

Sesión 1. Introducción al biocómputo en sistemas GNU/Linux

Pablo Vinuesa, Centro de Ciencias Genómicas - UNAM

2019-08-11

Contents

1	Presentación	2
1.1	Licencia y términos de uso	2
1.2	Referencias adicionales	2
2	Navegación del sistema y operaciones básicas - primer contacto con un sistema Linux	2
2.1	Conexión a un servidor y exploración de sus características básicas	2
2.2	Exploración del sistema de archivos	3
2.3	Moviéndonos por el sistema de archivos: comando cd	6
2.4	Generación de directorios: comando mkdir	7
2.5	Copiar, mover, renombrar y borrar archivos con: cp, mv y rm	8
2.6	Generación de ligas simbólicas a archivos: comando ln -s /ruta/al/archivo/fuente nombre_la_liga	9
2.7	Visualización de contenidos de archivos: comando head, tail, cat, less, more	9
2.8	Edición de archivos con los editores vim o [g n]edit	11
2.9	Edición de archivos con el editor de flujo sed (stream editor)	12
2.10	Uso de tuberías de herramientas UNIX/Linux para filtrado de texto con cut, grep, sort, uniq, wc y 	12
2.11	redireccionado de la salida STDOUT a un archivo con el comando >	15
2.12	Manual de cada comando: man command	15
2.13	Ayuda de cada comando: command -help	16
3	Inicios de programación en Bash	17
3.1	Asignación y uso de variables	17
3.2	Condicionales	18
3.3	Bucles for	20
3.4	Bash scripts - primeros pasos	22
4	El lenguaje de procesamiento de patrones AWK	25
4.1	Estructura de los programas AWK	26
4.2	Ejemplos básicos pero muy útiles de uso de AWK	27
5	Ejercicios integrativos de uso de herramientas de filtrado del Shell	28
5.1	Filtrado de archivos separados por tabuladores (tablas) con AWK y su graficado con R	28
5.2	Ejercicios de exploración y parseo de archivos FASTA	31
5.3	Solución a la práctica y un ejercicio adicional	32
6	Reto de programación - ejercicio de parseo de archivos FASTA	36
6.1	Inspección y estadísticas básicas de las secuencias descargadas	36
6.2	Edición de las cabeceras FASTA mediante herramientas de filtrado de UNIX	36

1 Presentación

Este apunte fue creado para el Taller 3 - Análisis comparativo de genomas microbianos: Pangenómica y filoinformática de los Talleres Internacionales de Bioinformática - TIB2019, celebrados en el Centro de Ciencias Genómicas de la Universidad Nacional Autónoma de México, del 29 de julio al 2 de agosto de 2019 por Pablo Vinuesa, CCG-UNAM.

Las versión actual (posteriores a 2 de Agosto de 2019) contiene extensiones y adaptaciones para el Taller de Ciencias Genómicas: de Moléculas a Ecosistemas, que impartimos investigadores del CCG-UNAM para alumnos de la Facultad de Ciencias - UNAM.

Version: 2019-08-11

1.1 Licencia y términos de uso

El material del T3, TIB-filoinfo lo distribuyo públicamente a través de este repositorio GitHub bajo la **Licencia No Comercial Creative Commons 4.0**

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0

1.2 Referencias adicionales

Una vez que domines los comandos básicos que se presentarán seguidamente, recomiendo revisar tutoriales mucho más detallados y completos como los siguientes:

- The Linux Command Line - a complete introduction. William E. Shotts, Jr. No Starch Press
- Bash Reference Manual
- Advanced Bash Scripting Guide
- Bioinformatics Data Skills: Reproducible and Robust Research with Open Source Tools. Vince Buffalo. O'Reilly Media 2014

2 Navegación del sistema y operaciones básicas - primer contacto con un sistema Linux

2.1 Conexión a un servidor y exploración de sus características básicas

Estas prácticas están diseñadas para correr en un servidor remoto, pero puedes hacerlo también en una sesión local, es decir, en tu máquina. Sólo tienes que poner en un directorio los archivos con los que vamos a trabajar, los cuales puedes descargar del directorio `sesion1_intro2linux/data` del sitio GitHub

2.1.1 `ssh` establecer sesion remota encriptada (segura) via `ssh` al servidor con número dado de IP

```
ssh -l $USER IP
```

2.1.2 `hostname` muestra el nombre del host (la máquina a la que estoy conectado) y la IP

```
hostname
hostname -i
```

```
## alisio
## 127.0.1.1
```

2.1.3 uname muestra el sistema operativo del host

```
uname
uname -a
```

```
## Linux
## Linux alisio 4.15.0-55-generic #60-Ubuntu SMP Tue Jul 2 18:22:20 UTC 2019 x86_64 x86_64 x86_64 GNU/L
```

2.1.4 top o htop muestran los procesos en ejecución y los recursos que consumen

```
# sales con q o CTRL-c
htop
```

2.2 Exploración del sistema de archivos

2.2.1 pwd imprime la ruta absoluta del directorio actual

```
# dónde me encuentro en el sistema?
pwd
```

```
## /home/vinuesa/Cursos/Taller_CG-FC/sesion1_intro2linux
```

2.2.2 ls lista contenidos del directorio

```
# Qué contiene el directorio actual?
ls | head
```

```
## assembly_summary.txt.gz
## fetch_recA_bradys_vinuesa_nuccion_screenshot.png
## intro_biocomputo_Linux_pt1.odp
## Intro_biocomputo_Linux_pt1.pdf
## linux_basic_commands.tab
## linux_commands.tab
## linux_very_basic_commands_table.csv
## pics
## recA_Bradyrhizobium_vinuesa.fna
## recA_Byuanmingense.fna
```

```
# mostrar todos (-a all) los archivos, incluidos los ocultos, y sus propiedades
ls -al
```

```
## total 17488
## drwxr-xr-x 5 vinuesa vinuesa    4096 ago 11 20:50 .
## drwxr-xr-x 3 vinuesa vinuesa    4096 ago  8 18:05 ..
## -rw-r--r-- 1 vinuesa vinuesa 6780296 ago  8 18:05 assembly_summary.txt.gz
## lrwxrwxrwx 1 vinuesa vinuesa    53 ago  9 20:00 fetch_recA_bradys_vinuesa_nuccion_screenshot.png -
```

```
## -rw-r--r-- 1 vinuesa vinuesa 6280842 ago 9 09:42 intro_biocomputo_Linux_pt1.odp
## -rw-r--r-- 1 vinuesa vinuesa 3128839 ago 8 18:05 Intro_biocomputo_Linux_pt1.pdf
## -rwxr-xr-x 1 vinuesa vinuesa 10193 ago 8 18:05 linux_basic_commands.tab
## lrwxrwxrwx 1 vinuesa vinuesa 78 ago 11 20:50 linux_commands.tab -> /home/vinuesa/Cursos/TIB/TIB
## -rwxr-xr-x 1 vinuesa vinuesa 1705 ago 8 18:05 linux_very_basic_commands_table.csv
## drwxr-xr-x 3 vinuesa vinuesa 4096 ago 9 20:00 pics
## -rw-r--r-- 1 vinuesa vinuesa 77803 ago 8 20:39 recA_Bradyrhizobium_vinuesa.fna
## -rw-r--r-- 1 vinuesa vinuesa 17704 ago 11 20:50 recA_Byuanmingense.fna
## drwxr-xr-x 2 vinuesa vinuesa 4096 ago 8 20:14 recA_seq_data
## -rwxr-xr-x 1 vinuesa vinuesa 8 ago 11 20:46 .Rhistory
## drwxr-xr-x 3 vinuesa vinuesa 4096 ago 8 18:33 TIB2019-T3
## -rwxr-xr-x 1 vinuesa vinuesa 6047 ago 8 18:05 working_with_linux_commands.code
## -rw-r--r-- 1 vinuesa vinuesa 1120635 ago 11 20:50 working_with_linux_commands.html
## -rw-r--r-- 1 vinuesa vinuesa 378222 ago 11 20:20 working_with_linux_commands.pdf
## -rw-r--r-- 1 vinuesa vinuesa 53305 ago 11 20:50 working_with_linux_commands.Rmd
```

2.2.3 file nos indica las propiedades de un archivo

mostrar las características de los archivos con file

```
file .Rhistory
file assembly_summary.txt.gz
file linux_commands.tab
file TIB2019-T3
```

```
## .Rhistory: ASCII text
## assembly_summary.txt.gz: gzip compressed data, was "assembly_summary.txt", last modified: Mon Jul 22
## linux_commands.tab: symbolic link to /home/vinuesa/Cursos/TIB/TIB19-T3/sesion1_intro2linux/linux_bas
## TIB2019-T3: directory
```

2.2.3.1 Veamos el contenido del directorio raiz

- corramos un *ls* sin argumentos

```
ls / | head
```

```
## bin
## boot
## cdrom
## dev
## etc
## home
## initrd.img
## initrd.img.old
## lib
## lib32
```

- mucha más información obtenemos con el formato largo de *ls*: *ls -l*

```
ls -l / | head -20
```

```
## total 2097260
## drwxr-xr-x 2 root root 4096 ago 9 09:09 bin
## drwxr-xr-x 4 root root 4096 ago 9 09:09 boot
## drwxr-xr-x 2 root root 4096 may 30 15:02 cdrom
## drwxr-xr-x 20 root root 4420 ago 11 17:41 dev
```

```
## drwxr-xr-x 151 root root      12288 ago  9 09:09 etc
## drwxr-xr-x  5 root root      4096 jul 19 21:46 home
## lrwxrwxrwx  1 root root        33 jul 24 17:51 initrd.img -> boot/initrd.img-4.15.0-55-generic
## lrwxrwxrwx  1 root root        33 jul 24 17:51 initrd.img.old -> boot/initrd.img-4.15.0-54-generic
## drwxr-xr-x 22 root root      4096 jun 26 17:04 lib
## drwxr-xr-x  2 root root      4096 jun 18 16:05 lib32
## drwxr-xr-x  2 root root      4096 abr 26 2018 lib64
## drwx----- 2 root root     16384 may 30 15:01 lost+found
## drwxr-xr-x  3 root root      4096 jun 26 18:32 media
## drwxr-xr-x  2 root root      4096 abr 26 2018 mnt
## drwxr-xr-x  4 root root      4096 ago  3 10:27 opt
## dr-xr-xr-x 360 root root        0 ago 11 17:39 proc
## drwx----- 5 root root      4096 jul  4 12:57 root
## drwxr-xr-x 35 root root      1100 ago 11 17:45 run
## drwxr-xr-x  2 root root     12288 ago  9 09:09 sbin
```

