Introducción al análisis bioinformático de secuencias NGS

Andrés Pérez-Figueroa (@anpefi)

2018-04-18

Contents

List of Tables

List of Figures

Este documento representa el guión de prácticas para la asignatura **Genómica marina** del **Máster Interuniversitario en Biología Marina“**

Preparación

Uso de una imagen Docker para el desarrollo de las prácticas

Dado que el objetivo de estas prácticas es adquirir unas nociones básicas que permitan desenvolverse con soltura en el análisis de secuencias genómicas, y que para ello es fundamental que se desarrolle en un entorno Linux con las dependencias instaladas, la mejor forma de llevarlas a cabo es mediante un contenedor de Docker. Así, se ha diseñado una image de Docker que contiene todos los programas y datos necesarios para el desarrollo de estas prácticas y los alumnos sólo tienene que encargarse de instalar Docker en su sistema operativo

Instalación de Docker

Antes de nada deberías instalar Docker en tu ordenador, para ello sigue las instrucciones adecuadas para tu sistema operativo.

Windows: https://docs.docker.com/docker-for-windows/install/#download-docker-for-windows

Mac: https://docs.docker.com/docker-for-mac/install/

Ejecutar el contenedor Docker del curso

Ejecutamos docker desde un directorio/carpeta local que utilizaremos para guardar los resultados que obtengamos.

```
docker run -ti -v $PWD:/home/user/workdir anpefi/genomica-marina:0.2
```

en Windows sustituimos $PWD por la ruta completa de la carpeta que queremos usar como directorio local

1 La línea de comandos UNIX

Una interfaz de línea de comandos (en inglés, command-line interface, CLI) es un **método** que permite a los usuarios dar instrucciones a algún programa informático por medio de una línea de texto simple. Por otra parte un *shell* o intérprete de comandos es el programa que provee una interfaz de usuario para acceder a los servicios del sistema operativo, pudiendo ser esta una interfaz de línea de comandos

1.1 Bash

Bash es un programa informático, cuya función consiste en interpretar órdenes. Es una shell de Unix y el intérprete de comandos por defecto en la mayoría de las distribuciones GNU/Linux, además de macOS. También se ha llevado a otros sistemas como Windows y Android. Incorpora características tales como control de procesos, redirección de entrada/salida, listado y lectura de ficheros, protección, comunicaciones y un lenguaje de órdenes para escribir programas por lotes o "scripts". Su nombre es un acrónimo de Bourne-again shell ("shell Bourne otra vez") haciendo un juego de palabras (born-again significa "nacido de nuevo") sobre la Bourne shell (sh), que fue uno de los primeros intérpretes importantes de Unix.

1.2 ¿Por qué usar una Shell?

Hay diferentes motivos para usar una shell en bioinformática y es que generalmente seremos, a la larga, más eficientes en la mayor parte de tareas si trabajamos en una interfaz de línea de comandos que si lo hacemos en una interfaz gŕafica. Estos son algunos de los motivos:

- Tenemos caracteristicas de los lenguages de programación (como loops o variables)
- Los comandos que ejecutamos se pueden empaquetar en scripts, y estos combinar en pipelines
- Autocompletado
- Guarda el historial de comandos ejecutados
- Permite el acceso fácil y estándar a servidores remotos
- Consume menos memoria y permite gestionar archivos enormes que no se pueden abrir en una GUI
- Con algo de práctica es más rápido teclear los comandos que mover el raton en la pantalla para hacer click en diferentes zonas
- Permite el uso de expresiones regulares lo que facilita las operaciones repetitivas

La línea de comandos nos proporciona una representación palabra a palabra de una serie de acciones que son explícitas, distribuíbles, repetibles y automatizables. La capacidad de expresas acciones con palabras (comandos) nos permite encadenar esas acciones de la foma más apropiada ara cada problema. Por otra parte, ejecutar herramientas en la línea de comandos requiere de un aprendizaje adiciona y un conocimiento en más profunidad de cómo funcionan los ordenadores. Sin embargo aprender a desenvolverse en la línea de comandos

(y en la programación en general) no es simplemente aprender una serie de conceptos, si no que es un aprendizaje a pensar de cierto modo, descomponiendo los problemas en pasos muy simples que puedan ser fácilmente resueltos.

1.3 ¿Cuales son los conocimientos mínimos que debería adquirir sobre la Shell?

1. Navegación a través del árbol de directorios
2. Como acceder a archivos localizados en directorios y explorar su contenido
3. Encontrar documentación acerca de los comandos (`man`)
4. Aprender sobre los argumentos que usan los programas y como especificarlos
5. Redirección y *piping*, cómo encadenar la salida de un programa para que sea la entrada de otro

1.4 Consideraciones sobre los sistemas Linux/Unix

1.4.1 El sistema de archivos

La estructura de directorios que sigue Linux es parecida a la de cualquier UNIX. No tenemos una "unidad" para cada unidad física de disco o partición como en Windows, sino que todos los discos duros o de red se montan bajo un sistema de directorios en árbol, y algunos de esos directorios enlazan con estas unidades físicas de disco. Básicamente, todo el contenido está organizado en ficheros dentro de un directorio, que a su ver puede ser parte de otro directorio superior, y así hasta que llegamos al directorio raiz (representado como `/`). En Unix, cada nivel de la jerarquia está separado por un /. Así, si encontramos que un archivo tiene la ruta /home/andres/archivo.txt significa que es un archivo llamado archivo.txt que está en el directorio andres que a su vez está en el directorio home que a su vez está en el directorio raíz (el / del principio). Eso es lo que se conoce como la ruta absoluta (*absolute path*). Por otra parte podemos referirnos a un archivo por su ruta relativa (*relative path*) al directorio actual. Imaginemos que estamos en el directorio /home, entonces podemos dar la ruta relativa al archivo anterior como andres/archivo.txt (Nota como no hay / al principio).

