Introducción a Linux

Contents

Qué es Linux

Linux es un sistema operativo muy versátil. Tiene una curva de aprendizaje que requiere cierto nivel de paciencia, pero afortunadamente hay múltiples [tutoriales](https://ryanstutorials.net/linuxtutorial/)

Uso de linux

- Sólo Linux: Ubuntu
- En Windows: Windows Subsystem for Linux o como una Virtual Machine

Comandos básicos

Skip to main content

Este es un ambiente menos gráfico pero mas sencillo para dar instrucciones...sin embargo hay que saber como darlas para que el sistema responda.

ls (listar)

Cuando se entra al sistema, el directorio donde uno se encuentra es el *home*. Para saber que hay dentro de *home*, escribimos

```
$ ls
```

```
  Cell In[1], line 1
    $ ls
    ^
SyntaxError: invalid syntax
```

El comando ls lista el contenido de directorio de trabajo (current directory).

Pero ls no lista realmene todos los archivos, solo aquellos que son visibles. Los archivos que comienzan con un punto (.) y se conocen como ocultos. Usualmente contienen informacion importante de configuracion que no debe ser alterada, a menos que sepamos bien que estamos haciendo.

Ahora listemos todos los archivos del directorio, incluyendo los ocultos:

```
$ ls -a
```

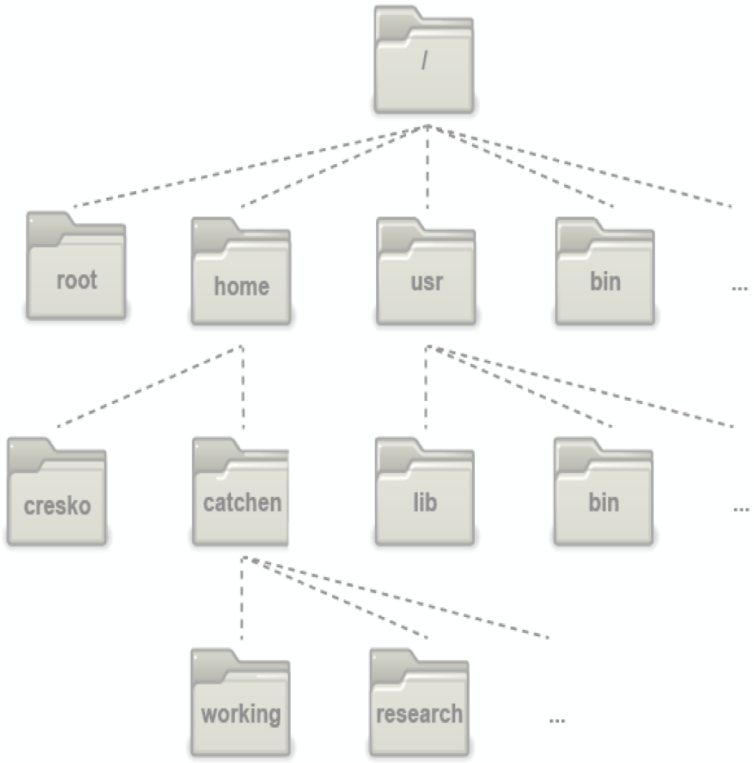
ls es un ejemplo de un comando con opciones: -a es un ejemplo de una opción. Las opciones cambianel comportamiento del comando.

Skip to main content

Nota 1: los nombres de archivos en linux son sensibles a mayúsculas y minúsculas (*case sensitive*), i.e. Test.txt es diferente de test.txt y ambos archivos pueden coexistir en una carpeta. **Nota 2:** los espacios generan muchos errores en los nombres de archivos y carpetas. Es mejor reemplazarlos por guión inferior, i.e Texto Final 2.txt reemplazar por Texto_Final_2.txt

Estructura de los directorios

Todos los archivos están agrupados en la estructura del directorio y esta estructura es jerárquica.