2.2.3.2 Veamos las primeras 5 entradas y últimas 5 del directorio /bin

```
ls /bin | head -5
```

```
## bash
## brltty
## bunzip2
## busybox
## bzip2
```

idem, pero con detalles de permisos etc

```
ls -l /bin | tail -5
```

```
## -rwxr-xr-x 1 root root      2131 abr 27 2017 zforce
## -rwxr-xr-x 1 root root      5938 abr 27 2017 zgrep
## -rwxr-xr-x 1 root root      2037 abr 27 2017 zless
## -rwxr-xr-x 1 root root      1910 abr 27 2017 zmore
## -rwxr-xr-x 1 root root      5047 abr 27 2017 znew
```

idem, pero ordenando los archivos por fechas de modificacion (-t), listando los mas recientes al final

```
ls -ltr /bin | head -20
```

```
## total 12480
## -rwxr-xr-x 1 root root        89 abr 26 2016 red
## -rwxr-xr-x 1 root root     51512 abr 26 2016 ed
## -rwxr-xr-x 1 root root     14328 ago 11 2016 ulockmgr_server
## -rwsr-xr-x 1 root root     30800 ago 11 2016 fusermount
## -rwxr-xr-x 1 root root     40056 abr 21 2017 efibootmgr
## -rwxr-xr-x 1 root root     18424 abr 21 2017 efibootdump
## -rwxr-xr-x 1 root root     35512 abr 21 2017 setfacl
## -rwxr-xr-x 1 root root     23160 abr 21 2017 getfacl
## -rwxr-xr-x 1 root root     14328 abr 21 2017 chacl
## -rwxr-xr-x 1 root root      5047 abr 27 2017 znew
## -rwxr-xr-x 1 root root      1910 abr 27 2017 zmore
## -rwxr-xr-x 1 root root      2037 abr 27 2017 zless
## -rwxr-xr-x 1 root root      5938 abr 27 2017 zgrep
## -rwxr-xr-x 1 root root      2131 abr 27 2017 zforce
## -rwxr-xr-x 1 root root       140 abr 27 2017 zfgrep
## -rwxr-xr-x 1 root root       140 abr 27 2017 zegrep
## -rwxr-xr-x 1 root root      5764 abr 27 2017 zdiff
```

```
## -rwxr-xr-x 1 root root 1777 abr 27 2017 zcmp
## -rwxr-xr-x 1 root root 1937 abr 27 2017 zcat
```

2.2.4 Expansión de caracteres con * y ?

```
# lista los archivos en /bin que empiezan por las letras b y c
ls /bin/b*
ls /bin/c*
```

```
## /bin/bash
## /bin/brlatty
## /bin/bunzip2
## /bin/busybox
## /bin/bzcat
## /bin/bzcmp
## /bin/bzdiff
## /bin/bzegrep
## /bin/bzexe
## /bin/bzfgrep
## /bin/bzgrep
## /bin/bzip2
## /bin/bzip2recover
## /bin/bzless
## /bin/bzmore
## /bin/cat
## /bin/chacl
## /bin/chgrp
## /bin/chmod
## /bin/chown
## /bin/chvt
## /bin/cp
## /bin/cpio
```

```
# lista los archivos en /bin que empiezan por la letra c seguida de uno o dos caracteres más
ls /bin/c?
ls /bin/c??
```

```
## /bin/cp
## /bin/cat
```

2.3 Moviéndonos por el sistema de archivos: comando cd

2.3.1 de nuevo, ¿dónde estoy?: imprime directorio actual con pwd

```
pwd
```

```
## /home/vinuesa/Cursos/Taller_CG-FC/sesion1_intro2linux
```

2.3.2 sube un directorio usando RUTA RELATIVA: cd ..

```
cd ..
```

- donde estoy?

```
pwd
```

```
## /home/vinuesa/Cursos/Taller_CG-FC/sesion1_intro2linux
```

2.3.3 regresa a tu home con cd

```
cd $HOME
```

```
# que es equivalente a:
```

```
cd ~
```

```
# o también a
```

```
cd
```

- cd cambiar directorios con rutas absolutas (/ruta/completa/al/dir) y relativas ../../

```
# a dónde nos lleva este comando?
```

```
cd /
```

```
pwd
```

```
## /
```

- cambia de nuevo a tu home

```
cd
```

```
pwd
```

```
## /home/vinuesa
```

- sube al directorio home/ usando la ruta relativa

```
cd ../
```

- y lista los contenidos

```
ls | head
```

```
## assembly_summary.txt.gz
## fetch_recA_bradys_vinuesa_nuclide_screenshot.png
## intro_biocomputo_Linux_pt1.odp
## Intro_biocomputo_Linux_pt1.pdf
## linux_basic_commands.tab
## linux_commands.tab
## linux_very_basic_commands_table.csv
## pics
## recA_Bradyrhizobium_vinuesa.fna
## recA_Byuanmingense.fna
```

2.4 Generación de directorios: comando mkdir

```
# vamos a $HOME y generamos el directorio TIB2019-T3
```

```
cd
```

```
if [ -d TIB2019-T3 ]; then
```

```
    echo "found dir TIB2019-T3"
```

```
else
```

```
mkdir TIB2019-T3
fi
```

```
## found dir TIB2019-T3
```

- comprueba los **permisos** del nuevo directorio

```
ls -ld TIB2019-T3
```

```
## drwxr-xr-x 3 vinuesa vinuesa 4096 ago  8 18:33 TIB2019-T3
```

- generemos un subdirectorio por debajo del que acabamos de crear:

```
mkdir -p TIB2019-T3/sesion1_linux && cd TIB2019-T3/sesion1_linux
```

- cambiamos a /home/vinuesa e intenta crear estos mismos directorios ahí

```
cd /home/vinuesa && mkdir -p TIB2019-T3/sesion1_linux
```

No puedes escribir en mi directorio, porque no te he otorgado permiso para ello ;)

2.5 Copiar, mover, renombrar y borrar archivos con: cp, mv y rm

```
# cambia a tu home, y luego a TIB2019-T3/sesion1_linux
cd && cd TIB2019-T3/sesion1_linux
```

2.5.1 copia de archivo simple: cp file .

- copia el archivo /space31/PIG/vinuesa/TIB2019-T3/sesion1_Linux/data/linux_basic_commands.tab al directorio actual

```
cp /space31/PIG/vinuesa/TIB2019-T3/sesion1_Linux/data/linux_basic_commands.tab . # <<< vean el punto, s
```

- otra manera, usando rutas absolutas y la variable de ambiente \$HOME

```
cp /space31/PIG/vinuesa/TIB2019-T3/sesion1_Linux/data/linux_basic_commands.tab $HOME/TIB2019-T3/sesion1
```

2.5.2 copiado de directorio: cp -r dir .

- copiar el directorio /space31/PIG/vinuesa/TIB2019-T3/sesion1_Linux/data/ a tu dir actual

```
# Noten el punto '.' y cp -r (recursively), necesario para copiar directorios completos
```

```
cp -r /space31/PIG/vinuesa/TIB2019-T3/sesion1_Linux/data .
```

2.5.3 Eliminar un directorio: rm -rf [recursively -r and force -f]

```
mkdir borrame
```

```
cp linux_basic_commands.tab borrame
```

```
ls borrame
```

```
rm -rf borrame
```

```
## linux_basic_commands.tab
```


Prueba ahora este comando

```
rm data
```

qué pasa?

¿Cómo tengo que borrar un directorio? `rm -rf directorio`

```
rm -rf data
```

2.6 Generación de ligas simbólicas a archivos: comando `ln -s /ruta/al/archivo/fuente nombre_la_liga`

Esto es muy importante, ya que permite ahorrar mucho espacio en disco al evitar la multiplicación de copias físicas en el disco duro del mismo archivo en el \$HOME de uno o más usuarios

```
hostn=$(hostname)
if [ "$hostn" == "Tenerife" ]; then
    ln -s /home/vinuesa/Cursos/OMICAS-UAEM_genomica/clase1_intro2linux/linux_basic_commands.tab comandos
elif [ "$hostn" == "buluc" ]; then
    ln -s /home/vinuesa/Cursos/TIB/TIB2019-T3/sesion1_linux/data/linux_basic_commands.tab comandos_de_linux
elif [ "$hostn" == "alisio" ]; then
    ln -s /home/vinuesa/Cursos/TIB/TIB19-T3/sesion1_intro2linux/linux_basic_commands.tab comandos_de_linux
elif [ "$hostn" == "bonampak" ]; then
    ln -s /space31/PIG/vinuesa/TIB2019-T3/sesion1_Linux/linux_basic_commands.tab comandos_de_linux.tab
    ln -s /space31/PIG/vinuesa/TIB2019-T3/sesion1_Linux/data/assembly_summary.txt.gz .
fi

# confirmamos que se generaron las ligas
ls -l | head
```

```
## total 17480
## -rw-r--r-- 1 vinuesa vinuesa 6780296 ago  8 18:05 assembly_summary.txt.gz
## lrwxrwxrwx 1 vinuesa vinuesa      78 ago 11 20:50 comandos_de_linux.tab -> /home/vinuesa/Cursos/TIB/
## lrwxrwxrwx 1 vinuesa vinuesa      53 ago  9 20:00 fetch_recA_bradys_vinuesa_nuccore_screenshot.png -
## -rw-r--r-- 1 vinuesa vinuesa 6280842 ago  9 09:42 intro_biocomputo_Linux_pt1.odp
## -rw-r--r-- 1 vinuesa vinuesa 3128839 ago  8 18:05 Intro_biocomputo_Linux_pt1.pdf
## -rwxr-xr-x 1 vinuesa vinuesa  10193 ago  8 18:05 linux_basic_commands.tab
## lrwxrwxrwx 1 vinuesa vinuesa      78 ago 11 20:50 linux_commands.tab -> /home/vinuesa/Cursos/TIB/TIB
## -rwxr-xr-x 1 vinuesa vinuesa   1705 ago  8 18:05 linux_very_basic_commands_table.csv
## drwxr-xr-x 3 vinuesa vinuesa   4096 ago  9 20:00 pics
```

2.6.1 renombramos la liga (o cualquier archivo o directorio)

```
mv comandos_de_linux.tab linux_commands.tab
```

2.7 Visualización de contenidos de archivos: comando `head`, `tail`, `cat`, `less`, `more`

2.7.1 uso de `head` y `tail` para desplegar la cabecera y cola de archivos

```
head linux_commands.tab
```

```
## IEEE Std 1003.1-2008 utilities Name Category Description First appeared
## admin SCCS Create and administer SCCS files PWB UNIX
## alias Misc Define or display aliases
## ar Misc Create and maintain library archives Version 1 AT&T UNIX
## asa Text processing Interpret carriage-control characters System V
## at Process management Execute commands at a later time Version 7 AT&T UNIX
## awk Text processing Pattern scanning and processing language Version 7 AT&T UNIX
## basename Filesystem Return non-directory portion of a pathname; see also dirname Version 7 AT&T UNIX
## batch Process management Schedule commands to be executed in a batch queue
## bc Misc Arbitrary-precision arithmetic language Version 6 AT&T UNIX
```

```
tail linux_commands.tab
```

```
## val SCCS Validate SCCS files System III
## vi Text processing Screen-oriented (visual) display editor 1BSD
## wait Process management Await process completion Version 4 AT&T UNIX
## wc Text processing Line, word and byte or character count Version 1 AT&T UNIX
## what SCCS Identify SCCS files PWB UNIX
## who System administration Display who is on the system Version 1 AT&T UNIX
## write Misc Write to another user's terminal Version 1 AT&T UNIX
## xargs Shell programming Construct argument lists and invoke utility PWB UNIX
## yacc C programming Yet another compiler compiler PWB UNIX
## zcat Text processing Expand and concatenate data 4.3BSD
```

le podemos indicar el numero de lineas a desplegar

```
head -3 linux_commands.tab
```

```
## IEEE Std 1003.1-2008 utilities Name Category Description First appeared
## admin SCCS Create and administer SCCS files PWB UNIX
## alias Misc Define or display aliases
```

```
tail -1 linux_commands.tab
```

```
## zcat Text processing Expand and concatenate data 4.3BSD
```