1.4.2 Directorios estándar

En los sistemas UNIX existe un 'Estándar de jerarquía de ficheros' (FHS - Filesystem Hierarchy Standard) que intenta definir unas bases, para que tanto los programas del sistema, como los usuarios y administradores, sepan donde encontrar lo que buscan. A continuacion teneis una lista con los directorios mas importantes del sistema y para que se usan.

/bin Comandos/programas binarios esenciales (cp, mv, ls, rm, etc.).

/boot Ficheros utilizados durante el arranque del sistema.
/dev Dispositivos esenciales, discos duros, terminales, sonido, video, lectores dvd/cd, etc.
/etc Ficheros de configuración utilizados en todo el sistema y que son específicos del ordenador.
/home Directorios personales de inicio de los usuarios. Generalmente, como usuarios, sólo trabajaremos en nuestro directorio personal dentro de este /home
/lib Bibliotecas compartidas esenciales para los binarios de /bin/, /sbin/ y el núcleo del sistema.
/mnt Sistemas montados temporalmente
/media Puntos de montaje para dispositivos de medios como unidades lectoras de discos compactos. {-}
/opt Paquetes de aplicaciones estáticas. En el cluster, este directorio se encuentra montado sobre el head, para albergar una serie de aplicaciones de uso para todos los nodos.
/proc Sistema de ficheros virtual que documenta sucesos y estados del núcleo.
/root Directorio de inicio del usuario root (super-usuario).
/sbin Comandos/programas binarios de administración de sistema.
/tmp Ficheros temporales.
/srv Datos específicos de sitio servidos por el sistema.
/usr Jerarquía secundaria para datos compartidos de solo lectura (Unix system resources). Este directorio puede ser compartido por múltiples ordenadores y no debe contener datos específicos del ordenador que los comparte.
/var Ficheros variables, como son logs, bases de datos, directorio raíz de servidores HTTP y FTP, colas de correo, ficheros temporales, etc.

1.4.3 Nombres de archivos en Linux

Los nombres de archivos en Linux (como en todos los UNIX) distinguen mayúsculas de minúsculas, esto es, son "case sensitive". Los archivos README, readme, REadme y rEadme por ejemplo son archivos distintos y por lo tanto al ser nombres distintos pueden estar en el mismo directorio.

En Linux los archivos no tienen por qué tener una extensión. La suelen tener a modo orientativo, pero no es en absoluto necesario. Linux sabe qué contiene cada archivo independientemente de cuál sea su extensión. Por comodidad, podremos llamar a todos nuestros archivos de texto con la extensión .texto, o a todos nuestros documentos con la extensión .documento, de esta manera, podremos luego agruparlos más fácilmente.

Los ficheros y directorios ocultos en Linux comienzan su nombre por un punto (.)

Los nombres de archivos o directorios pueden ser muy largos, de más de 200 caracteres, lo cual nos da bastante flexibilidad para asociar el nombre de un archivo a lo que contiene. No obstante, hay ciertos caracteres que nunca se deberían utilizar a la hora de nombrar un archivo. Uno de ellos es el espacio, nunca llamaremos a un fichero con un nombre que contenga un espacio. Tampoco son recomendados otros caracteres raros como signos de puntuación (a excepción del punto), acentos o la ñ. En algunos casos Linux ni siquiera nos

permitirá usarlos. Los recomendables son las letras A-Z, a-z, los números (0-9), el punto, el guión (-) y el guión bajo (_) para nombrar un archivo. Los acentos y la ñ tampoco se recomiendan.

1.4.4 Permisos de archivos y directorios

En Linux, todo archivo y directorio (*nota: los directorios son, a efectos técnicos, un tipo especial de archivo*) tiene tres niveles de permisos de acceso: los que se aplican al propietario del archivo, los que se aplican al grupo que tiene el archivo y los que se aplican a todos los usuarios del sistema. Podemos ver los permisos cuando listamos un directorio con ls -l:

```
andres@head:~/curso$ ls -l
total 212
-rwxr--r-- 1 andres users 46801 2009-10-29 16:26 a.out
-rw-r--r-- 1 andres users  1629 2009-10-29 16:26 poblacion.c
drwxr-xr-x 2 andres users  4096 2010-10-02 18:30 prueba
-rwxr-xr-x 1 andres users 47740 2009-10-29 18:05 rec
-rw-r--r-- 1 andres users   711 2009-10-29 16:26 varianza.c
```

Veamos por partes el listado, tomando como ejemplo la primera línea. La primera columna (-rwxr-xr–) es el tipo de archivo y sus permisos, la siguiente columna (1) es el número de enlaces al archivo, la tercera columna (andres) representa al propietario del archivo, la cuarta columna (users) representa al grupo al que pertence al archivo y las siguientes son el tamaño, la fecha y hora de última modificación y por último el nombre delarchivo o directorio.

1.5 Comandos básicos

Vamos a practicar sobre la marcha algunos de los comandos en bash más utizados y útiles. Para ello, abrimos el contenedor Docker de este curso y en la terminal ejecutaremos los comandos que iremos explicando a continuación.

1.5.1 ls

Para empezar un comando muy sencillo que simplemente le pide al sistema operativo que nos de un **lis**tado del contenido de un directorio

```
ls
```

Aquí simplemente hemos escrito el nombre del comando y este nos devuelve por pantalla (salida estándar, *standard output*) una lista de archivos y directorios. Esta es la interacción más básica, ya que sólo hemos usado el nombre del comando y no le hemos dado argumentos para precisar más lo que queremos de él.

