#DataClubUy

Introducción Bash

BASH

THE BOURNE-AGAIN SHELL

Bash es un **intérprete de comandos** que ejecuta, una por una, las instrucciones introducidas por el usuario o contenidas en un script y devuelve los resultados. En otras palabras, actúa como interfaz entre el kernel Linux y los usuarios o programas del modo texto. Además, incorpora numerosas utilidades de programación y mejoras sobre sh, su shell predecesora. Debido a que es una herramienta desarrollada por GNU, suele ser utilizada por defecto en las distros actuales.

https://denovatoanovato.net/diferencias-entre-terminal-consola-y-shell/

Bash no es una terminal. Tampoco es la única shell disponible.

Ayuda!!

- Google o Duckgogo o el motor de búsqueda que uses.

Stacksoverflow: es un foro de ayuda para programación en general, viene dividido por lenguajes. La gente hace preguntas y contesta. Es como un yahoo answers super pro, lxs programadorxs con muchas preguntas respondidas ganan puntos que son algo muy visible para un CV con ese perfil. Este foro es mejor para dudas de bash, R, python y no de programas genéticos (Biostars por ejemplo).

Grupos de usuarios de un software. Por ejemplo este de Stacks. Son ideales para aclarar detalles específicos y mensajes de error de ese software (ie, no de dudas de cómo usar la línea de comando). La comunidad es súper chida y muchas veces lxs autores del software son quiénes responden.

Cómo pedir/buscar ayuda

- Busca en inglés
- Piensa bien cuáles son las palabras clave de tu pregunta y cómo generalizar tu caso a algo que cualquiera entienda y que no sea específico a tu computadora.
- Si vas a pedir ayuda en un foro, lee las reglas y tipo de preguntas atentidas por el foro antes de preguntar.
- Sigue estas recomendaciones de Stacksoverflow sobre cómo redactar una buena pregunta
- Si recibes un mensaje de error en la Terminal, R, etc. **Copia-pégalo** a tu motor de búsqueda favorito.

Gracias a Alicia Yañes por su valioso material de referencia.


BUENO
EMPECEMOS
memegenerator.es

Introducción a la consola y línea de comando de bash

Abrir laterminal.

(ya sé que usan windows y no tienen shell Bash, así que https://mobaxterm.mobatek.net/ **)**

Ahora sí, abrir la terminal.

La primera línea dice la fecha y hora y "ttys001". ttys viene de "Teletype" (un poco de historia aquí) y significa que el *input* es nuestro teclado, identificado como ttys001.

La otra línea dirá el nombre del equipo, el directorio donde estoy (`~` significa "home directory") y el nombre del usuario con quién estoy en la terminal .

`$` significa que la terminal está corriendo con un interpretador Shell o Bash y por un usuario sin mayores privilegios. Si termina en `#` significa que la estás corriendo como root que es un "súper usuario" con permisos para desconfigurarlo todo, ten cuidado.

En la Terminal no existe el *point and click*.

El que funcione como una Línea de Comando significa que tienes que darle comandos de qué hacer línea por línea.

Por ejemplo:

`date` nos responde la fecha actual

`echo algo` nos responde el texto "algo". También lo puedes utilizar con más de una palabra.

```
echo hello world
```

Para facilidad visual, de aquí en adelante utilizaremos la opción con el texto a partir de `$` (al menos que sea pertinente ver lo que hay antes).

Este tipo de formato te lo encontrarás en diversos foros de ayuda, libros de textos y manuales.

Ojo

Se pone $ solo para distinguir que lo que sigue es un comando que debes poner en la Terminal, por lo que **NO debes copiar los comandos a tu terminal incluyendo el** `$`.

Ejemplo de errores

La práctica hace al maestro.