2.7.2 cat despliega uno o más archivos, concatenándolos

```
cat linux_commands.tab | head
```

```
## IEEE Std 1003.1-2008 utilities Name Category Description First appeared
## admin SCCS Create and administer SCCS files PWB UNIX
## alias Misc Define or display aliases
## ar Misc Create and maintain library archives Version 1 AT&T UNIX
## asa Text processing Interpret carriage-control characters System V
## at Process management Execute commands at a later time Version 7 AT&T UNIX
## awk Text processing Pattern scanning and processing language Version 7 AT&T UNIX
## basename Filesystem Return non-directory portion of a pathname; see also dirname Version 7 AT&T UNIX
## batch Process management Schedule commands to be executed in a batch queue
## bc Misc Arbitrary-precision arithmetic language Version 6 AT&T UNIX
```

cat -n nos permite añadir números de línea a los archivos desplegados

```
cat -n linux_commands.tab | head
```

```
##      1 IEEE Std 1003.1-2008 utilities Name Category Description First appeared
##      2 admin SCCS Create and administer SCCS files PWB UNIX
##      3 alias Misc Define or display aliases
```

```
##      4  ar  Misc      Create and maintain library archives      Version 1 AT&T UNIX
##      5  asa   Text processing      Interpret carriage-control characters      System V
##      6  at   Process management  Execute commands at a later time      Version 7 AT&T UNIX
##      7  awk   Text processing      Pattern scanning and processing language      Version 7 AT&T UNIX
##      8  basename  Filesystem  Return non-directory portion of a pathname; see also dirname      Ver
##      9  batch  Process management  Schedule commands to be executed in a batch queue
##     10  bc  Misc      Arbitrary-precision arithmetic language      Version 6 AT&T UNIX
```

2.7.3 el paginador less despliega archivos página a página

```
less linux_commands.tab | head
```

```
## IEEE Std 1003.1-2008 utilities Name  Category  Description  First appeared
## admin  SCCS      Create and administer SCCS files      PWB UNIX
## alias  Misc      Define or display aliases
## ar     Misc      Create and maintain library archives      Version 1 AT&T UNIX
## asa    Text processing      Interpret carriage-control characters      System V
## at     Process management  Execute commands at a later time      Version 7 AT&T UNIX
## awk    Text processing      Pattern scanning and processing language      Version 7 AT&T UNIX
## basename  Filesystem  Return non-directory portion of a pathname; see also dirname      Version 7 AT
## batch   Process management  Schedule commands to be executed in a batch queue
## bc     Misc      Arbitrary-precision arithmetic language      Version 6 AT&T UNIX
```

Nota: con *q* salimos del paginador less

```
# less -L archivo nos permitiría navegar horizontalmente archivos con líneas largas, como tablas grandes
less -L linux_commands.tab
```

```
# usa less --help para ver más opciones
```

2.8 Edición de archivos con los editores vim o [g|n]edit

vim (vi improved) es un poderoso editor programable presente en todos los sistemas UNIX. La principal característica tanto de Vim como de Vi consiste en que disponen de diferentes modos entre los que se alterna para realizar ciertas operaciones, lo que los diferencia de la mayoría de editores comunes, que tienen un solo modo en el que se introducen las órdenes mediante combinaciones de teclas (o interfaces gráficas). Se controla por completo mediante el teclado desde un Terminal, por lo que puede usarse sin problemas a través de conexiones remotas ya que no carga el sistema al no desplegar un entorno gráfico.

Es muy recomendable aprender a usar VIM, pero no tenemos tiempo de hacerlo en el TIB, por lo que les recomiendo este tutorial de uso de VIM en español, o directamente en su terminal tecleando el comando

```
vimtutor
```

```
# para salir de vim,
```

```
<ESC> # para estar seguros que estamos en modo ex
:q
```

En el taller usaremos generalmente el editor con ambiente gráfico gedit, de uso muy sencillo y similar al block de notas de Windows o similar

```
# noten el uso de & al final de la sentencia para enviar el proceso al fondo
# para evitar que bloquee la terminal
gedit linux_commands.tab &
```

2.9 Edición de archivos con el editor de flujo sed (stream editor)

sed (stream editor) es un editor de flujo, una potente herramienta de tratamiento de texto para el sistema operativo Unix que acepta como entrada un archivo, lo lee y modifica línea a línea de acuerdo a un script, mostrando el resultado por salida estándar (normalmente en pantalla, a menos que se realice una redirección). Sed permite manipular flujos de datos, como por ejemplo cortar líneas, buscar y reemplazar texto (con soporte de expresiones regulares), entre otras cosas. Posee muchas características de ed y ex.

La sintaxis general de la orden sed es:

```
$ sed [-n] [-e'script'] [-f archivo] archivo1 archivo2 ...
```

donde:

-n indica que se suprima la salida estándar.

-e indica que se ejecute el script que viene a continuación. Si no se emplea la opción -f se puede omitir.

-f indica que las órdenes se tomarán de un archivo

2.9.1 Ejemplos de uso básico de sed:

- Cambia todas las minúsculas a mayúsculas de archivo:

```
$ sed 'y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/' archivo
```

- Borra la 1ª línea de archivo:

```
$ sed '1d' archivo
```

- Elimina las líneas en blanco. Nótese el uso de expresiones regulares, donde:
 - // delimitan la expresión regular. Noten que hay que escaparla entre commillas sencillas.
 - ^ indica el inicio de la línea
 - \$ indica el término de la línea

```
$ sed '/^$/d' archivo
```

- Genera una lista numerada de los nombres de campos o cabeceras del archivo linux_commands.tab
 - // delimitan la expresión regular. Noten que hay que escaparla entre commillas sencillas.
 - \t representa al tabulador
 - \n representa el salto de línea
 - //g la g indica que se reemplacen todas las instancias

```
head -1 linux_commands.tab | sed 's/\t/\n/g' | cat -n
```

```
##      1  IEEE Std 1003.1-2008 utilities Name
##      2  Category
##      3  Description
##      4  First appeared
```

2.10 Uso de tuberías de herramientas UNIX/Linux para filtrado de texto con cut, grep, sort, uniq, wc y |

UNIX y Linux ofrecen una gran cantidad de herramientas para todo tipo de trabajos, cada una generalmente con muchas opciones. En bioinformática y genómica, los archivos de texto plano (ASCII) son los más comunes. Por ello es muy útil dominar algunas de las herramientas de filtrado de texto más comunes. Como ejemplo, trabajaremos con el archivo assembly_summary.txt, que contiene los datos de ensamblajes genómicos de la división RefSeq de GenBank. Lo descargué y comprimí con los siguientes comandos:

```
wget -c ftp://ftp.ncbi.nlm.nih.gov/genomes/refseq/bacteria/assembly_summary.txt
gzip assembly_summary.txt
```

- Exploremos el archivo comprimido (con compresión gnu zip) con usando los comandos zless o zcat

```
zless assembly_summary.txt.gz
```

```
# veamos las 5 primeras líneas del archivo
```

```
zcat assembly_summary.txt.gz | head -5
```

```
## # See ftp://ftp.ncbi.nlm.nih.gov/genomes/README_assembly_summary.txt for a description of the columns
## # assembly_accession bioproject biosample wgs_master refseq_category taxid species_taxid organism
## GCF_000010525.1 PRJNA224116 SAMD00060925 representative genome 438753 7 Azorhizobium ca
## GCF_000007365.1 PRJNA224116 SAMN02604269 representative genome 198804 9 Buchnera aphidi
## GCF_000007725.1 PRJNA224116 SAMN02604289 representative genome 224915 9 Buchnera aphidi
```

2.10.1 Ejemplos de herramientas de filtrado de texto en acción

- cut corta líneas de texto/tablas por delimitadores de campo (-d) específicos (TAB por defecto), extrayendo los campos indicados con -f (cut -d' ' -f1-3,5,9)
- sort ordena (sort -u; sort -nrk2; sort -dk1)
- wc cuenta líneas, palabras y caracteres (wc -l)
- uniq regresa listas de valores únicos (uniq -c)
- grep Filtra las líneas de un archivo que contienen (o no) caracteres o expresiones regulares (grep -E '^XXX|YYY|zzz\$'; grep -v '^#')
- el pipe '|' conecta la salida de un comando con la entrada >STDIN> de otro
- ¿cuántas líneas tiene el archivo assembly_summary.txt.gz?

```
# ¿cuántas líneas tiene el archivo assembly_summary.txt.gz?
```

```
zcat assembly_summary.txt.gz | wc
```

```
zcat assembly_summary.txt.gz | wc -l
```

```
## 161297 3788695 48497020
```

```
## 161297
```

- la columna assembly_level (#12) indica el estado del ensamble. ¿Cuáles son los niveles de la variable categórica assembly_level (valores únicos de la misma)?

```
# la columna assembly_level (#12) indica el estado del ensamble. ¿Cuáles son los niveles de la variable
```

```
zcat assembly_summary.txt.gz | grep -v "^#" | cut -f 12 | sort -u
```

```
## Chromosome
```

```
## Complete Genome
```

```
## Contig
```

```
## Scaffold
```

- ¿cuántos genomas hay por nivel de la variable categórica assembly_level?

```
# ¿cuántos genomas hay por nivel de la variable categórica assembly_level?
```

```
zcat assembly_summary.txt.gz | grep -v "^#" | cut -f 12 | sort | uniq -c
```

```
## 2018 Chromosome
```

```
## 13983 Complete Genome
```

```
## 82755 Contig
```

```
## 62539 Scaffold
```

- asocia cada nombre de columna de la cabecera con el número de la columna correspondiente

```
# asocia cada nombre de columna de la cabecera con el número de la columna correspondiente
zcat assembly_summary.txt.gz | head -2 | sed '1d; s/\t/\n/g' | cat -n
```

```
##      1      # assembly_accession
##      2      bioproject
##      3      biosample
##      4      wgs_master
##      5      refseq_category
##      6      taxid
##      7      species_taxid
##      8      organism_name
##      9      infraspecific_name
##     10      isolate
##     11      version_status
##     12      assembly_level
##     13      release_type
##     14      genome_rep
##     15      seq_rel_date
##     16      asm_name
##     17      submitter
##     18      gbrs_paired_asm
##     19      paired_asm_comp
##     20      ftp_path
##     21      excluded_from_refseq
##     22      relation_to_type_material
```