1.5.2 man

En sistemas UNIX generalmente los comandos (que son realmente programas ejecutables) suelen venir con una completa documentación que se puede consultar en la terminal mediante el comando `man` (acrónimo de manual). Las páginas de manual son muy importantes ya que nos indican, entre otras cosas, cuales son los argumentos que acepta un comando. Podemos ver cuales son las opciones del comando `ls` viendo su página del manual:

```
man ls
```

Esto nos muestra una completa página de información sobre el comando ls. Por ejemplo nos indica en la SYNOPSIS que la manera de llamar al comando es `ls [OPTION]... [FILE]...` Eso significa que al comando ls le podemos añadir **argumentos** opcionales (al estar indicados entre corchetes indica que son opcionales, como no hay ningún argumento obligatorio hemos podido usar el comando anteriormente sin argumentos) y de dos tipos, opciones y ficheros. Las argumentos de opciones se pueden dar en formato corto (-) o formato largo (–). Por ejemplo `ls -a` es lo mismo que `ls --all` y lo que viene a decir es que nos mostrará el contenido completo del directorio (mostrando los archivos ocultos, que empiezan por "."). Por tanto, si ahora tecleamos (Para salir de la página del manual presiona la tecla "q")

```
ls -a
```

Ejercicio 1.1. ¿Cual es la diferencia con el resultado que nos daba antes?

Otra cosa que nos indica el manual de ls en DESCRIPTION es que `List information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.` Por tanto, si no incluímos un argumento de FILE, nos está devolviendo, por defecto, el contenido del directorio actual, que si hemos lanzado el contendor docker correctamente, nos mostrará el contenido de nuestro directorio local desde el que lo hemos lanzado. Si queremos que nos devuelva el contenido de cualquier otro directorio tenemos que añadir el nombre del directorio como argumento

```
ls -a data
```

Esto nos devuelve el contenido del directorio data.

Los puntos supensivos tras el [OPTION] o el [FILE] nos indican que podemos añadir varios argumentos del mismo tipo. Además, en el caso de las opciones, si usamos los argumentos cortos podemos combinarlos, por ejemplo `ls -a -s home` es lo mismo que `ls -as home` o que usando los argumentos largos `ls --all --size home` (nota la diferencia entre -as, argumentos cortos combinados, y –all, argumento largo, que no se pueden combinar). Una cosa importante en la anterior salida es que aparecen listados dos directorios especiales `.` y `..` que hacen referencia al directorio actual y al directorio anterior (*parental directory*) respectivamente. Esa notación la podemos usar siempre y por ejemplo `ls .` listará el contenido del directorio actual.

Por último, para finalizar con los argumentos, tambien podemos añadir varios argumentos del tipo FILE. Por ejemplo, es muy habitual usar la opción -l (que devuelve información sobre los permisos, fecha y tamaño de los archivos), y podemos usarlo para varios directorios en

una sola llamada:

```
ls -larth . data
```

Ejercicio 1.2. En este ejemplo, ¿qué es lo que nos muestra el comando?¿Qué significa -larth?
Ejercicio 1.3. Repite el comando pero usando rutas absolutas ¿Hay alguna diferencia?

1.5.3 cd

cd viene de **c**hange **d**irectory y, por tanto, nos permite cambiar el directorio actual a cualquier otro dado por su ruta absoluta o relativa.

```
cd data
```

nos moverá al directorio home. Si te fijas, en el prompt (ese mensaje delante de donde escribimos los mensajes) aparecerá ahora `:~/data#`. Si hacemos ls, nos mostrará el contenido de data, no el del directorio /home/user como antes. Otra forma de saber en que directorio nos encontramos es mediante el comando `pwd` (***p**ath to **w**orking **d**irectory*) que nos devolverá la ruta absoluta del directorio actual. Una forma rápida de vovler a nuestro directorio home (/home/user) es utilizar `~` como sinónimo, así `cd ~` nos devuelve al directorio home. Aún más rápido es usar `cd` sin argumentos, que nos devuelve al directorio home.

Si estamos escribiendo una ruta de un archivo, siempre podemos usar la tabulación para autocompletar, si no hay ambiguedad, el nombre del siguiente elemento de la jerarquía. Así, si estamos tecleando `cd /home/u` y pulsamos la tecla <TAB> el comando se auto completará a `cd /home/user/` y si volvemos a dar a <TAB> nos mostrará todos los archivos y directorios para escribir algún caracter que provoque la desambiguación. Este autocompletado es enormemente útil y se aplica cuando usamos archivos como argumento en cualquier comando.
Ejercicio 1.4. ¿Cómo volvemos al directorio anterior?

echo

echo simplemente muestra una línea de texto

```
echo Hola, colega
```

A bote pronto puede parecer un comando innecesario ya que simplemente escribe lo que le decimos, pero resulta muy útil cuando lo usamos de un modo programático, por ejemplo en un script o para mostrar el valor de ciertas variables. Así

```
echo $PATH
```

lo que hace es mostrarnos el valor guardado en la variable PATH (para mostrarla la antecedemos con el signo $). La variable PATH es una importante variable de entorno (Variables del sistema que afectan a todos los programas) que nos muestra una lista (separada por :) de las rutas en las que el sistema va a buscar un ejecutable cuando escribimos un comando. Si en esos directorios no se encuentra el archivo ejecutable nos dará un error si intentamos ejecutarlo sólo con su nombre. Si un ejecutable no está en uno de los directorios del PATH, lo podremos ejecutar llamándolo pr su ruta relativa o absoluta (Nota: si está en el directorio actual la ruta relativa debe incluir este como `./`, por ejemplo `./miprograma`).