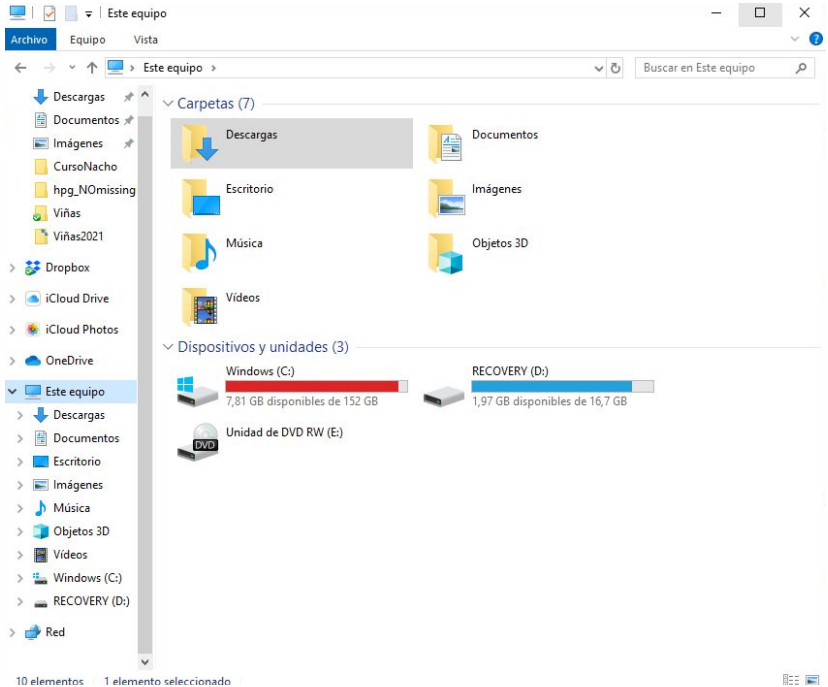
A continuación veremos algunos de los comandos básicos, pero para practicar más te recomiendo :

- Terminar el curso Learn the Command Line de CodeAcademy.
- Adentrarse en el bosque con este juego de bash.

Funciones básicas de navegación y manejo de archivos con bash

Windows, Mac y las interfaces gráficas de Linux (como Ubuntu y Biolinux) tienen un sistema de archivos que estamos acostumbrados a explorar a través carpetas y subcarpetas que podemos ver en una ventana.

Por ejemplo así:



A continuación vamos a ver cómo navegar por este mismo sistema de archivos, pero desde la Terminal y con el teclado en vez de desde una ventana y con clicks.

`pwd` nos da el directorio en donde estamos (viene de print working directory).

`cd` viene de change directory y sirve para movernos a otro directorio.

Como se explicó antes, el texto antes de `$` nos indica el nombre del equipo, el directorio actual y el nombre del usuario. El directorio actual cambió .

Ahora vamos a navegar hacia otros directorios. La navegación se puede hacer de diferentes maneras:

Para esto hay varias opciones:

Moverse hacia adelante/abajo (i.e. adentro de subdirectorios):

- **Absolute path** que es dar la ruta (dirección) completa desde root hasta el directorio que queremos.
- **Relative path** que es dar la ruta desde donde estamos hasta el directorio que queremos.

Moverse para atrás (hacia el directorio raíz):

Igual puede ser con rutas absolutas o relativas, pero utilizando `..` que representa parent directory, es decir el directorio arriba (o atrás, como le entiendas mejor):

`cd ..` por lo tanto te lleva a un directorio arriba de donde estés.

Moverse para atrás y para adelante en la misma línea

Puedes usar `..` muchas veces. Ojo con incluir `/` para separar niveles.

```
$ cd ../../Manzanas/
```

Es decir `../` se puede combinar con una ruta relativa. Ejemplo:

```
cd ../../Manzanas
```

`cd ./` te lleva al directorio en el que estás. Lo importante a recordar es que `.` significa "el directorio actual".

Errores comunes al usar `cd`

- Ojo, con tus espacios `cd..` no es lo mismo que `cd ..`. Otro clásico es poner `cd ...`

Tips de acceso rápido en la Terminal

Flecha arriba/abajo para acceder a los últimos comandos utilizados TAB para "completar" la escritura del path o nombre de archivos.

`ls`

Enlista los archivos y directorios presentes en un directorio.

Nota que los enlista en orden alfabético.

Para tener más info de los archivos:

`ls -l` brinda la misma lista, pero con datos sobre: si es un directorio (d) o un archivo (-), permisos (si es sólo lectura, editable, etc y por quién, detalles más adelante), número de links al archivo, qué usuario es el dueño, a qué grupo pertenece dicho usuario, tamaño en bytes, fecha-hora en que se modificó y el nombre del directorio o archivo.