- genera una estadística del número de genomas por especie (columna # 8), y muestra sólo las 10 especies con más genomas secuenciados!

```
# genera una estadística del número de genomas por especie (columna # 8), y muestra sólo las 10 especies
zcat assembly_summary.txt.gz | grep -v "^#" | cut -f8 | sort | uniq -c | sort -nrk1 | head -10
```

```
##    14089 Escherichia coli
##     8039 Streptococcus pneumoniae
##     6398 Klebsiella pneumoniae
##     5924 Staphylococcus aureus
##     4556 Mycobacterium tuberculosis
##     4358 Pseudomonas aeruginosa
##     3164 Acinetobacter baumannii
##     2789 Listeria monocytogenes
##     2173 Salmonella enterica subsp. enterica serovar Typhi
##     1792 Clostridioides difficile
```

- ¿Cuántos genomas completos hay del género Acinetobacter?

```
# ¿Cuántos genomas completos hay del género Acinetobacter?
zcat assembly_summary.txt.gz | grep Acinetobacter | grep Complete | wc -l
```

```
# también puedes usar zgrep para evitar la llamada primero a zcat
zgrep Acinetobacter assembly_summary.txt.gz | grep Complete | wc -l
```

```
## 220
## 220
```

```
# ojo: Linux es sensible a mayúsculas y minúsculas: prueba este comando para comprobarlo
zgrep acinetobacter assembly_summary.txt.gz | grep Complete | wc -l # no encuentra nada
```

```
# grep -i lo hace insensible a la fuente
zgrep -i acinetobacter assembly_summary.txt.gz | grep Complete | wc -l
```

```
## 220
```

- filtra y cuenta las líneas que contienen Acinetobacter o Stenotrophomonas

```
# filtra y cuenta las líneas que contienen Acinetobacter o Stenotrophomonas
zgrep -E 'Acinetobacter|Stenotrophomonas' assembly_summary.txt.gz | wc -l
```

```
## 5170
```

- Cuenta los genomas de Acinetobacter, Pseudomonas y Klebsiella (por género) y presenta una lista ordenada por número decreciente de genomas

```
# Cuenta los genomas de Acinetobacter, Pseudomonas y Klebsiella (por género) y presenta una lista ordenada por número decreciente de genomas
zgrep -E 'Acinetobacter|Pseudomonas|Klebsiella' assembly_summary.txt.gz | cut -f 8 | cut -d' ' -f1 | sort -nr
```

```
##      8951 Pseudomonas
##      8515 Klebsiella
##      4747 Acinetobacter
##         7 [Pseudomonas]
##         1 Candidatus
```

- Cuenta los genomas de Acinetobacter, Pseudomonas y Klebsiella (por género), con salida ordenada alfabéticamente por género

```
# filtra las líneas que contienen Filesystem o Text processing y ordénalas alfabéticamente según las entradas
# eliminando las entradas de Candidatus y [Pseudomonas]
```

```
zgrep -E 'Acinetobacter|Pseudomonas|Klebsiella' assembly_summary.txt.gz | cut -f 8 | cut -d' ' -f1 | sort
```

```
##      4747 Acinetobacter
##      8515 Klebsiella
##      8951 Pseudomonas
```

2.11 redireccionado de la salida STOUT a un archivo con el comando >

```
zgrep Stenotrophomonas assembly_summary.txt.gz | cut -f8,20 > Stenotrophomonas_complete_genomes_and ftp
```

- ahora podemos explorar el archivo con head u otros comandos como less

```
head -3 Stenotrophomonas_complete_genomes_and ftp_paths.txt
```

```
## Stenotrophomonas maltophilia R551-3 ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/020/665/GCF_000000000
## Stenotrophomonas maltophilia K279a ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/072/485/GCF_000000000
## Stenotrophomonas maltophilia JV3 ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/223/885/GCF_000223885
```

Veremos la gran utilidad y versatilidad de combinaciones de estos comandos para el procesamiento de archivos de secuencias en un ejercicio más adelante.

2.12 Manual de cada comando: man command

```
# mira las opciones de cut y sort en la manpage
man cut | head -30
```

```
## CUT(1)
##
```

User Commands

```

## NAME
##      cut - remove sections from each line of files
##
## SYNOPSIS
##      cut OPTION... [FILE]...
##
## DESCRIPTION
##      Print selected parts of lines from each FILE to standard output.
##
##      With no FILE, or when FILE is -, read standard input.
##
##      Mandatory arguments to long options are mandatory for short options too.
##
##      -b, --bytes=LIST
##              select only these bytes
##
##      -c, --characters=LIST
##              select only these characters
##
##      -d, --delimiter=DELIM
##              use DELIM instead of TAB for field delimiter
##
##      -f, --fields=LIST
##              select only these fields; also print any line that contains no delimiter character, u
##
##      -n      (ignored)
##
##      --complement

```

2.13 Ayuda de cada comando: command -help

```

# mira las opciones de cut y sort en la manpage
cut --help

```

```

## Modo de empleo: cut OPCIÓN... [FICHERO]...
## Extrae las partes seleccionadas de cada FICHERO en la salida estándar:
##
## Sin FICHERO, o cuando FICHERO es -, lee la entrada estándar.
##
## Los argumentos obligatorios para las opciones largas son también obligatorios
## para las opciones cortas.
##      -b, --bytes=LISTA      muestra solamente estos bytes
##      -c, --characters=LISTA selecciona solamente estos caracteres
##      -d, --delimiter=DELIM  usa DELIM en vez de caracteres de tabulación para delimitar los campos
##      -f, --fields=LISTA     selecciona solamente estos campos; también muestra
##                              cualquier línea que no tenga un carácter
##                              delimitador, a menos que se especifique la
##                              opción -s
##      -n                      (no tiene efecto)
##      --complement            complementa el conjunto de bytes, caracteres o campos
##                              seleccionados
##      -s, --only-delimited    no muestra las líneas que no contienen
##                              delimitadores

```



```
##      --output-delimiter=CADENA  utiliza CADENA como el delimitador del
##                                resultado. Por omisión se utiliza el
##                                delimitador de la entrada
##  -z, --zero-terminated  line delimiter is NUL, not newline
##      --help      muestra esta ayuda y finaliza
##      --version  informa de la versión y finaliza
##
## Utilice uno, y solamente uno de -b, -c o -f. Cada LISTA está compuesta por un
## rango, o muchos rangos separados por comas. La entrada seleccionada se escribe
## en el mismo orden en el que se lee, y se escribe exactamente una vez.
## Each range is one of:
##
##  N      N'th byte, character or field, counted from 1
##  N-     from N'th byte, character or field, to end of line
##  N-M    from N'th to M'th (included) byte, character or field
##  -M     from first to M'th (included) byte, character or field
##
## ayuda en línea sobre GNU coreutils: <http://www.gnu.org/software/coreutils/>
## Informe de errores de traducción en cut a <http://translationproject.org/team/es.html>
## Full documentation at: <http://www.gnu.org/software/coreutils/cut>
## or available locally via: info '(coreutils) cut invocation'
```

3 Inicios de programación en *Bash*

Vamos a aprender algunos de los elementos esenciales del lenguaje *Bash* de programación Shell

3.1 Asignación y uso de variables

- La sintaxis básica de asignación es:

```
varName=VALUE
```

- para recuperar el valor de una variable, le añadimos el prefijo \$. Para imprimir el valor asignado a la variable, usamos echo \$varName

```
archivo_de_comandos_linux=linux_commands.tab
echo "$archivo_de_comandos_linux"
```

```
## linux_commands.tab
```

3.1.1 Captura de la salida de un comando con \$(comando)

```
wkdir=$(pwd)
date=$(date | awk '{print $3,$2,$6}' | sed 's/ //g')
h=$(hostname)
echo ">>> working in: $wkdir at <$h> on <$date>"
```

```
## >>> working in: /home/vinuesa/Cursos/Taller_CG-FC/sesion1_intro2linux at <alisio> on <11ago2019>
```

3.1.2 Modificación de variables y operaciones con ellas

```
wkdir=$(pwd)
echo "wkdir: $wkdir"

# 1. cortemos caracteres por la izquierda (todos los caracteres por la izquierda, hasta llegar a último
basedir=${wkdir##*/}
echo "basedir: $basedir # \${wkdir##*/}"

# 2. cortemos caracteres por la derecha (cualquier caracter hasta llegar a /)
echo "path to basedir: ${wkdir%/*} # \${wkdir%/*}"

# 3. contar el número de caracteres (longitud) de la variable
echo "basedir has ${#basedir} characters # \${#basedir}"

## wkdir: /home/vinuesa/Cursos/Taller_CG-FC/sesion1_intro2linux
## basedir: sesion1_intro2linux # ${wkdir##*/}
## path to basedir: /home/vinuesa/Cursos/Taller_CG-FC # ${wkdir%/*}
## basedir has 19 characters # ${#basedir}
```

3.2 Condicionales

- La sintaxis básica de un condicional simple en formato de una línea es así

```
if [ condición ]; then orden1; orden2 ...; fi
```

- también hay una versión más corta para test siples

```
[ condición ] && setecia1 && sentencia2
```

- En un script, lo escribimos generalmente como un bloque indentado, para mejor legibilidad

```
if [ condición ]; then
    orden1
    orden2
fi
```

3.2.1 Comparación de íntegros en condicionales

```
i=5
j=3

if [ "$i" -lt "$j" ]; then
    echo "$i < $j"
elif [ "$i" -gt "$j" ]; then
    echo "$i > $j "
fi
```

```
## 5 > 3
```

3.2.2 Comparación de cadenas de caracteres en condicionales

```

c=carla
j=juan

if [ "$c" == "$j" ]; then
    echo "$c = $j"
elif [ "$c" != "$j" ]; then
    echo "c:$c != j:$j "
fi

## c:carla != j:juan

```

3.2.3 Comprobación de la existencia de un archivo de tamaño > 0 bytes

```

touch empty_file
ls -l empty_file
ls -l *gz
f=$(ls *gz)

if [ -e empty_file ]; then
    echo "empty_file file exists"
fi

if [ ! -s empty_file ]; then
    echo "empty_file file exists but is empty"
fi

if [ -s "$f" ]; then
    size=$(du -h assembly_summary.txt.gz | cut -f1)
    # o tambien
    # size=$(ls -hs assembly_summary.txt.gz | cut -d' ' -f1)
    echo "$f exists and has size: $size"
fi

## -rw-r--r-- 1 vinuesa vinuesa 0 ago 11 20:50 empty_file
## -rw-r--r-- 1 vinuesa vinuesa 6780296 ago  8 18:05 assembly_summary.txt.gz
## empty_file file exists
## empty_file file exists but is empty
## assembly_summary.txt.gz exists and has size: 6.5M

```

3.2.4 La versión corta de test [condición] && comando1 && comando2 ...

```

# también podemos usar la versión corta del test:
f=$(ls *gz)
[ -s "$f" ] && echo "$f exists and is non-empty"

## assembly_summary.txt.gz exists and is non-empty

```

3.2.5 if; elif; else

```

if [[ "$OSTYPE" == "linux-gnu" ]]
then

```

```

OS='linux'
no_cores=$(awk '/^processor/{n+=1}END{print n}' /proc/cpuinfo)
host=$(hostname)
echo "running on $host under $OS with $no_cores cores :)"
elif [[ "$OSTYPE" == "darwin"* ]]
then
    OS='darwin'
    no_cores=$(sysctl -n hw.ncpu)
    host=$(hostname)
    echo "running on $host under $OS with $no_cores cores :)"
else
    OS='windows'
    echo "oh no! another windows box :( ... you should better change to linux :) "
fi

```

running on alisio under linux with 12 cores :)

3.3 Bucles for

la sintaxis general de un bucle for en *Bash* es:

for ALIAS in LIST; do CMD1; CMD2; done

donde el usuario tiene que cambiar los términos en mayúsculas por opciones concretas. ALIAS es el nombre de una variable temporal a la que se asigna secuencialmente cada valor de LIST.