1.5.4 mkdir

Otro comado sencillo pero imprescindible es mkdir (***ma****k****e*** ***dir****ectory*) que nos permite crear directorios. Creemos, por ejemplo, un directorio que usaremos como directorio de trabajo

```
cd
mkdir workdir
```

Una opción interensate de mkdir, que podemos ver en su página de manual, es la de hacer directorios intermedios si estos no existe. Imagina que estamos en /home/user/wordir, que está vacio, y queremos crear un directorio nivel3, dentro de otro nivel2 a su vez dentro de uno llamado nivel1. O sea queremos crear el directorio /home/user/workdir/nivel1/nivel2/nivel3. No es necesario ir creando cada directorio y moviéndonos a él (con cd) crear el siguiente, podemos hacerlo todo de una vez con `mkdir -p /home/user/workdir/nivel1/nivel2/nivel3` o, dado que estamos en /home/user/workdir usando la ruta relativa `mkdir -p nivel1/nivel2/nivel3`

1.5.5 touch

De forma similar a mkdir, touch nos permite crear o actualizar archivos. Si ejecutamos

```
touch touched_file
ls -la touched_file
```

podemos comprobar que ha creado un archivo vacio llamado touched_file. En realidad, touch lo que hace es cambiar el *timestamp* del archivo y si no existe lo crea.

1.5.6 rm

rem*ove* sirve para borrar archivos (y directorios, aunque no por defecto). Y aquí borrar significa borrar, no enviar a una "papelera de reciclaje" como en otros sistemas gráficos. Por tanto se debe usar con cuidado y responsabilidad.
Ejercicio 1.5. Consulta el manual de rm y borra los directorios nivelX que hemos creado antes con un solo comando

1.5.7 Un inciso: sustitución de nombres de archivo

Muchos comandos toman como argumentos nombres de archivo como argumento. Estos se pueden citar literalmente uno a uno o bien se pueden especificar de forma genérica. Existen unos caracteres especiales que permiten fabricar expresiones utilizadas como modelos de nombre de archivo.

- El carácter *****. El asterisco representa un conjunto de caracteres cualquiera.
- El carácter **?**. El interrogante representa un único carácter cualquiera

- Los caracteres . Los corchetes permiten especificar la lista de caracteres que se esperan en una posición concreta del nombre de archivo. Dentro de esta lista se pueden usar intervalos (por ejemplo `[a-z]` para especificar cualquier minúscula) o el concepto de negación (representado por **!**, como en '[!a-z] para especificar que esa posición no sea una minúscula).

Veamos unos ejemplos

```
# Volvemos al home
cd
# Listamos el contenido del directorio example1
ls example1
# queremos ver solo los archivos de ejemplo, que se llaman file_example_1 y file_ejemp
# Para ello usamos la expresión *mpl* que significa cualquiere grupo de caracteres seg
ls example1/*mpl*

#Ahora queremos borrar todos los archivos txt
rm example1/*.txt
```

1.5.8 cp

Este comando copia (***cop****y*) un archivo manteniendo el original

```
cp ~/example1/original ~/copia
cp ~/example1/*.txt ~
```

Ejercicio 1.6. ¿Dónde ha creado el fichero copia del primer comando cp?
Ejercicio 1.7. ¿Qué significa el segundo comando cp?¿Cuantos archivos copia?¿Qué nombre les asigna?
Ejercicio 1.8. ¿Cómo copiaríamos el directorio example1 a unos llamado example2?

1.5.9 mv

Este comando mueve archivos (***mov****e*), con lo que el archivo que especifiquemos como destino **sustituirá** al de origen. Vamos a mover el archivo original del directorio example1 a un directorio temporal que crearemos

```
mkdir ~/temp
mv ~/example1/original ~/temp/original
ls ~/example1
ls ~/example2
```

Como vemos, el archivo original ya no esta en example1 si no en `temp`. El comando mv se suele usar mucho para **renombrar archivos** que es el proceso de moverlo a otro nombre. Por ejemplo podemos renombrar el archivo de antes

```
mv ~/temp/original ~/temp/renombrado
```

Como se ve, se puede mover dentro del mismo directorio, no es necesario que se mueva a diferentes directorios, siempre y cuando le demos otro nombre. Desde el punto de vista bioinformático es muy importante destacar que `mv` no equivale a un `cp` + un `rm` y que al hacer un move no estamos duplicando el contenido del archivo (técnicamente solo cambia el índice que indica en que sectores del disco está y cual es su nombre/ruta). Esto es vital saberlo cuando trabajamos con archivos grandes ya que si copiamos un archivo grande (digamos de 50GB) estamos duplicando el espacio utilizado en disco, además de estar utilizando un tiempo considerable para el proceso de copia (el archivo se está replicando entero en otra parte del disco). Sin embargo si movemos el archivo, el contenido este no se está escribiendo en ningún otro sitio del disco con lo cual nunca ocupará más espacio y el proceso es inmediato. Por lo tanto tened en cuenta esto cuando penséis en copiar archivos grandes, ya que generalmente no es necesario y podemos gestionar mejor el asunto con mv o con links simbólicos.

1.5.10 less/more

Ya sabemos como movernos por directorios, crear y borrar archivos, copiarlos, etc. Ahora exploraremos como ver el contenido de un archivo y navegar por él. Tanto `less` como `more` son visores de archivo (sólo permiten ver el contenido, no editarlo) con paginación. Esto es útil para navegar por archivos grandes.