Pero se pueden ordenar por fecha (y hora, aunque no se vea) de modificación, por ejemplo:

```
ls -lt
```

Tip: presiona "q" para salir de la pantalla de `man`.

`mkdir`

Crea un directorio.

`cp`

Copia un archivo o directorio de lugar A a lugar B.

```
$ cp -r otrosPerros ../gatos/
```

¿Por qué utilicé el flag `-r` en el ejemplo anterior?

`mv`

Mueve un archivo o directorio de lugar A a lugar B. Equivalente a "cut-paste".

Nota que con `mv` no es necesario utilizar `-r` para mover directorios.

```
$ mv otrosPerros/ ../perros/
```

`rm`

Borra (CUIDADO al usar esto) archivos o directorios.

```
$ rm -r otrosPerros
```

```
$ rm -r ../perros/otrosPerros
```

`tar`

Es un método de ultra comprensión (más que zip) utilizado por sistemas Linux/Unix. Viene de "*tape archive*" y originalmente surgió para comprimir archivos para los discos "tape" de respaldo.

La compresión tar genera archivos "tarball", gzip y bzip. Con terminaciones como `.tar.gz`. Este tipo de compresión es muy utilizada en datos genómicos.

Crear un tar.gz

```
tar cvzf all_perros.tar.gz perros
```

`c` Crea un nuevo archivo .tar.

`v` Muestra "*Verbosely*" el progreso de la compresión.

`z` Especifica que queremos un `.gzip`.

`f` Nombre del archivo tar que queremos como resultado.

```
$ls -lhS
```

¿Cómo borro el tar que acabo de crear?

Descomprimir (Untar) archivos .tar.gz

```
$ tar -xvf all_perros.tar.gz
```

¿Qué hacen las flags `-xvf`?

Crear archivos desde la terminal

Es posible crear archivos de texto directamente desde la terminal utilizando programas como `vi` y `nano` o el comando `touch`.

Touch solo crea un archivo sin contenido.

```
$ touch ejemplo
```

Con `Nano` o `Vim` podemos

```
$ nano ejemplo.txt
```

`vim` o `vi` funciona muy parecido. Solo que tienes que aprenderte (o revisar el trencito) los shortcuts (atajos) del teclado. **Hay gente que vive solo con vim.**

`curl` Sirve para bajar archivos de internet a la computadora.

Sintaxis:

```
curl [opciones] [direccionURLdelarchivo]
```

O bajar sencuencias de ADN de GeneBank! (instrucciones de cómo construir la url aquí)

```
$ curl -s "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&rettype=fasta&id=AB080944".1
```

En bioinformática `curl` se utiliza para transferir desde archivos FASTA de secuencias individuales de GeneBank hasta genomas completos.

Nota: `wget` hace algo parecido a `curl`, pero lo salva a un archivo directamente. No existe de base en Mac, pero es posible instalarlo.

Comodines o el uso de * ? [] {}

*

"Comodín" o *wildcard*. Cualquier texto (uno o más caracteres) a partir (derecha o izquierda) de dónde se ponga el `*`.

```
$ ls *.bed
```

```
$ ls nuevos*
```

?

Parecido a ´*´ pero para un sólo carácter.

```
$ ls *.b??
```

[]

```
$ ls [a-z]*.bed
```

Funciones básicas de exploración de archivos con bash

`more`

Nos permite ver el archivo una línea (flecha abajo) o página a la vez (barra espaciadora). Para salir: `q`

(puedes salir con `q` si no quieres escrolear (satamente!, esa palabra no existe en español) todo el archivo para abajo)

`less`

Igual que `more` pero se desarrolló más recientemente y puede abrir archivos binarios y otras cosas raras. Juego de palabras con que *less is more*.

Se recomienda usar `less` en la vida.

`head`

Muestra las primeras líneas de un archivo (default 10).

`tail`

Muestra las últimas líneas de un archivo.

`wc`

Brinda el número de líneas, el número de palabras y el número de caracteres del archivo.

`cat`

Viene de *Concatenate*. Sirve para unir uno detrás de otro varios archivos, o para imprimir todo el contenido de un archivo a la consola.

