dei programmi installabili
`apt search` *`PAROLA`*: cerca i programmi installabili disponibili che hanno quella parola nella descrizione.
`apt install` *`PROGRAMMA`*: installa un programma.
Esempio:

```
apt install wget
apt search genome mapping minimap
apt install minimap2
```

brew
Come apt ma solo per utenti mac: `brew install PROGRAMMA`

conda
A volte vogliamo installare dei programmi ed usarli solo in contesti specifici senza che questi vadano a confliggere con programmi usati in altri contesti. Possiamo isolare questi contesti come *environments* utilizzando *conda*, un manager di ambienti e programmi che ci permette di installare rapidamente da internet tutta una serie di programmi andando a scaricare tutte le librerie necessarie per il loro funzionamento.
Potete scaricare conda con (se il link non è attivo verificate quello corrente):

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
chmod +x Miniconda3-latest-Linux-x86_64.sh; bash Miniconda3-latest-Linux-x86_64.sh
```

Con conda potete creare un ambiente `(conda create AMBIENTE)` e poi attivarlo `(conda activate AMBIENTE)`. Vedrete a questo punto il nome dell'ambiente apparire tra parentesi nel prompt), e potrete a questo punto installare programmi che saranno disponibili solo una volta attivato l'ambiente, pertanto in modo isolato dagli altri ambienti al fine di evitare conflitti tra programmi e specificare librerie speciali. Per installare usare `conda install PROGRAMMA,` potenzialmente da uno specifico repository con -c.
La lista di ambienti creati e pacchetti installati nell'ambiente attivo sono visualizzati rispettivamente con conda env list e conda list. Ad esempio possiamo creare un ambiente per lavorare su files bam :

```
conda -n env_bam python=3.9
conda activate env_bam
conda install -c bioconda samtools
conda list
samtools --help
conda deactivate
```

Raccomando l'installazione d