1.5.11 cat

cat es otro visor de archivos, pero a diferencia de more o less, cat muestra todo el contenido del archivo de un tirón. Por tanto no parece un comando muy útil para trabajar con archivos grandes (de hecho es peligroso, ya que puede colgarnos durante un buen tiempo mientras hace el scrolling de todo el fichero), pero tiene una utilidad grande cuando trabajos con pipes (como veremos más adelante)

1.5.12 head/tail

Otros visores muy útiles ya que permiten visaulizar las primeras (head) o las últimas (líneas). Por defecto nos muestra las 10 primeras/últimas líneas, pero tienen una serie de opciones que nos permiten operaciones muy útiles.

1.5.13 wc

Este comando se usa para contar el número de líneas, palabras o caracteres de un archivo. Esto es muy útil para conocer cuanta información tiene un archivo, por ejemplo si tenemos un archivo de texto con una lista de genes, cada uno en una línea, podemos saber cuantos son exáctamente con `wc -l <nombre de archivo>`

1.5.14 El editor de texto nano

Existen muchos editores de texto para línea de comandos (emacs, vi/vim, joe, ..) pero voy a comentaros como usar `nano`, que es un editor muy sencillo, fácil de usar y que cubrirá nuestras necesidades de edición rápida de archivos dentro de la línea de comandos.

Para abrir un archivo (o crearlo si es que no existe) simplemente teclea `nano <nombre_archivo>` y se abrirá una interfaz de edición. Ahora ya puedes editar el archivo, moviéndote con las teclas de cursor. En la parte inferior de la pantalla te muestra los atajos de teclas. Si deseas guardar las modificaciones hechas, presiona Ctrl+O. Para salir de nano, presiona Ctrl+X. Si has pedido salir de un fichero modificado, te preguntará si lo deseas salvar. Presiona N si no quieres salvar el fichero o Y en caso que si quieras. Entonces te pedirá un nombre para el fichero, escríbelo y presiona Enter.

Si por error presionas que quieres guardar el fichero, no te preocupes porque podemos cancelar presionando Ctrl+C, siempre que estés en la pantalla donde se escribe el nombre del fichero.

Para cortar una sola línea, usa Ctrl+K. La línea desaparecerá. Para pegarla, sencillamente mueve el cursor a donde quieras pegar el texto y presiona Ctrl+U. La línea reaparece. Para mover varias línea, córtalas presionando Ctrl+K varias veces y luego pégalas pulsando Ctrl+U una sola vez. El párrafo completo aparecerá donde quiera.

Si necesitas un control un poco más fino, entonces debes marcar el texto. Mueve el cursor al comienzo del texto que quieres cortar. Presiona Ctrl+6 (o Alt+A). Ahora mueve el cursor hasta el final del texto que quiere cortar: el texto debe quedar resaltado. Si quieres cancelar, sencillamente presione Ctrl+6 nuevamente. Pulsa `^K` para cortar el texto marcado. Usa Ctrl+U para pegarlo.

Buscar una palabra es fácil, simplemente presiona Ctrl+W e introduce el término a buscar. Para buscar el mismo término otra vez, presione Alt+W.
Ejercicio 1.9. Crea, usando nano, un archivo en ~/workdir llamado reyes.txt y escribe en él una lista de los últimos 4 reyes de España que recuerdes, con un nombre en cada línea. Guárdalo y sal de nano
Ejercicio 1.10. muestra el contenido de reyes.txt sin posibilidad de editarlo

1.5.15 grep

Este es uno de los comandos más útiles a la hora de trabajar con grandes ficheros de texto. Grep permite buscar texto dentro de archivos. Básicamente toma una expresión regular, la busca en una serie de archivos e imprime las líneas completas que contengan la expresión regular buscada. Por ejemplo, podemos buscar las líneas de todos los archivos de texto que contienen el nombre Felipe:

```
grep "Felipe" *.txt
```

Esto devolverá la línea `Felipe VI` del archivo reyes.txt

Ejercicio 1.11. Pídele (con la ayuda de man) al comando grep que muestre sólo el nombre del archivo donde ecuentre la cadena "Felipe"
Ejercicio 1.12. Pídele al comando grep que muestre la línea donde aparece "Felipe" y la línea inmediatamente posterior

1.5.16 Redirecciones

Una utilidad importante, a estas alturas, de los comandos unix es la redirección de las salidas o entradas estándar. La salida estándar, lo que un comando muestra como resultado, está asociada por defecto a la terminal, pero podemos redireccionarla para que el resultado lo guarde en un archivo en vez de msotrarlo en la terminal. El comando de redirección de la salida estándar a archivo es `>` con lo que si el archivo no existe, se crea, y si existe se sobreescribe. El comando `>>` permite la concatenación, así si el archivo ya existe no lo sobreescribe, si no que añade el resultado al final.

```
echo punto_1 > puntos.txt
cat puntos.txt
echo punto_2 > puntos.txt
cat puntos.txt
echo punto_2 >> puntos.txt
cat puntos.txt
```

Tambien existe la posibilidad de redireccionar la salida de error (los mensajes de error) con `2>` o `2>>`; o de redireccionar la entrada con `<` o de combinar varias de estas redirecciones (p.e. a veces es útil que un comando no nos muestre nada por pantalla: `comando 1> resultados.txt 2> /dev/null`

1.5.17 cut

Sirve para recuperar (cortar) caracteres o campos de una línea

```
cut -c1-3 reyes.txt #Devuelve los tres primeros caracteres de cada línea
cut -c2,6 reyes.txt #Devuelve el segundo y sexto carateres de cada línea
cut -d' ' -f1 reyes.txt
```

Ejercicio 1.13. ¿Qué devuelve el último comando?¿Por qué?