Para saber donde nos encontramos en la estructura del directorio usamos el comando “print working directory”

```
$ pwd
```

Skip to main content

Crear directorios

mkdir (make directory)

Utilizamos este comando para crear un subdirectorio en el directorio donde estamos

```
$ mkdir biocomp_clase1
```

Para ver el directorio que acabamos de crear escribimos

```
$ ls
```

Ejercicio crea una carpeta dentro de biocomp_clase1 que se llame "ejercicios"

Cambiar directorios

cd (change directory)

Utilizamos este comando para cambiar del directorio donde estamos a otro directorio. Para "entrar" al directorio que acabamos de crear escribirmos

```
$ cd biocomp_clase1
```

Escriba ls para ver qué hay en el directorio (debería estar vacío). Para volver al directorio parental escribimos

```
$ cd ..
```

Si seguimos escribiendo cd .. recurrentemente vamos subiendo en la jerarquía de directorios

Skip to main content

~ (home)

Si escribimos `cd` sin puntos llegamos al directorio home. Este se denota también como `~`

```
ls ~/
```

Copiar archivos

cp (copy)

El comando para copiar un archivo arch1 en el directorio actual a un archivo identico pero nombrado arch2 es:

```
$ cp arch1 arch2
```

Ahora vamos a tomar un archivo de internet y a usar el comando cp para copiarlo en el directorio que creamos.

Primero, cd al directorio biocomp_clase1:

```
$ cd ~/biocomp_clase1
```

Luego escribe:

```
$ wget http://www.ee.surrey.ac.uk/Teaching/Unix/science.txt
```

Qué crees que hace el comando wget?

Ejercicio crea una copia backup del archivo "science.txt" que descargaste, llamada "science.bak"

Skip to main content

Usando [TAB]

`[TAB]` es muy útil en la línea de comandos de Linux: completa automaticamente los comandos que se utilizan. Intenta, por ejemplo, `$ cd ~/bioc` y después `TAB`. Esto también ayudará a hacer sugerencias en caso de múltiples opciones.

Moviendo archivos

mv (move)

`mv arch1 arch2` mueve o cambia el nombre de `arch1` a `arch2` y por lo tanto solo queda un

Vamos a mover el archivo `science.bak` a al folder `ejercicios` creado previamente.

Primero, cambia de directorio a `biocomp_clase1`. Despué escribe:

```
$ mv science.bak ejercicios/
```

escribe `ls` y `ls backup` para ver si funcionó.

Remover archivos y directorios

rm (remove), rmdir (remove directory)

Para borrar (remover) un archivos, usamos el comando `rm` command. Como ejemplo, vamos a crear una copia del archivo `science.txt` y borrarlo.

dentro del directorio `biocomp_clase1`, escribe:

```
$ cp science.txt tempfile.txt
$ ls
$ rm tempfile.txt
$ ls
```

Skip to main content

Puedes utilizar el comando `rmdir` para remover el directorio (asegúrate primero que esté vacío). Intenta remover el directorio `backup`. No podrás porque linux no te permitirá remover un directorio que no está vacío. Para borrar un directorio (que no esté vacío) con sus subdirectorios puedes usar `-r` (recursivo):

```
$ rm -r /path/directory
```

Este comando pedirá confirmación para algunos archivos que sean importantes. Si estás absolutamente seguro de este paso, puedes añadir el comando `-f` (f de forzar):

```
$ rm -rf /path/delete
```

Atención Los directorios que se borran con ``rm`` se pierden *para siempre*. No van a ni siquiera a la papelera, si borran directamente.

Mostrar los contenidos de un archivo en la pantalla

less

El comando `less` muestra los contenidos de un archivo en la pantalla. Escribe:

```
$ less science.txt
```

Presiona la `[barra-espaciadora]` si quieres continuar hacia abajo, y escribe `[q]` si quieres terminar la lectura.

head

El comando `head` muestra las diez primeras lineas de un archivo en la pantalla:

```
$ head science.txt
```

luego escribe:

Skip to main content

```
$ head -5 science.txt
```

Qué diferencia hizo el `-5` en el comando?

tail

El comando tail muestra las últimas 10 lineas de un archivo:

```
$ tail science.txt
```

Ejercicio Como podrías ver las últimas 15 lineas de un archivo?

Searching the contents of a file

Búsequeda Simple usando `less`

Usando `less`, se puede buscar dentro de un archivo de texto palabras (un patrón). Por ejemplo, para buscar dentro del archivo `science.txt` la palabra "science", escribimos:

```
$ less science.txt
```

después, aún en `less`, escribe slash `[/]` seguido por la palabra a buscar:

```
/science
```

Y `[enter]`. Escribe `[n]` para buscar por la siguiente ocurrencia en el texto.

grep

`grep` busca en el archivo palabras o patrones dentro de un texto:

```
$ grep science science.txt
```

Como se puede ver `grep` muestra cada línea que contiene la palabra science. Ahora intente

Skip to main content

```
$ grep Science science.txt
```

El comando `grep` es “case sensitive”, distingue entre S y s.