Es decir, básicamente es como copiar-pegar un archivo al final de otro.

Pregunta: **¿Y si quisiéramos tener el resultado en un archivo nuevo?**

Regular expressions y búsqueda de patrones (`grep`)

¿Qué son las expresiones regulares

Las *expresiones regulares* son una herramienta de búsqueda o búsqueda-remplazo de cadenas de texto acorde a un patrón dado. Existen en la línea de comando, pero también en otros lenguajes, como R y casi cualquier buscador de texto.

Una expresión regular se puede pensar como una combinación de caracteres literales y metacaracteres.

- Los caracteres literales son de los que están formadas las palabras en el lenguaje utilizado. Ejemplo: "c","o","n","a","b","i","o","2","0","6"
- Los metacaracteres son aquellos que tienen una función particular en la expresión regular. Ejemplo: "*","?",".","|","^","$","(",")","[","]"

Las expresiones regulares también se conocen como *regexp*, *regex* o `grep` (global regular expression print), que es el comando que utilizaremos. Pero en realidad `grep` solo es uno de los comandos que las utiliza, es decir hay otros.

¿Para qué sirven?

Las principales aplicaciones de las expresiones regulares en bioinformática son:

- Reformatear archivos de datos. Una se la vive haciendo esto
- Decirle a un algoritmo que realice análisis en ciertas muestras y no otras
- Identificar patrones cortos de ADN en una secuencia, por ejemplo enzimas de restricción o índices.

Formas comunes de usar `grep`:

`grep` a secas:

```
$ grep ">" razasPerros
```

`grep -c`

Para contar en cuántas líneas aparece la expresión de búsqueda

```
$ grep -c ">" razasPerros
```

grep -l

Sólo enlista los archivos donde se encontró la expresión, pero no las líneas.

```
$ grep -l Physalis *.fasta

  tomatesverdes.fasta
```

grep -i

Hace que la búsqueda sea insensible a Mayúsculas/minúsculas.

```
  $ grep -l physalis *.fasta

  $

  $ grep -li physalis *.fasta

  tomatesverdes.fasta
```

`grep -E`

Lee el texto entre comillas como una expresión regular completa, es decir con operadores, cuantificadores y posicionadores. Es útil utilizarlo junto con `-o` para mostrar solo la parte del texto encontrado que cumple con la expresión regular.

```
$ grep -oE "\| \w+ \w+" tomatesverdes.fasta
```

Buena referencia de expresiones regulares aquí

Y buenos ejemplos de cómo usar `grep` aquí

`sed` es particularmente útil para sustituir una expresión regular (como una palabra) por otra.

`awk` es parecido, pero es particularmente útil para archivos con filas y columnas, pues puedes acceder específicamente a ellas.

Redirección con bash

>

>>

Redirige el Standar output (*stdout*) a un archivo en vez de imprimirlo en pantalla.

Nota que si el archivo catejemplo.txt ya existe será borrado por el comando anterior. Si no deseas que esto ocurra sino que el nuevo contenido se agregue al final de un archivo ya existente entonces usa `>>`.

|

Toma el stdout de un comando y lo convierte en el input de otro

```
$ ls | wc -l
```

Loops con bash

Los *for loops* permiten repetir una serie de comandos con diferentes *variables de una lista*.

Sintaxis:

```
for i in list; do
 command1
 command2
 ..
  done
```

"i" puede leerse como "el elemento i de la lista". Y la lista no es más que el conjunto total de las variables que queremos.

Definir variables

Los for loops utilizan *variables* definidas por el usuario, es decir "i" y "perro" en los ejemplos anteriores. Sin embargo, también pueden crearse variables afuera de un for loop, y usarlas para lo que queramos.

Observaciones importantes

- NO debe haber espacios entre el símbolo = y la variable o su valor.
- El nombre de la variable puede ser cualquier cosa que queramos MENOS el nombre de un comando que exista.

Las variables se pueden usar para acortar algo que escribamos muy seguido (como un path o un nombre de archivo largos) y conjuntar con otras variables dentro de un loop.

```
$ ladrido=guau
$ for i in perro perrito perrote; do
> echo El $i hace $ladrido
> done
```


I ♥ #!/bin/bash