1.5.18 sort

Creamos con nano un archivo llamado cartas.txt con este contenido.

```
sota
caballo
rey
rey
```

```
caballo
sota
```

Ahora podemos ordenarlas con sort:

```
sort cartas.txt > cartas_ordenadas.txt
cat cartas_ordenadas.txt
```

1.5.19 uniq

Este comando permite eliminar/mostrar/contar las líneas repetidas de un archivo. **Solamente porcesa las líneas idénticas que son consecutivas** por eso se suele combinar con sort. Tomemos como ejemplo el fichero cartas.txt que hemos creado antes.

```
uniq cartas.txt
uniq -c cartas.txt
uniq -d cartas.txt
```

Ejercicio 1.14. ¿Qué ha pasado en cada uno de los comandos anteriores?

Ahora hacemos lo mismo para el archivo ordenado con sort:

```
uniq cartas_ordenadas.txt
uniq -c cartas_ordenadas.txt
uniq -d cartas_ordenadas.txt
```

Ejercicio 1.15. ¿Qué diferencia hay con el ejemplo anterior?

1.6 Descargar contenido online con wget

Una tarea que repite en muchas ocasiones es la de descargar ficheros. Esto también se puede hacer desde línea de comandos con wget (u otros comandos como curl, ftp, sftp, rsync que pueden ser más adecuados en algunas ocasiones). Lo único que necesitamos es una URL del archivo a descargar (que se puede obtener con el navegador mediante click derecho y "copiar url").

Vamos a descargar un archivo con las anotaciones genómicas de *Crassotrea gigas* que podemos encontrar en el NCBI.

```
cd ~/workdir #para asegurarnos que estamos en el directorio de trabajo
wget -O oyster.gff.gz ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/297/895/GCF_0002978
ls -larth
```

1.7 Pipelines

Los pipelines o tuberías son secuencias de procesos encadenados juntos mediante sus entradas y salidas estándar, de tal manera que la salida de un proceso se toma directamente como

entrada en el proceso siguiente. Esto se implemente mediante el carácter | y lo que hace básicamente es conectar dos comandos de unix. Un ejemplo:

```
cat reyes.txt | grep "Alfonso" | wc -l
```

Ejercicio 1.16. ¿Qué devuelve ese pipeline?¿Qué significa el valor que devuelve?

1.8 Compresión de archivos

En bioinformática encontraremos ficheros enormes que generalmente tendrán que venir comprimidos. Un archivo comprimido contiene la misma información que el archivo sin comprimir pero ha sido optimizado para su tamaño. Sin embargo necesita ser descomprimido para acceder a su contenido

Por ejemplo, si intentamos hacer un head del fichero que descargamos anteriormente:

```
head oyster.gff.gz
```

Resulta que no entendemos nada. Necesitamos descomprimir su contenido (con gunzip, ya que su extensión es .gz) pero podemos hacerlo "*on the fly*" para agunos formatos (como el gz)

```
head oyster.gff.gz | head
```

Y hay algunos comandos y programas que nos permiten trabajar directamente con los archivos comprimidos (ya que ellos hacen la descompresión internamente). Un ejemplo es `less`

```
less oyster.gff.gz
```

La compresión requiere muchos más recursos computacionales que la descompresión. Siempre es más rápido restaurar un archivo desde una forma comprimida que crear esa forma comprimida. Se pueden comprimir archivos individuales (.gz, .bz, bz2 o zip) o directorios completos (.tar.gz, .tar.bz2, . . .)

1.9 Analisis de datos básico con la Shell

Con los elementos que ya tenemos podemos hacer un análisis básico mediante comandos de la shell. Cojamos como referencia el archivo oyster.gff.gz que descargamos antes. Se trata de un archivo de anotaciones genómicas para *Crassotrea gigas* en formato gff comprimido. Ya veremos en más detalle este formato, pero por ahora nos interesa que fila describe a una secuencia y que en la tercera columna contiene información acerca de las *feature* (si esas secuencias representa un gen, un exon, un intron, una CDS,. . .).

Vamos a intentar averiguar cuantos features hay de cada tipo.

Primero descomprimimos el archivo

```
ls -larth
gunzip oyster.gff
ls -larth
```

Observamos que al descomprimirlo se ha borrado el archivo comprimido, y que descomprimido ocupa mucho más.
Ejercicio 1.17. ¿Cuantas líneas contiene el archivo?

Ahora examinamos las primeras líneas

```
cat oyster.gff | head
```

Vemos que hay líneas que comienzan con # que son comentarios y no nos dan información. Vamos a filtrarlas

```
cat oyster.gff | grep -v -e "^#" |  head
```

Ejercicio 1.18. ¿qué significan los argumentos de grep en la instrucción anterior?

Para evitar que terminos como "gene" o "exon" que puedan aparecer en otras columnas, vamos a quedarnos sólo con la tercera columna que es la que sabemos que contiene las features y es lo que queremos saber

```
cat oyster.gff | grep -v -e "^#" | cut -f3 |  head
```

Ahora ya tenemos una sóla columna con los tipos de features, podemos usar sort y uniq para que nos digan cuantas hay de cada tipo

```
cat oyster.gff | grep -v -e "^#" | cut -f3 |  sort | uniq -c | sort
```

Y así tenemos los datos que queríamos, usando solamente comandos del shell.
Ejercicio 1.19. Intentad obtener la misma tabla mediante Excel o similar ¿Cuanto más se tarda?