Así por ejemplo, si tuviéramos muchos archivos de secuencias homólogas con la extensión *.faa en un directorio, podríamos alinearlas secuencialmente con *clustalo* con un comando como el siguiente:

for file in .faa; do clustalo -i \$file -o \${file%.*}_cluAln.faa; done

donde ALIAS=file, LIST=*.faa y CMD1 es una llamada al programa de alineamientos múltiples clustalo que veremos más adelante en este taller.

3.3.1 Ejemplo de bucle for, acoplado a las herramientas de filtrado y de manipulación de variables

La idea del ejercicio es generar archivos a partir de linux_basic_commands.tab que contengan sólo los comandos de cada clase, nombrando a los archivo resultantes con el valor de dicha clase, almacenados en la segunda columna de la tabla

```

# veamos la cabecera y cola del archivo linux_basic_commands.tab
head linux_basic_commands.tab
echo '-----'
tail linux_basic_commands.tab

```

```

## IEEE Std 1003.1-2008 utilities Name Category Description First appeared
## admin SCCS Create and administer SCCS files PWB UNIX
## alias Misc Define or display aliases
## ar Misc Create and maintain library archives Version 1 AT&T UNIX
## asa Text processing Interpret carriage-control characters System V
## at Process management Execute commands at a later time Version 7 AT&T UNIX
## awk Text processing Pattern scanning and processing language Version 7 AT&T UNIX
## basename Filesystem Return non-directory portion of a pathname; see also dirname Version 7 A
## batch Process management Schedule commands to be executed in a batch queue
## bc Misc Arbitrary-precision arithmetic language Version 6 AT&T UNIX

```

```
## -----
## val  SCCS      Validate SCCS files      System III
## vi   Text processing    Screen-oriented (visual) display editor      1BSD
## wait Process management Await process completion    Version 4 AT&T UNIX
## wc   Text processing    Line, word and byte or character count    Version 1 AT&T UNIX
## what SCCS      Identify SCCS files      PWB UNIX
## who  System administration    Display who is on the system      Version 1 AT&T UNIX
## write Misc      Write to another user's terminal      Version 1 AT&T UNIX
## xargs Shell programming    Construct argument lists and invoke utility      PWB UNIX
## yacc C programming    Yet another compiler compiler    PWB UNIX
## zcat  Text processing    Expand and concatenate data      4.3BSD
```

Antes de correr el bucle, lista los archivos en el directorio de trabajo

```
# veamos el contenido del directorio antes de correr el bucle
ls
```

Ahora el bucle. En este caso ALIAS=type y LIST corresponde a la lista de valores únicos almacenados en la segunda columna de la tabla: `$(cut -f2 linux_basic_commands.tab | sort -u)`

```
#>>> Ejemplo integrativo: usa un bucle for, acoplado a las herramientas de filtrado arriba mostradas,
# para generar archivos que contengan solo los comandos de las diferentes categorias
# nombrando a los archivos por estas

# for type in $(cut -f2 linux_basic_commands.tab | sort -u); do grep "$type" linux_basic_commands.tab >
for type in $(cut -f2 linux_basic_commands.tab | sort -u); do
    grep "$type" linux_basic_commands.tab > ${type}.cmds
done
```

Y voilà:

```
# veamos el contenido del directorio después de correr el bucle
ls *.cmds
```

```
## administration.cmds
## Batch.cmds
## Category.cmds
## C.cmds
## Filesystem.cmds
## FORTRAN77.cmds
## management.cmds
## Misc.cmds
## Network.cmds
## Process.cmds
## processing.cmds
## programming.cmds
## Programming.cmds
## SCCS.cmds
## Shell.cmds
## System.cmds
## Text.cmds
## utilities.cmds
```

```
# veamos el contenido de uno de los nuevos archivos generados
cat programming.cmds
```

```
## cc/c99    C programming    Compile standard C programs      IEEE Std 1003.1-2001
## cflow     C programming    Generate a C-language call graph    System V
```

```

## command Shell programming Execute a simple command
## ctags C programming Create a tags file 3BSD
## cxref C programming Generate a C-language program cross-reference table System V
## echo Shell programming Write arguments to standard output Version 2 AT&T UNIX
## expr Shell programming Evaluate arguments as an expression Version 7 AT&T UNIX
## false Shell programming Return false value Version 7 AT&T UNIX
## fort77 FORTRAN77 programming FORTRAN compiler XPG4
## getopt Shell programming Parse utility options
## lex C programming Generate programs for lexical tasks Version 7 AT&T UNIX
## logger Shell programming Log messages 4.3BSD
## nm C programming Write the name list of an object file Version 1 AT&T UNIX
## printf Shell programming Write formatted output 4.3BSD-Reno
## read Shell programming Read a line from standard input
## sh Shell programming Shell, the standard command language interpreter Version 7 AT&T UNIX (in
## sleep Shell programming Suspend execution for an interval Version 4 AT&T UNIX
## strings C programming Find printable strings in files 2BSD
## strip C programming Remove unnecessary information from executable files Version 1 AT&T UNIX
## tee Shell programming Duplicate the standard output Version 5 AT&T UNIX
## test Shell programming Evaluate expression Version 7 AT&T UNIX
## true Shell programming Return true value Version 7 AT&T UNIX
## xargs Shell programming Construct argument lists and invoke utility PWB UNIX
## yacc C programming Yet another compiler compiler PWB UNIX
# finalmente borremos los nuevos archivos generados
rm *.cmds

```

3.4 Bash scripts - primeros pasos

Para finalizar esta sección, voy a mostrarles varios ejemplos muy sencillos de **Shell scripts** escritos en *Bash* que integran varios aspectos de la sintaxis básica del lenguaje arriba descritas, así como extensiones adicionales como son **argumentos posicionales** y **funciones**

3.4.1 la “shebang line” (`#!/usr/bin/env bash`) y permisos de ejecución - el script `ls_dir`

Se conocen como *scripts* a archivos de texto plano (codificación ASCII) que contienen los comandos a ser ejecutados secuencialmente por un **intérprete de comandos** particular, como *bash*, *perl*, *R* ...

Les muestro abajo el código del script `ls_dir`.

```

#!/usr/bin/env bash

# the 1st line in a script is the so-called shebang line
# which indicates the system which command interpreter
# should read and execute the following code, bash in this case
# The shebang line shown above, is a portable version for bash scripts

# AUTHOR: Pablo Vinuesa
# AIM: learning basic BASH-programming constructs for TIB2019-T3
# https://github.com/vinuesa/TIB-filoinfo

for file in $(ls)
do
    if [ -d "$file" ]
    then

```

```

        echo "$file"
    fi
done

```

Puedes copiar el código a un archivo que vas a nombrar como `ls_dir`. Usa para ello el comando `cat`, de la manera abajo indicada:

```

cat > ls_dir
PEGA AQUÍ EL CÓDIGO ARRIBA MOSTRADO
CTRL-D

```

El script `ls_dir` busca los directorios presentes en el directorio actual usando un bucle `for` para analizar cada archivo econtrado por `ls`, evaluando seguidamente si el archivo en turno es un directorio con un condicional `if [-d "$file"]`

Noten que la primera línea inicia con lo que se conoce como una **shebang line**:

```
#!/usr/bin/env bash
```

Esta línea, en la cabecera del archivo (¡sin dejar espacios a la izquierda o arriba de la línea!), le indica al sistema operativo qué **intérprete de comandos** usar para la ejecución del código que sigue. En este caso se llama al intérprete de comandos `bash`

Noten también que cualquier línea que inicie con `#` después de la shebang line, representa un comentario, es decir, que el texto que sigue al gato no es interpretado por `bash`

Para ejecutar el script como si fuera un comando cualquiera de *Linux*, le damos **permisos de ejecución**:

```
chmod 755 ls_dir
```

Comprueba los permisos:

```

$ ls -l ls_dir

-rwxr-xr-x 1 vinuesa vinuesa 493 ago 11 18:37 ls_dir

```

Como puedes ver el usuario, el grupo al que pertenece y todos los demás pueden ejecutarlo (*x*)

Finalmente copia o mueve el script a un directorio que esté en el *PATH*, típicamente a *HOME/bin*

```
mv ls_dir ~/bin
```

y ya puedes usarlo:

```
ls_dir
```

```

## pics
## recA_seq_data
## TIB2019-T3

```

Nota: el script `ls_dir` funciona perfectamente, pero no es la manera más eficiente de buscar directorios. La función `find` de *Linux* es mucho más eficiente.

3.4.2 Funciones y argumentos posicionales en *Bash* - el script `find_dir`

Los scripts pueden recibir opciones dadas por los usuarios para comportarse acorde a ellas. Esto les da gran versatilidad.

La manera más sencilla de pasarle opciones al script es mediante **argumentos posicionales**, como los usados por `find_dir`

```
#!/usr/bin/env bash
```

```
# The 1st line in a script is the so-called shebang line
```

```

# which indicates the system which command interpreter
# should read and execute the following code, bash in this case
# The shebang line shown above, is a portable version for bash scripts

# AUTHOR: Pablo Vinuesa
# AIM: learning basic BASH-programming constructs for TIB2019-T3
# https://github.com/vinuesa/TIB-filoinfo

# global variables
progrname=${0##*/}      # find_dir
VERSION='0.1_29Jul19'

# Function definition
function print_help()
{
    # this is a so-called HERE-DOC, to easily print out formatted text
    cat << HELP

    $progrname v$VERSION usage synopsis:

    $progrname <int [maximal search depth for (sub)directories, e.g.:1>

    AIM: find directories below the current one with the desired max_depth

    USAGE EXAMPLES:
        $progrname 1
        $progrname 2

    HELP

    exit 1
}

# capture user-provided arguments in variables $1, $2 ... $9
# and save them to named variables, for better readability
max_depth=$1
type=${2:-d} # if not provided, the second argument is set to 'd' by default.

# check arguments
[ -z $max_depth ] && print_help

# execute find command with the provided arguments
find . -maxdepth $max_depth -type $type

Guarda el código arriba mostrado en un archivo llamado find_dir, dale permisos de ejecución, y cópialo a
HOME/bin, como se mostró en el ejemplo anterior.

find_dir hace la siguiente llamada a find
find . -maxdepth $max_depth -type $type

para encontrar directorios y subdirectorios por debajo del actual, al nivel de profundidad indicado por el
usuario, quien pasa un íntegro al script como único argumento

find_dir introduce la función print_help(),
function print_help(){

```



```
YOUR CODE GOES INHERE ...
}
```

que simplemente imprime un mensaje de ayuda si es llamado sin argumentos

```
[ -z $max_depth ] && print_help
```

Probemos el script:

- primero llamando al script sin argumento:

```
find_dir
```

```
find_dir v0.1_29Jul19 usage synopsis:
```

```
find_dir <int [maximal search depth for (sub)directories, e.g.:1>
```

```
AIM: find directories below the current one at the desired max_depth
```

```
USAGE EXAMPLES
```

```
find_dir 1
```

```
find_dir 2
```

- ahora pasándole un argumento:

```
find_dir 1
```

```
## .
## ./pics
## ./TIB2019-T3
## ./recA_seq_data
```

3.4.3 Alineamiento múltiple de secuencias e interconversión de formatos con *align_seqs_with_clustal_or_muscle.sh* y *convert_alnFormats_using_clustalw.sh*

Para finalizar esta sección de introducción a la programación en *Bash*, explora el código del script *align_seqs_with_clustal_or_muscle.sh*, que permite hacer alineamientos múltiples con *clustalw* o con *muscle*, dos programas muy populares para este fin.