```
#programmi generali:
sudo apt install wget vim ssh
```

```
#programmi di genomica
sudo apt install wget samtools minimap2 bwa bowtie2 bcftools
```

Gestione utente e permessi

chmod
`chmod +x FILE` da il permesso ad un utente di eseguire un *FILE;* `chmod +r` di leggerlo e `chmod +w` di modificarlo. Questi comandi possono anche essere specificate insieme con `chmod +xaw` o con codici numerici, e richiedere sudo. Per visualizzare i permessi di un file usare `ls -l FILE,` il che darà un output che mostrerà una riga di questo tipo `-rwxr-xr—,` in cui il primo carattere indica se abbiamo un file (-), una directory (d) o un link (l) ed i caratteri successivi indicano a gruppi di tre, read, write ed esecutibilità per l'utente (i primi 3), il gruppo dell'utente e tutti (gli ultimi 3), rispettivamente.

id USERNAME
Restituisce l'id di USERNAME.

sudo useradd *USERNAME*
Aggiunge un utente chiamato *USERNAME*. Può essere poi rimosso con `userdel -r` *USERNAME* (-r per rimuovere anche la sua home folder).

sudo passwd *USERNAME*
cambia la password a *USERNAME*

sudo usermod
Cambia gli attributi dell'utente. Ad esempio `sudo usermod -d /new/home` *USERNAME* definisce una nuova cartella home per l'utente username, mentre `sudo usermod -l newname` *USERNAME* cambia il nome username to newname

su username
Entra come utente username, oppure esegue un singolo comando come username con `sudo -u username COMANDO`

Interazione con la rete e networking

wget *LINK*
scarica un file da un indirizzo web *LINK*. Per scaricare dentro una specifica *CARTELLA*: `wget -P` *CARTELLA LINK*

curl
Alternativa a `wget`. Con
`curl -O https://example.com/file.zip` : scarica usando il nome originale
`curl -o X.ZIP https://example.com/file.zip` : scarica *file.zip* rinominandolo *X.ZIP*.

ssh

Vi permette di entrare su un server. Ssh sta per “Secure Shell”. La tipica sintassi è `ssh *USERNAME@HOSTNAME*` oppure `ssh *IP*`. Per accedere a molti server potrebbe essere necessario utilizzare un file chiave, con la flag -i, ad esempio: `ssh –i *FILE_CHIAVE* *USERNAME@HOSTNAME*`. Se volete provare esistono dei server pubblici per provare (ad esempio con `ssh sdf.org` o `ssh test.rebex.net`) ma potrebbero essere instabili (e magari non volete provare per sicurezza).

scp
Usa lo stesso protocollo di ssh, ma potete utilizzarlo per copiare/trasferire files da un server remoto al vostro computer:
`scp *USERNAME@HOSTNAME:PATH_DEL_FILE_REMOTO .*`
o dal vostro computer al server remoto:
`scp *FILE USERNAME@HOSTNAME:PATH_DEL_FILE_REMOTO*`
Potete usare una chiave con -i, come per ssh. Notate che quando lavorate su un server in remoto, spesso genererete dei plots, e potrebbe essere difficile/impossibile visualizzarli in remoto: per questo basta trasferirli sul vostro computer con scp e visualizzarli normalmente.

ping
Testa se un server remoto è raggiungibile. Ad esempio: `ping sdf.org`

Scrittura di programmi e buone pratiche

vim
Vim è un editor di testo leggero. Con vim FILE potete creare/modificare un file. Normalmente, se premerete un carattere, a differenza di Word, questo non verrà inserito nel testo. Per modificare il file, una volta aperto, premete `i` per entrare in modalità INSERT, evidenziata da una scritta INSERT in basso, e durante la quale potrete inserire/modificare caratteri. Per uscire dalla modalità INSERT premete il tasto **`Esc`**. Per salvare, scrivete (non in modalità INSERT) **`:w`**, per uscire **`:q`**, e per salvare ed uscire **`:wq`**. Per forzare l’uscita senza salvare usate **`:q!`**. Esistono altre funzionalità piu’ avanzate molto utili. Ad esempio, potete evidenziare una (o piu’) linee già scritta con **`Shift+v`**, e poi copiarle premendo **`y`** (o tagliarle con **`x`**). Per incollarle (paste), usate poi **`p`**. Vim salva le modifiche non salvate in un file nascosto .swp. Se uscite incorrettamente, vi chiederà se volete ripristinarlo. In questo caso seguite le istruzioni che vim vi darà ed infine cancellate il file .swp indicatovi da vim con `rm`, per evitare questo messaggio quando proverete a riaprire il file.

#COMMENTO
Il simbolo cancelletto permette di introdurre dei “commenti”, cioè delle stringhe che non vengono interpretate come comandi (con alcune eccezioni, vedi prossima sezione sulla *shebang*). Usateli il piu’ possibile per spiegare cosa fanno le varie sezioni del vostro codice e dei vostri programmi!! Oggi sapete cosa fanno, domani non lo ricorderete!

`#!/bin/bash`

Con il terminale è molto semplice scrivere ed eseguire un programma. Semplicemente è necessario scrivere il tipico programma `hello_world` che stampa "Hello World!" semplicemente scrivendo
echo `Hello World!`
all'interno di un file (ad esempio con vim), chiamato ad esempio hello_world.sh. Possiamo poi rendere eseguibile questo programma con
`chmod +x hello_world.sh`
ed infine eseguirlo chiamando il path del file in questione, ad esempio (se il file è stato creato nella cartella corrente)
`./hello_world.sh`
Notate che il programma ha capito automaticamente che doveva interpretare "echo" come un comando Linux. Questo non è dovuto all'estenzione .sh, la quale è solo una convenzione (avremmo potuto chiamare il nostro script hello_world.pdf), ma solo in quanto di default il terminale Linux/bash interpreta il programma come Linux/bash. Potremmo però volere assicurarci di questa interpretazione, aggiungendo la shebang (null'altro che una prima linea che definisce il linguaggio di interpretazione)
`#!/bin/bash`
In alternativa avremmo potuto scegliere altre shebangs. Ad esempio, per programmi R, python o awk avremmo usato:
```
#!/usr/bin/env python3
#!/usr/bin/env Rscript
#!/usr/bin/awk -f
```
e li avremmo eseguiti rispettivamente come
```
python3 PROGRAMMA
Rscript PROGRAMMA
awk -f PROGRAMMA
```

./PROGRAMMA ARGOMENTO
La caratteristica principale di un programma è la flessibilità (la possibilità di usarlo in vari contesti). Questa è data dagli argomenti: delle informazioni aggiuntive che permettono al programma di funzionare in vari modi. In linux è molto semplice definire argomenti posizionali con `$1`,`$2`, etc. per il primo e il secondo argomento, rispettivamente. Ad esempio il `hello_words.sh` definito come
`echo "La prima parola è $1, la seconda $2"`
quando viene eseguito come
`./hello_words.sh forza Roma`
stampa:
`La prima parola è forza, la seconda Roma`

git
https://github.com/ è un sito che dove vengono depositati programmi. Il grande vantaggio di github è che potete usarlo per salvare i vostri programmi gradualmente, in modo tale che vengano salvati tutti i passaggi intermedi e possiate ritornare a versioni precedenti se le modifiche introdotte non vi piacciono. Per questo consiglio di craere un account github su github.com, e creare un piccolo repository PROVA. Esistono molte funzioni per git, ma quelle principali e necessarie sono le seguenti. Prima di tutto scarichiamo il progetto (o lo stesso per installare altri programmi create da altri utenti!):
`git clone https://github.com/TUOUSERNAME/PROVA.git`

Dentro questa cartella potete creare files e modificarli. Dopo ogni modifica, specifico i files modificati. Per aggiungere tutte le modifiche:
`git add .`
Poi assegno un “commit”, una descrizione definitiva di una versione, a queste modifiche. Da ora, potrò tornare a questo “commit” quando vorrò. Inserite descrizioni che chiarifichino brevemente cosa avete fatto:
`git commit -m "Prima modifica a PROVA"`
Con git push uploaderete sul sito, cosi che la vostra modifica sia online:
`git push`
Per scaricarle, usate
`git pull`
Per vedere i vecchi commit disponibili usate
`git log`
e per caricarne uno
`git checkout NOME_DEL_COMMIT`
Per tornare all’ultimo, quello attuale:
`git checkout main`
Durante l’esecuzione di git add e git commit, github prova a risolvere potenziali conflitti tra le versioni online e locali (ad esempio se modificate due file diversi). A volte però, se modificate online e localmente lo stesso file, potrebbero emergere dei conflitti (github non sa quale versione tenere).
Per vedere quali files danno problemi
`git status`
Modificate questi files per risolvere il conflitto. Spesso sarà qualche riga in piu’ o in meno evidenziata nel codice da github con:

```
<<<<<<< HEAD:main-branch
differenze…
=======
```

Github non è necessario per programmare: ma quando inizierete programmi complessi sarà utilissimo, provatelo!

Un esempio istruttivo

Se non volete leggere tutte le noiose spiegazioni sopra, potrete provare ad imparare linux semplicemente eseguendo il codice a seguire e cercando di capire riga dopo riga (magari aiutandovi dove serve con le descrizioni sopra).

```
echo 'Ciao!'
whoami
pwd
mkdir PROVE
mkdir PROVE/SOTTOPROVE
cd PROVE
cd SOTTOPROVE
pwd
cd ..
touch PROVA.txt
ls
```

```
ls *.txt
rm PROVA.txt
rm -r SOTTOPROVE
ls
echo "Ciao" > PROVA.txt
whoami >> PROVA.txt
echo "Addio!" >> PROVA.txt
cat PROVA.txt
wc -l PROVA.txt
head -2 PROVA.txt
tail -2 PROVA.txt
less PROVA.txt
cp PROVA.txt PROVA2.txt
ls
mv PROVA2.txt PROVA_COPIA.txt
ls
cat PROVA_COPIA.txt
cat PROVA.txt PROVA_COPIA.txt
head -1 PROVA.txt > PROVA_COPIA.txt
cat PROVA_COPIA.txt
man ls
NOME="Francesco_Totti"
echo $NOME
MIONOME=$( whoami )
if [ $MIONOME == $NOME ];then
echo "Salve Capitano!"
else
echo "Salve ${MIONOME}!"
fi
for FILE in *.txt;do
echo $FILE
done
for i in pippo pluto topolino minnie paperina paperino
paperone;do
echo $i >> NOMI.txt
done
cat NOMI.txt | grep paper
cat NOMI.txt | grep ^paper
cat NOMI.txt | grep paper$
cat NOMI.txt | grep e$
cat NOMI.txt | tr e i
cat NOMI.txt | sed s/one/ona/
cat NOMI.txt | sed s/one/ona/ | grep paper.na
paste NOMI.txt NOMI.txt | sed s/one/ona/g | grep paperona
paste NOMI.txt NOMI.txt | sed s/one/ona/ | grep paperona
paste NOMI.txt NOMI.txt | sed s/one/ona/ | grep paperona | awk
'{print $2,$1}'
cat NOMI.txt | awk '{print $1,substr($1,1,3)}'
cat NOMI.txt | awk -v OFS='\t' '{if (substr($1,1,3)==
"pap"){city="Paperopoli"} else { city="Topolinia"}; print
$1,city}'
```

```
echo '#!/bin/bash' > hello.sh
echo 'echo Hello $( whoami) !' >> hello.sh
ls -l hello.sh
chmod +x hello.sh
./hello.sh
#Installazione di alcuni comandi utili (chiederà la password)
sudo apt install git wget curl vim ssh
#Installazione di alcuni programmi bioinformatici utili
sudo apt install samtools bwa bowtie2 bcftools vcftools
minimap2 miniasm
```