1.10 Shell scripting

Los scripts no son más que unos archivos que combinan diferentes comandos y pipelines con el fin de automatizarlos. En su formato más simple consisten en una serie de comandos escritos en líneas de texto. Por ejemplo consideremos el siguiente script guardado en el archivo `hello.sh`

```
#!/bin/bash
# comment
echo Hello World
```

La primera línea es de tipo especial y nos indica que shell debe usar (en este caso bash). La segunda línea, como va precedida de un "#" nos indica que es un comentario, no se va a ejecutar. Es muy importante comentar todos los pasos en un script. La tercera línea ya es un comando que se va a ejecutar.

Para ejecutar el script primero le damos permiso de ejecución `chmod +x hello.sh` y luego lo podemos ejecutar llamándolo por su ruta relativa (o la completa):

```
./hello.sh
```

Ahora podemos escribir un script que automatize un análisis como el que hemos hecho antes para las features de *Crassotrea gigas* y guardarlo en un fichero oyster_features.sh

```
#!/bin/bash
# script para el análisis de features del genoma de Crassostrea gigas

#Creamos un nuevo directorio para este análisis
cd ~/workdir
mkdir oyster
cd oyster

#Obtención de los datos
wget -O oyster.gff.gz ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/297/895/GCF_0002978

#Descomprimimos
gunzip oyster.gff.gz

#Obtenemos la tabla de features
cat oyster.gff | grep -v -e "^#" | cut -f3 |  sort | uniq -c | sort
```

Ahora podemos correr ese script, pero en vez de que nos muestre la tabla de features, que nos la guarde en un fichero llamado oyster_features.txt

```
cd ~/workdir
chmod +x oyster_features.sh
./oyster_features.sh > oyster_features.txt
```

Ahora ya tenemos nuestro análisis básico totalmente automatizado en un script. Tiene la ventaja de que es fácilmente editable para hacer lo mismo con otros ficheros de datos.
Ejercicio 1.20. Modifica el script para hacer el mismo análisis en *Anguilla anguilla* (busca el gff en https://www.ncbi.nlm.nih.gov/genome/10841?genome_assembly_id=59496). ¿Cual es el resultado?¿Qué ha podido pasar?

1.11 Otras consideraciones importantes

Hay una serie de temas y conceptos, que por motivos de tiempo no podemos hablar en profundidad en este curso, pero que suponen un conocimiento transversal importante para el día a día en el análisis bioinformático.

- Otros comandos útiles (sed, awk, find, . . .)
- Expresiones regulares
- Conexión a terminales remotas (ssh, sftp,. . .)

- Gestión de trabajos en sistemas de colas (slurm, sge, . . .)
- Nociones de programación (bash, python, perl, R, . . .)
- Reproducibilidad y comunicación (markdown, Jupyter)
- Control de versiones (git)

2 Representación y generación de secuencias

2.1 Tipos de secuencias

Los formatos de datos de secuencia más comunes son los formatos GenBank, FASTA y FASTQ. Los primeros representan información de secuencia curada. El formato FASTQ, por otro lado, representa datos obtenidos experimentalmente medidos a través de NGS.

El formato GenBank tiene la ventaja de ser un formato genérico que puede representar una amplia variedad de información mientras se mantiene esta información legible por humanos. Y ese era su propósito. Por esta misma razón, sin embargo, no está optimizado para ninguna tarea de análisis de datos en particular. Por lo tanto, se usa principalmente para intercambiar información y casi nunca es adecuado como entrada para un protocolo de análisis.

2.2 Formato FASTA

El formato FASTA es el caballo de batalla de la bioinformática siendo un standard *de facto*. Representa la información de las secuencias en su forma más simple, con una línea de identificación de la secuencia (marcada por el caracter `>`) y el resto de líneas con la secuencia, construída con un alfabeto que corresponda a alguna entidad biologógica (ADN, ARN, Proteinas,. . .)

```
>identificador Y cualquier otro texto adicional en la línea de identificacion
CATGTCAGTCANCGACTGATCGATCGTACGTAGCTAGCTACGTAGCTACGATCGATCGCTAGCT
AGTCGATCAGTCGATCGTAGCTACGTACGACTACATTAGCGCCGCTCG-GCGCGGCGGCCATAT
```

Ejercicio 2.1. En la secuencia anterior, ¿cuales son todos los carácteres presentes y que significan?
Ejercicio 2.2. Examina el archivo ~/data/example.fa ¿a qué tipo molécula corresponden las secuencias?¿cuantas secuencias diferentes hay en el archivo?
Ejercicio 2.3. Descarga el archivo FASTA del genoma completo de la bacteria *Prochlococcus marinus* (ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/007/925/GCF_000007925.1_ASM792v1/GCF_000007925.1_ASM792v1_genomic.fna.gz) y obten el tamaño del genoma del mismo.

Sin embargo la simplicidad del formato FASTA y la falta de reglas estrictas pueden ser un problema cuando se usan en diferentes programas, ya que estos podrían no gestionar bien archivos FASTA con secuencias muy largas, con alfabetos extendidos, con líneas de diferente

tamaño o con diferencias entre mayúsculas y minúsculas. Es por ello que hay que leer bien la documentación del programa que usaremos para analizar nuestras secuencias FASTA.

2.3 Formato FASTQ

El formato FASTQ es el estándar de facto que usan los secuenciadores NGS para exponer las lecturas. Puede ser pensado como una variante del formato FASTA que le permite asociar una medida de calidad a cada secuencia base (FASTa con Qualities). En términos más simples, es un formato en el que para cada base, asociamos una medida de confiabilidad: la base es A y la probabilidad de que estemos equivocados es 1/1000. Conceptualmente, el formato es muy similar al FASTA pero sufre de aún más defectos que el formato FASTA.