Explora también el código del script *convert_alnFormats_using_clustalw.sh*, que permite hacer interconversión de formatos con *clustalw*.

Los alineamientos múltiples y las llamadas a *clustalw* las explicaremos en la sesión4_alineamientos.

Con lo aprendido hasta ahora y los comentarios que documentan los scripts, deberías entender lo que hacen.

Vuelve a explorar ambos scripts después de haber revisado la sesión4_alineamientos.

4 El lenguaje de procesamiento de patrones AWK

AWK es un lenguaje de programación diseñado para procesar datos de texto, ya sean archivos o flujos de datos. El nombre AWK deriva de las iniciales de los apellidos de sus autores: Alfred Aho, Peter Weinberger, y Brian Kernighan. *awk*, cuando está escrito todo en minúsculas, hace referencia al programa de Unix que interpreta programas escritos en el lenguaje de programación AWK. Es decir AWK es un lenguaje interpretado por el intérprete de comando *awk*.

AWK fue creado como un reemplazo a los algoritmos escritos en C para análisis de texto. Fue una de las primeras herramientas en aparecer en Unix (en la versión 3). Ganó popularidad rápidamente por la gran funcionalidad permitía añadir a las tuberías de comandos de Unix. Por ello se considera como una de las utilidades necesarias de este sistema operativo y de Linux.

Debido a su densa notación, todos estos lenguajes son frecuentemente usados para escribir programas de una línea, como veremos seguidamente.

4.1 Estructura de los programas AWK

En general, a *awk* se le dan dos piezas de datos: un fichero de órdenes y un archivo de entrada.

Un fichero de órdenes (que puede ser un fichero real, o puede ser incluido en la invocación de *awk* desde la línea de comandos) contiene una serie de sentencias que le indican a *awk* cómo procesar el fichero de entrada.

El fichero primario de entrada es normalmente texto estructurado con un formato particular, como archivos con campos separados por tabuladores (tablas).

4.1.1 Estructura básica - ejemplos genéricos

- Un programa AWK típico consiste en una serie de líneas, cada una de la forma:

/patrón/ { acción }, donde:

- la acción por defecto es imprimir {print}
- patrón es una expresión regular
- acción es una orden.

La mayoría de las implementaciones de AWK usan expresiones regulares extendidas (EREs) por defecto. AWK lee línea por línea el fichero de entrada. Cuando encuentra una línea que coincide con el “patrón”, ejecuta la(s) orden(es) indicadas en “acción”.

- Para llamar a *awk* desde la línea de comandos, usaríamos una sintaxis de este tipo:

```
awk 'CODIGO AWK' ARCHIVO_A_PROCESAR
```

- para usarlo en una tubería de UNIX, conecta el STDOUT de un programa al STDIN de *awk* mediante |:

```
STDOUT_programaX | awk 'CODIGO AWK' > output_file.txt
```

4.1.2 Formas alternativas del código AWK:

BEGIN { acción } Ejecuta las órdenes de acción al comienzo de la ejecución, antes de que los datos comiencen a ser procesados.

END { acción } Similar a la forma previa, pero ejecuta las órdenes de acción después de que todos los datos sean procesados.

/patrón/ Imprime las líneas que contienen al patrón.

{ acción } Ejecuta acción por cada línea en la entrada.

Cada una de estas formas pueden ser incluidas varias veces en un archivo o script de AWK. El script es procesado de manera progresiva, línea por línea, de izquierda a dercha. Entonces, si hubiera dos declaraciones “BEGIN”, sus contenidos serán ejecutados en orden de aparición. Las declaraciones “BEGIN” y “END” no necesitan estar en forma ordenada.

4.1.3 Sintaxis condensada de AWK

- AWK, al ser un lenguaje de programación completo, contiene sintaxis para escribir:
 - condicionales y bucles for\$ y \$while
 - operadores aritméticos +, -, *, /, %, =, ++, -, +=, -=, ...)
 - operadores booleanos ||, &&
 - operadores relacionales <, <=, == !=, >=, >
 - funciones integradas: length(str); int(num); index(str1, str2); split(str,arr,del); sprintf(fmt,args); substr(str,pos,len); tolower(str); toupper(str)
 - funciones escritas por el usuario function FUNNAME (arg1, arg1){code}
 - estructuras de datos como arreglos asociativos (hashes o diccionarios): array[string]=value, entre otros.
- AWK Maneja también una serie de variables propias, de las que les resalto sólo las más usadas:

```
$0      guarda el valor de la fila actual en memoria de un archivo de entrada
$1-$n   guarda los contenidos de los campos de una fila
FILENAME nombre del archivo de entrada actualmente en procesamiento
FS      separador de campos (por defecto SPACE or TAB)
NR      guarda el número de campos delimitados por FS en registro o fila actual
OFS     separador de campo de la salida (SPACE por defecto)
ORS     separador de registro de la salida (\n por defecto)
```

4.2 Ejemplos básicos pero muy útiles de uso de AWK

No podemos aprender aquí más que algunos idiomas de AWK muy útiles, como veremos seguidamente

4.2.1 filtrado de archivos con AWK

- Cuenta el número de procesadores de tu sistema:

```
# el archivo /proc/cpuinfo contiene la información sobre las cpus del sistema, incluyendo los cores/pro
head -5 /proc/cpuinfo
```

```
## processor      : 0
## vendor_id      : GenuineIntel
## cpu family     : 6
## model          : 158
## model name     : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
```

Recordemos la sintaxis general de código AWK: /patrón/ { acción }

```
# usamos el patrón '/^processor/', seguido de la acción {cuanta instancias} y terminamos con un bloque E
# este código de AWK se lo pasamos directamente al intérprete de comandos awk como un una cadena entre
# awk 'CODIGO AWK' ARCHIVO_A_PROCESAR
awk '/^processor/{n++} END{ print "This computer has", n, "processors"}' /proc/cpuinfo
```

```
## This computer has 12 processors
```

- Imprime líneas con 12 o menos caracteres de entre las primeras 20 líneas del archivo /proc/cpuinfo :

```
# con head -20 filtramos las primeras 20 líneas, las cuales pasamos a awk con |
# recuerda: la acción por defecto de awk es imprimir, en este caso las líneas que satisfagan la condici
head -20 /proc/cpuinfo | awk 'length <= 12'
```

```
## model          : 158
## core id        : 0
```

```
## apicid      : 0
## fpu        : yes
## wp         : yes
```

- Imprime líneas con 30 o más caracteres de entre las primeras 20 líneas del archivo /proc/cpuinfo :

```
head -20 /proc/cpuinfo | awk 'length >= 30'
```

```
## model name   : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
## flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts a
```

5 Ejercicios integrativos de uso de herramientas de filtrado del Shell

5.1 Filtrado de archivos separados por tabuladores (tablas) con AWK y su graficado con R

- Asocia cada nombre de columna de la tabla assembly_summary.txt.gz con su número de campo

```
# asocia cada nombre de columna de la cabecera con el número de la columna correspondiente
zcat assembly_summary.txt.gz | head -2 | sed '1d; s/\t/\n/g' | cat -n
```

```
##      1  # assembly_accession
##      2  bioproject
##      3  biosample
##      4  wgs_master
##      5  refseq_category
##      6  taxid
##      7  species_taxid
##      8  organism_name
##      9  infraspecific_name
##     10  isolate
##     11  version_status
##     12  assembly_level
##     13  release_type
##     14  genome_rep
##     15  seq_rel_date
##     16  asm_name
##     17  submitter
##     18  gbrs_paired_asm
##     19  paired_asm_comp
##     20  ftp_path
##     21  excluded_from_refseq
##     22  relation_to_type_material
```

- cuenta aquellas entradas de la tabla que tienen un número de accesión revisado v2

```
# >>> ojo, es importante definir FS="\t", para que tome como campos sólo a aquellos separados por tabuladores
# ejemplo de código AWK con la estructura: BEGIN_BLOCK, condición, acción, END_BLOCK
# recuerden, como assembly_summary.txt.gz está comprimido, necesitamos zcat para poderlo leer y enviarlo a awk
zcat assembly_summary.txt.gz | awk 'BEGIN{FS="\t"} $16 ~ /v2$/ {n++} END{print n}'
```

```
## 4488
```

- cuenta aquellas entradas de la tabla que tienen un número de accesoión revisado v2, publicados en 2019 para genomas en estado Scaffold

```
# ejemplo de código AWK con la estructura: BEGIN_BLOCK, condición1 EX condición2 EX condición3, acción,
zcat assembly_summary.txt.gz | awk 'BEGIN{FS="\t"} $16 ~ /v2$/ && $15 ~ /2019/ && $12 == "Scaffold" {nt+
## 31
```

- veamos las entradas de la tabla que tienen un número de accesoión revisado v2, publicados en 2019 para genomas en estado Scaffold, pero imprime sólo los campos organism_name y ftp_path en formato tabla (OFS="␣"), imprimiendo sólo las primeras 3 líneas

```
# ejemplo de código AWK con la estructura: BEGIN_BLOCK, condición1 EX condición2 EX condición3, acción
zcat assembly_summary.txt.gz | awk 'BEGIN{FS="\t"; OFS="\t"} $16 ~ /v2$/ && $15 ~ /201./ && $12 == "Sca

## Leptospira interrogans ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/002/370/085/GCF_002370085.2_ASM2
## Helicobacter pylori GAM100Ai ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/310/005/GCF_000310005.2_
## Helicobacter pylori GAM101Biv ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/344/945/GCF_00034494
```