Para ignorar distinciones entre mayúsculas y minúsculas, utilice la opción `-i`:

```
$ grep -i science science.txt
```

Para buscar una frase o patrón, se debe encerrarla entre comillas. Por ejemplo, para buscar spinning top, escriba:

```
$ grep -i 'spinning top' science.txt
```

Otras opciones de `grep` son:

- `-v` muestra las líneas que NO contienen el patrón
- `-n` precede cada línea que contiene el patrón con el número de línea
- `-c` muestra el número total de líneas que contienen el patrón

Intente las diferentes opciones y sus combinaciones. Se pueden utilizar mas de una opción al mismo tiempo! Por ejemplo, el número de líneas sin la palabra science or Science es:

```
$ grep -ivc science science.txt
```

Conclusiones: la línea de comando

La línea de comando de linux es muy poderoso. Tan poderoso que la gran mayoría de los servidores y super computadoras del mundo funcionan con los sistemas linux (sin interfase gráfica, i.e. usando solamente la línea de comando).

Aquí navegamos por algunos ejemplos simples, pero esto es solo un “abre-bocas”. Por ejemplo, el comando `grep` es muy importante y útil para buscar patrones, and `wget` para bajar archivos. Ambos comandos mucho mas rápidos que si se usaran la interfase gráfica.

Para profundizar sobre linux hay varios tutoriales, por ejemplo tutoria llinux

Skip to main content

Scripts Ejecutables

Ahora que ya estamos un poco familiarizados con la línea de comando, vamos a utilizar algunas herramientas básicas de programación: al digitar un comando, este da una instrucción para recibir una inofrmación. Este proceso se puede **automatizar** (por ejemplo mover, renombrar y copiar archivos), escribiendo el comando en un **script**.

Los scripts son la forma mas simple de escribir un programa. Vamos a escribir un **script bash** y a ejecutarlo.

Dentro del directorio `biocomp_clase1` vamos a crear un nuevo archivo llamado `miscript.sh`. You can do this using the default (graphical) text editor in MyBinder. Puedes hacer esto utilizando un editor de texto de terminal llamado `nano`

```
$ nano miscript.sh
```

Este comando inicia el editor de texto. A continuación escriba `echo Hello World!`. "Hello World!" es una frase icónica en programación y "echo" es una instrucción que reproduce el texto que sigue. Guarde el archivo usando `[ctrl-o] [enter]` y cierre el programa con `[ctrl-x]`. En la terminal, ejecute el script con el siguiente comando:

```
$ bash miscript.sh
```

Bash es un "interpretador" que lee el archivo y entiende cómo ejecutar el comando.

Hacer un archivo ejecutable

Liste los archivos en el directorio con la opción `-l` para mas información. Here is how it looks like on MyBinder:

```
$ ls -l
total 12
-rw-r--r-- 1 usuario root   18 Jun 26 10:56 myscript.sh
```

Skip to main content

La primera lista de caracteres indica los **permisos** del archivo. Aquí vemos que el usuario tiene permiso de lectura (`r`) y escritura (`w`), pero no de ejecución. Para entender mejor este tema puede consultar este tutorial sobre permisos. Vamos a cambiar el permiso de ejecución

```
$ chmod a+x miscript.sh
```

Ahora todos los usuarios tienen permiso para ejecutar el archivo. Como es un script muy simple y sin consecuencias, no nos preocupa que todos tengan permisos. En otros casos es muy importante tener cuidado. Ahora lo podemos ejecutar:

```
$ ./myscript.sh
```

Ejemplo de script: Con frecuencia se recomienda añadir una primera línea en el scrip que indica el interpretador que se debe usar. En este caso es el interpretador por defecto (bash), pero no es siempre el caso. Además se usa escribir una línea que describe el propósito del script y que está precedida por el símbolo ``r`` que indica que es un comentario

```
#!/bin/bash
# Script de ejemplo para la clase 1 de Comp. Biol

echo Hello World!
```

Variables, declaraciones y loops

Una variable: ...

```
$ counter=1
$ echo $counter
$ 1
$ echo counter
$ counter
```

Qué diferencia hay entre `counter` y `$counter`? Que la segunda es una **variable** y por lo

Skip to main content

If

Una declaracion (statement) condicional es **if** y determina que, si un chequeo es verdadero, realice una acción, y si es falso no lo haga. **If** tiene el siguiente formato:

`if [ <un chequeo> ]`

`then`

`<comando>`

`fi`

Todo lo que esté entre *then* y *fi* se ejecutará solamente is el chequeo (entre corchetes) es verdadero. Un ejemplo simple

```
$ num1=10
$ num2=5

if [ $num1 -gt $num2 ]; then
echo "num1 es $num1 y es mayor que $num2"
fi
```

While loop

Los loops son muy importantes porque permiten que un comando corra recurrentemente hasta que suceda una situación declarada (statement) o la expresión sea falsa. Este principio es fundamental para la automatización. El loop **while** define que, mientras una expresión es verdadera, continúe ejecutando las líneas de código. Tiene el siguiente formato

`while [ un chequeo ]`

`do`

`<comando>`

`done`

Skip to main content

```
$ counter=1
$ while [ $counter -le 10 ]
$ do
$ echo $counter
$ ((counter++))
$ done
```

Vamos a descomponer las líneas:

- Linea 1: inicializa la variable **counter** con un valor.
- Linea 2: Chequea que mientras (**While**) la declaración es verdad (counter es menor o igual a 10)
- Linea 3: ejecutemos el siguiente comando.
- Linea 4: aquí podemos poner cualquier comando. En este caso Here **echo** se utiliza porque es simple para ilustrar el propósito.
- Linea 6: incrementamos la variable **counter** por 1.
- Linea 7: al final del loop, vuelve al inicio a hacer el chequeo. Si es verdad, ejecuta el comando, si es falso conitnua a la siguiente linnea.

Este programa que ejecutamos en la terminal lo podemos escribir en un script para correrlo las veces que necesitemos y para registrar lo que hicimos

```
$ nano while_loop.sh

#!/bin/bash
# Basic while loop
counter=1
while [ $counter -le 10 ]
do
echo $counter
((counter++))
done
echo All done
```

`[ctrl-o] [enter]` y `[ctrl-x]`

Ejercicio Haga el script ejecutable y córralo

Skip to main content

For loop

Este loop toma cada item en una lista (en orden, uno despeués del otro), asigna ese item como valor a la variable **var**, ejecuta los comandos entre **do** y **done** y vuelve al inicio para tomar el siguiente item y repetir recurrentemente. Tiene la siguiente sintaxis:

`for var in <list>`

`do`

`<comando>`

`done`

El siguiente ejemplo ilustra su uso en un directorio que contiene los archivos fasta de secuencias de covid19. Primero vamos a crear un directorio que se llame **sars_covid19** y luego vamos a descargar allí las secuencias del:

- genoma
- proteinas
- cds (genes codificadores de proteínas)

```
$ mkdir sars_covid19
$ curl http://ftp.ensemblgenomes.org/pub/viruses/fasta/sars_cov_2/dna/Sars_cov
$ curl http://ftp.ensemblgenomes.org/pub/viruses/fasta/sars_cov_2/pep/Sars_cov
$ curl http://ftp.ensemblgenomes.org/pub/viruses/fasta/sars_cov_2/cds/Sars_cov
```

curl ("Client url") es un comando de línea que permite comunicarse con un servidor. La opción `-o` le asigna un nombre al archivo (en este caso simplificamos el nombre). Donde crees que quedaron los archivos?

```
cd sars-covid19
ls
gunzip *.gz
```

Cuando descargamos los archivos los nombramos con un `./sars_covid` que precede al

Skip to main content

directorio donde nos encontramos `.`

```
for f in $(ls);
    do wc -l $f;
        done
```

Note como listamos el directorio usando `ls` y lo encerramos en el comanod `$()`. Esto nos permite crear una variable que contiene el resultado del comando `ls`. Luego iteramos sobre cada item del directorio **sars_covid19** y contamos el número de líneas con `wc -l`

Ejercicio Evaluable 1) crea una carpeta nueva 2) copie los archivos A, B, C 3) crea un script en bash (con el encabezado sobre el interpretador y un comentario del propósito) que itere sobre los archivos del directorio y despliegue los nombres de los archivos. Nombre el script script_eval1_sunombre.sh 4) ejecute script_eval1_sunombre.sh (tenga en cuenta los permisos) 5) ejecute el comando: cat script_eval1_sunombre.sh > reporte1.txt 6) Ejecute el comando: history >> reporte1.txt 7)Suba por la asignación de tarea de Teams el archivo reporte.txt