Cada secuencia en un archivo FASTQ consiste de 4 líneas

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

La primera línea, que empieza con una @ es el identificador de secuencia. La segunda línea consiste en la secuencia de nucleótidos en si misma. La tercera línea empieza con un carácter '+' y, opcionalmente, puede ir seguida del mismo identificador de secuencia de la primera línea. Finalmente, la cuarta línea codifica los valores de calidad de la secuencia de la segunda línea, y debe contener el mismo número de caracteres que esta.

El la línea que codifica la calidad encontramos una serie de caracteres ASCII en los que cada uno representa un valor decimal de calidad, el *Phred score* (entre 0 y 40), algo que veremos en el siguiente apartado. Ordenados de menor a mayor estos son los caracteres ASCII usados para representar calidad en FASTQ:

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI
|    |    |    |    |    |    |    |    |
0....5...10...15...20...25...30...35...40
```

Dependiendo del tipo y versión de plataforma de secuenciación se toman diferentes caracteres :

2.4 Base Quality scores

El *Phred score* $\boldsymbol{Q}$ se usa para calcular la probabilidad de una base sea asignada de forma incorrecta mediante la fórmula

$$P = 10^{-}Q/10$$

. Esto se resume en

Q	Error	Accuracy
0	1 en 1	0%

Q	Error	Accuracy
10	1 en 10	90%
20	1 en 100	99%
30	1 en 1000	99.9%
40	1 en 10,000	99.99%

2.5 Evaluación de la calidad

Guía sobre uso e interpretación de FastQC

3 Formatos de archivo de mapeados y variantes

3.1 Formato SAM/BAM

Un fichero SAM (Sequence Alignment/Map format) es un fichero de texto tabulado para la representación de alineamientos de secuencias contra un genoma o secuencia de referencia (mapeos). Está compuesto por una sección de cabecera (opcional) y una sección de alineamiento.

Definición formato SAM/BAM

- Sección de encabezado: La sección del encabezado no es obligatoria, pero la mayoría de los softwares NGS lo requieren. Cada línea de la sección de encabezado comienza con '@' y un código de tipo de registro de dos letras. Contiene información sobre cinco temas principales: - archivo de alineamiento: versión de formato, clasificación; - secuencia(s) de referencia: p.e. nombre, longitud, especie, url; - grupo de lectura: carril de secuenciación, muestra, centro de secuenciación, libreria, etc .; - programa: nombre y versión del alineador, parámetros utilizados para la alineación; - comentarios personalizados.

- Sección de alineamiento: Cada lectura (read) en el alineamiento (y, a veces, lecturas no alineadas) está representada por una fila que consta de campos delimitados por tabulaciones (básicamente columnas). Si una lectura se mapea a más de una ubicación, cada asignación tendrá su propia fila en el archivo sam. Hay 11 campos obligatorios en cada fila (en las primeras 11 columnas): - nombre de la lectura - indicador (FLAG) bit a bit (codifica información sobre la lectura, por ejemplo, mapeada / no asignada, emparejada / no emparejada, mapeada hacia adelante / atrás, etc.) - nombre de la secuencia de referencia - posición inicial de las lecturas mapeadas en la secuencia de referencia - Calidad del alineamiento - cadena CIGAR (esto es básicamente una breve descripción de la alineación) - nombre de referencia para el *mate read* (para datos emparejados) - posición del *mate read* (para datos emparejados) - distancia entre

lecturas emparejadas (para datos emparejados) - secuencia de nucleótidos de la lectura - calidad por base de la lectura

```
@HD VN:1.5 SO:coordinate
@SQ SN:ref LN:45
r001 99 ref 7 30 8M2I4M1D3M = 37 39 TTAGATAAAGGATACTG *
r002 0 ref 9 30 3S6M1P1I4M * 0 0 AAAAGATAAGGATA *
r003 0 ref 9 30 5S6M * 0 0 GCCTAAGCTAA * SA:Z:ref,29,-,6H5M,17,0;
r004 0 ref 16 30 6M14N5M * 0 0 ATAGCTTCAGC *
r003 2064 ref 29 17 6H5M * 0 0 TAGGC * SA:Z:ref,9,+,5S6M,30,1;
r001 147 ref 37 30 9M = 7 -39 CAGCGGCAT * NM:i:1
```

El formato BAM contiene la misma información que los SAM pero comprimida en forma binaria y generalmente ordenada.

3.2 Formato gff

Las anotaciones genómicas se guardan en ficheros gff. Especificaciones

3.3 Archivos VCF

Variant Calling Format. En este tipo de archivos, generados por *variant callers* se almacenan sólo las varaintes respecto al genoma de referencia.

```
##fileformat=VCFv4.0
##fileDate=20110705
##reference=1000GenomesPilot-NCBI37
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=.,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS    ID        REF  ALT     QUAL FILTER INFO                              FORMA
2      4370   rs6057    G    A       29   .      NS=2;DP=13;AF=0.5;DB;H2           GT:GQ
2      7330   .         T    A       3    q10    NS=5;DP=12;AF=0.017               GT:GQ
2      110696 rs6055    A    G,T     67   PASS   NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ
```

```
2      130237 .         T    .       47   .      NS=2;DP=16;AA=T                GT:GQ
2      134567 microsat1 GTCT G,GTACT 50   PASS   NS=2;DP=9;AA=G                 GT:GQ
```

Especificación VCF

Bibliografía

- El Arte del Terminal
- Data Science at the Command Line
- The Biostar Handbook: A Beginner's Guide to Bioinformatics
- Bash One-liners for Bioinformatics
- Linux Cheat Sheet
- Command Line Tools for Genomic Data Science (curso online)