- veamos las entradas de la tabla que tienen un número de accesoión revisado v2, publicados en 2019 para genomas en estado Scaffold, pero imprime sólo los campos organism_name y ftp_path separados por '~', imprimiendo sólo las primeras 3 líneas

```
# ejemplo de código AWK con la estructura: BEGIN_BLOCK, condición1 EX condición2 EX condición3, acción
zcat assembly_summary.txt.gz | awk 'BEGIN{FS="\t"; OFS=" ~~~ "} $16 ~ /v2$/ && $15 ~ /201./ && $12 == "

## Leptospira interrogans ~~~ ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/002/370/085/GCF_002370085.2_ASM
## Helicobacter pylori GAM100Ai ~~~ ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/310/005/GCF_00031000
## Helicobacter pylori GAM101Biv ~~~ ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/344/945/GCF_0003449
```

- genera una estadística del número de genomas por especie (columna # 8) del género Pseudomonas en formato tabular [EspecieTABnum_genomas], y muestra sólo las especies con al menos 20 genomas secuenciados. Añade una cabecera a la salida.

```
# genera una estadística del número de genomas por especie (columna # 8) del género Pseudomonas en form
echo -e "Especie\tnum_genomas"
zcat assembly_summary.txt.gz | grep Pseudomonas | cut -f8 | sort | uniq -c | sort -nrk1 | sed 's/Pseudon

## Especie num_genomas
## Pseudomonas_aeruginosa 4358
## Pseudomonas_sp. 1499
## Pseudomonas_fluorescens 113
## Pseudomonas_syringae 87
## Pseudomonas_putida 86
## Pseudomonas_stutzeri 69
## Pseudomonas_syringae 64
## Pseudomonas_syringae 53
## Pseudomonas_coronafaciens 47
## Pseudomonas_protegens 37
## Pseudomonas_savastanoi 36
## Pseudomonas_savastanoi 36
## Pseudomonas_chlororaphis 24
## Pseudomonas_lundensis 21
```

- Repitamos el ejercicio anterior, generando un archivo con campos separados por comas (csv), que se puede leer en R para generar un *dataframe* de R y generar una gráfica fácilmente. Para ello lo guardamos en un archivo

```
# Noten que primero imprimimos una cabecera al archivo Pseudomonas_species_with_gt_20_genomes.csv, y s
echo -e "Especie,num_genomas" > Pseudomonas_species_with_gt_20_genomes.csv
zcat assembly_summary.txt.gz | grep Pseudomonas | cut -f8 | sort | uniq -c | sort -nrk1 | sed 's/Pseudon
```

- Visualiza la cabecera del archivo que acabamos de escribir

```
head -5 Pseudomonas_species_with_gt_20_genomes.csv
```

```
## Especie,num_genomas
## Pseudomonas_aeruginosa,4358
## Pseudomonas_sp.,1499
## Pseudomonas_fluorescens,113
## Pseudomonas_syringae,87
```

- Llamemos a R para hacer una gráfica

```
$ R
```

```
# 1. leemos el archivo a una estructura de datos tipo dataframe
dfr <- read.csv(file = "Pseudomonas_species_with_gt_20_genomes.csv", header = TRUE)
```

```
str(dfr)
```

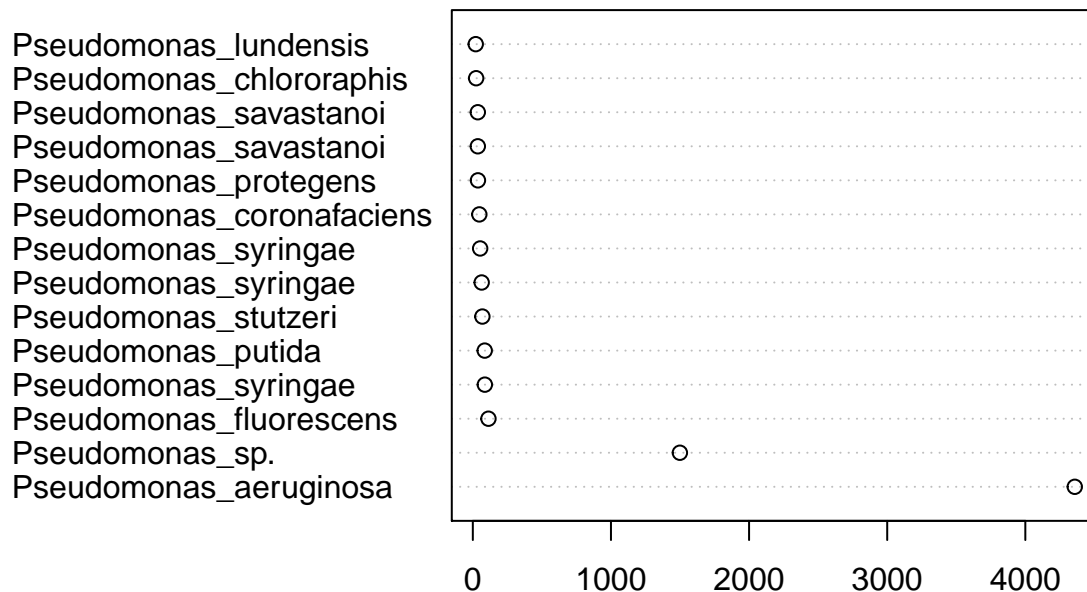
```
## 'data.frame': 14 obs. of 2 variables:
## $ Especie : Factor w/ 11 levels "Pseudomonas_aeruginosa",...: 1 9 4 11 7 10 11 11 3 6 ...
## $ num_genomas: int 4358 1499 113 87 86 69 64 53 47 37 ...
```

```
head(dfr)
```

```
##               Especie num_genomas
## 1 Pseudomonas_aeruginosa      4358
## 2      Pseudomonas_sp.      1499
## 3 Pseudomonas_fluorescens      113
## 4      Pseudomonas_syringae       87
## 5      Pseudomonas_putida       86
## 6      Pseudomonas_stutzeri       69
```

```
dotchart(dfr$num_genomas, labels = dfr$Especie, main = "Número de genomas por especie")
```

Número de genomas por especie



5.1.1 Reto de programación *awk* y *R*

Repita el ejercicio anterior, incluyendo el graficado del número de genomas por especie para el género *Acinetobacter*, pero graficando sólo aquellas especies con mínimo 5 y máximo 100 genomas

¡Felicidades, ya estás aprendiendo a programar! No es tan difícil, ¿verdad?

5.2 Ejercicios de exploración y parseo de archivos FASTA

Te propongo el siguiente ejercicio con un archivo de secuencias de DNA en formato FASTA para practicar algunos aspectos de lo aprendido en esta primera sesión.

Para correr los ejercicios, asegúrate de tener el archivo `recA_Bradyrhizobium_vinuesa.fna` en el directorio actual de trabajo.

El archivo `recA_Bradyrhizobium_vinuesa.fna` contiene secuencias del gen *recA* de bacterias del género *Bradyrhizobium* depositadas en GenBank por P. Vinuesa.

5.2.1 Búsqueda y descarga de secuencias en GenBank usando el sistema ENTREZ

Este bloque muestra el comando que usé para descargarlas usando el sistema ENTREZ de NCBI. El comando debe pegarse en la ventana superior del sistema ENTREZ.

```
# pega esta sentencia en la ventana de captura para interrogar la base de datos
# de nucleótidos de NCBI mediante el sistema ENTREZ
'Bradyrhizobium[orgn] AND vinuesa[auth] AND recA[gene]'
```

No hace falta que las descargues de NCBI. Para facilitar el acceso a las mismas, usa el siguiente código

5.2.2 Acceso a las secuencias

En primer lugar, debes estar en tu directorio \$HOME/TIB2019-T3/sesion1_linux, y desde ahí generar una liga simbólica al archivo FASTA con las secuencias

```
cd $HOME/TIB2019-T3/sesion1_linux
ln -s /space31/PIG/vinuesa/TIB2019-T3/sesion1_Linux/data/recA_Bradyrhizobium_vinuesa.fna .

ls recA_Bradyrhizobium_vinuesa.fna
```

5.2.3 Inspección y estadísticas básicas de las secuencias descargadas

1. ¿Cuántas secuencias hay en el archivo recA_Bradyrhizobium_vinuesa.fna?
2. Explora la cabecera y cola del archivo con head y tail
3. Despliega las 5 primeras líneas de cabeceras fasta usando **grep** y **head** para explorar su estructura en detalle
4. Calcula el número de generos que contiene el archivo FASTA
5. Calcula el número de especies que contiene el archivo FASTA
6. Imprime una lista ordenada de mayor a menor, del numero de especies que contiene el archivo FASTA

5.2.4 Edición de las cabeceras FASTA mediante herramientas de filtrado de UNIX

1. Explora nuevamente todas las cabeceras FASTA del archivo recA_Bradyrhizobium_vinuesa.fna usando **grep** y **less**
2. Simplifica las cabeceras FASTA usando el comando **sed** (stream editor)

El objetivo es eliminar redundancia y los campos gb|no.de.acceso, así como todos los caracteres '(, ; :)' que impedirían el despliegue de un árbol filogenético, al tratarse de caracteres reservados del formato NEWICK. Dejar solo el numero GI, así como el género, especie y cepa indicados entre corchetes.

Es decir vamos a: - reducir Bradyrhizobium a 'B' - eliminar 'recombinase ...' y reemplazarlo por '[' - eliminar 'genosp.' - sustituir espacios por guiones bajos

Nota: hagan uso de expresiones regulares como '.*' y '[:space:]'

3. Cuando estén satisfechos con el resultado, guarden la salida del comando en un archivo llamado recA_Bradyrhizobium_vinuesa.fnaed

5.3 Solución a la práctica y un ejercicio adicional

Este ejercicio está basado en un capítulo que escribí para el manual de Sistemática Molecular y Bioinformática. Guía práctica, editado por la Facultad de Ciencias.

5.3.1 Inspección y estadísticas básicas de las secuencias descargadas

1. ¿Cuántas secuencias hay en el archivo recA_Bradyrhizobium_vinuesa.fna?

```
grep -c '^>' recA_Bradyrhizobium_vinuesa.fna
```

```
## 125
```

2. Veamos las 5 primeras líneas de cabeceras fasta usando **grep** y **head**


```
grep '^>' recA_Bradyrhizobium_vinuesa.fna | head -5
```

```
## >EU574327.1 Bradyrhizobium liaoningense strain ViHaR5 recombination protein A (recA) gene, partial co
## >EU574326.1 Bradyrhizobium liaoningense strain ViHaR4 recombination protein A (recA) gene, partial co
## >EU574325.1 Bradyrhizobium liaoningense strain ViHaR3 recombination protein A (recA) gene, partial co
## >EU574324.1 Bradyrhizobium liaoningense strain ViHaR2 recombination protein A (recA) gene, partial co
## >EU574323.1 Bradyrhizobium liaoningense strain ViHaR1 recombination protein A (recA) gene, partial co
```

3. Cuenta el numero de generos y especies que contiene el archivo FASTA

```
grep '^>' recA_Bradyrhizobium_vinuesa.fna | cut -d' ' -f2,3 | sort | uniq -c
```

```
##      18 Bradyrhizobium canariense
##      18 Bradyrhizobium elkanii
##       6 Bradyrhizobium genosp.
##      28 Bradyrhizobium japonicum
##      15 Bradyrhizobium liaoningense
##       8 Bradyrhizobium sp.
##      32 Bradyrhizobium yuanmingense
```

4. Imprime una lista ordenada de mayor a menor, del numero de especies que contiene el archivo FASTA

```
grep '^>' recA_Bradyrhizobium_vinuesa.fna | cut -d' ' -f2,3 | sort | uniq -c | sort -nrk1
```

```
##      32 Bradyrhizobium yuanmingense
##      28 Bradyrhizobium japonicum
##      18 Bradyrhizobium elkanii
##      18 Bradyrhizobium canariense
##      15 Bradyrhizobium liaoningense
##       8 Bradyrhizobium sp.
##       6 Bradyrhizobium genosp.
```

5.3.2 Edición de las cabeceras FASTA mediante herramientas de filtrado de UNIX

5. Exploremos todas las cabeceras FASTA del archivo recA_Bradyrhizobium_vinuesa.fna usando **grep**

```
# grep '^>' recA_Bradyrhizobium_vinuesa.fna | less # para verlas por página
grep '^>' recA_Bradyrhizobium_vinuesa.fna | head # para no hacer muy extensa la salida
```

```
## >EU574327.1 Bradyrhizobium liaoningense strain ViHaR5 recombination protein A (recA) gene, partial co
## >EU574326.1 Bradyrhizobium liaoningense strain ViHaR4 recombination protein A (recA) gene, partial co
## >EU574325.1 Bradyrhizobium liaoningense strain ViHaR3 recombination protein A (recA) gene, partial co
## >EU574324.1 Bradyrhizobium liaoningense strain ViHaR2 recombination protein A (recA) gene, partial co
## >EU574323.1 Bradyrhizobium liaoningense strain ViHaR1 recombination protein A (recA) gene, partial co
## >EU574322.1 Bradyrhizobium liaoningense strain ViHaG8 recombination protein A (recA) gene, partial co
## >EU574321.1 Bradyrhizobium liaoningense strain ViHaG7 recombination protein A (recA) gene, partial co
## >EU574320.1 Bradyrhizobium liaoningense strain ViHaG6 recombination protein A (recA) gene, partial co
## >EU574319.1 Bradyrhizobium yuanmingense strain ViHaG5 recombination protein A (recA) gene, partial co
## >EU574318.1 Bradyrhizobium yuanmingense strain ViHaG4 recombination protein A (recA) gene, partial co
```

6. simplifiquemos las cabeceras FASTA usando el comando **sed** (stream editor)

El objetivo es eliminar redundancia y los campos gb|no.de.acceso, así como todos los caracteres ‘(, ; :)’ que impedirían el despliegue de un árbol filogenético, al tratarse de caracteres reservados del formato NEWICK. Dejar solo el numero de accesión, así como el género, especie y cepa indicados entre corchetes.

Es decir vamos a: - reducir Bradyrhizobium a ‘B.’ - eliminar ‘ recombination ...’ y reemplazarlo por ‘]’ - eliminar ‘genosp.’ - sustituir espacios por guiones bajos

Noten el uso de expresiones regulares como `.*` y `[:space:]`

```
sed 's/ Bra/ [Bra/; s/|gb.*| /|/; s/Bradyrhizobium /B/; s/genosp\. //; s/ recomb.*//; s/[:space:]]/_/_/; s/|gb.*| /|/; s/Bradyrhizobium /B/; s/genosp\. //; s/ recomb.*//; s/[:space:]]/_/_/;
```

```
## >EU574327.1_[Bliaoningense_strain_ViHaR5]
## >EU574326.1_[Bliaoningense_strain_ViHaR4]
## >EU574325.1_[Bliaoningense_strain_ViHaR3]
## >EU574324.1_[Bliaoningense_strain_ViHaR2]
## >EU574323.1_[Bliaoningense_strain_ViHaR1]
## >AY591544.1_[Bjaponicum_bv._genistearum_strain_BC-P14]
## >AY591543.1_[Bbeta_strain_BC-P6]
## >AY591542.1_[Bcanariense_bv._genistearum_strain_BC-P5]
## >AY591541.1_[Bcanariense_bv._genistearum_strain_BC-C2]
## >AY591540.1_[Balpha_bv._genistearum_strain_BC-C1]
```

8. Cuando estamos satisfechos con el resultado, guardamos la salida del comando en un archivo usando `>` para redirigir el flujo de STDOUT a un archivo de texto

```
sed 's/ Bra/ [Bra/; s/|gb.*| /|/; s/Bradyrhizobium /B/; s/genosp\. //; s/ recomb.*//; s/[:space:]]/_/_/;
```

5.3.3 Generación automática de archivos FASTA especie-específicos (avanzado)

9. Convertir archivos FASTA a formato “FASTAB” usando `perl` 1-liners.

Vamos a transformar los FASTAS de tal manera que las secuencias queden en la misma línea que su cabecera, separada de ésta por un tabulador. Esto puede ser muy útil para filtrar el archivo resultante con `grep`. Veamos un ejemplo:

```
perl -pe 'unless(/^>){s/\n//g}; if(/^>){s/\n\t/g}; s/>/\n>/' recA_Bradyrhizobium_vinuesa.fnaed | head
```

```
##
## >EU574327.1_[Bliaoningense_strain_ViHaR5] ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574326.1_[Bliaoningense_strain_ViHaR4] ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574325.1_[Bliaoningense_strain_ViHaR3] ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574324.1_[Bliaoningense_strain_ViHaR2] ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574323.1_[Bliaoningense_strain_ViHaR1] ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574322.1_[Bliaoningense_strain_ViHaG8] ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574321.1_[Bliaoningense_strain_ViHaG7] ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574320.1_[Bliaoningense_strain_ViHaG6] ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574319.1_[Byuanmingense_strain_ViHaG5] ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCTCCGGCT
```

```
perl -pe 'unless(/^>){s/\n//g}; if(/^>){s/\n\t/g}; s/>/\n>/' recA_Bradyrhizobium_vinuesa.fnaed > recA
```

10. Filtrar el archivo `fnaedtab` generado en 9 para obtener solo las secuencias de `B._yuanmingense` del mismo, guardarlo en un archivo y convertirlo de nuevo a formato FASTA.

```
grep yuanmingense recA_Bradyrhizobium_vinuesa.fnaedtab | head -5
```

```
## >EU574319.1_[Byuanmingense_strain_ViHaG5] ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574318.1_[Byuanmingense_strain_ViHaG4] ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574297.1_[Byuanmingense_strain_InRo02] ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCTCCGGCT
## >EU574296.1_[Byuanmingense_strain_InKo02] ATGAAGCTCGGCAAGAACGATCGTCCATGGACATCGAGGCGGTCTCTCCGGCT
## >EU574295.1_[Byuanmingense_strain_InKo01] ATGAAGCTCGGCAAGAACGATCGTCCATGGACATCGAGGCGGTGTCTCCGGCT
```

```
grep yuanmingense recA_Bradyrhizobium_vinuesa.fnaedtab > recA_Byuanmingense.fnaedtab
```

11. Estas dos líneas no contienen nada nuevo en cuanto a sintaxis. Simplemente llamamos a perl para sustituir los tabuladores por saltos de línea y así reconstituir el FASTA.

```
perl -pe 'if(/^>){s/\t/\n/}' recA_Byuanmingense.fnaedtab | head -5
```

```
## >EU574319.1_[Byuanmingense_strain_ViHaG5]
## ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTCGCTCGGGCTCGATATCGCGCTCGGCATCGGCGGCTTGCCCAAGG
## >EU574318.1_[Byuanmingense_strain_ViHaG4]
## ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTCGCTCGGGCTCGATATCGCGCTCGGCATCGGCGGCTTGCCCAAGG
## >EU574297.1_[Byuanmingense_strain_InRo02]
```

```
perl -pe 'if(/^>){s/\t/\n/}' recA_Byuanmingense.fnaedtab > recA_Byuanmingense.fna
```

12. Llamar a un bucle for de shell para generar archivos fastab para todas las especies

```
for sp in $(grep '^>' recA_Bradyrhizobium_vinuesa.fnaedtab | cut -d_ -f2 | sort -u | sed 's/\[//'); do
  grep "$sp" recA_Bradyrhizobium_vinuesa.fnaedtab > "recA_${sp}.fnaedtab"
done
```

13. Veamos el resultado

```
ls *fnaedtab
```

```
## recA_Balpha.fnaedtab
## recA_Bbeta.fnaedtab
## recA_Bcanariense.fnaedtab
## recA_Belkanii.fnaedtab
## recA_Bjaponicum.fnaedtab
## recA_Bliaoningense.fnaedtab
## recA_Bradyrhizobium_vinuesa.fnaedtab
## recA_Bsp..fnaedtab
## recA_Byuanmingense.fnaedtab
```

```
head -5 recA_Bjaponicum.fnaedtab
```

```
## >EU574316.1_[Bjaponicum_strain_NeRa16] ATGAAGCTCGGCAAGAACGACCGGTGCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCT
## >EU574315.1_[Bjaponicum_strain_NeRa15] ATGAAGCTCGGCAAGAACGACCGGTGCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCT
## >EU574314.1_[Bjaponicum_strain_NeRa14] ATGAAGCTCGGCAAGAACGACCGGTGCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCT
## >EU574313.1_[Bjaponicum_strain_NeRa12] ATGAAGCTCGGCAAGAACGACCGGTGCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCT
## >EU574312.1_[Bjaponicum_strain_NeRa11] ATGAAGCTCGGCAAGAACGACCGGTGCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCT
```

14. Finalmente convertimos todos los archivos fnatabed a FASTA con el siguiente bucle for:

```
for file in $(ls *fnaedtab | grep -v vinuesa); do perl -pe 'if(/^>){s/\t/\n/}' $file > ${file%.*}.fas;
```

15. Visualizemos las cabeceras de dos archivos FASTA especie-específicos

```
grep '>' recA_Bjaponicum.fas | head -5
```

```
## >EU574316.1_[Bjaponicum_strain_NeRa16]
## >EU574315.1_[Bjaponicum_strain_NeRa15]
## >EU574314.1_[Bjaponicum_strain_NeRa14]
## >EU574313.1_[Bjaponicum_strain_NeRa12]
## >EU574312.1_[Bjaponicum_strain_NeRa11]
```

16. y confirmemos que son fastas regulares

```
head -6 recA_Bjaponicum.fas
```

```
## >EU574316.1_[Bjaponicum_strain_NeRa16]
## ATGAAGCTCGGCAAGAACGACCGGTGCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTCGGGCTCGACATTGCACTGGGGATCGGCGGTCTGCCCAAGG
```

```
## >EU574315.1_[Bjaponicum_strain_NeRa15]
## ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTCGGGCTCGACATTGCACTGGGGATCGGCGGTCTGCCCAAGG
## >EU574314.1_[Bjaponicum_strain_NeRa14]
## ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTCGGGCTCGACATTGCGCTGGGGATCGGCGGTCTGCCCAAGG
```

17. si quieren, borren todos los archivos generados, para empezar con un directorio de trabajo limpio, para repetir el ejercicio ;)

```
rm *fnaed *fnaedtab *fas Stenotrophomonas_complete_genomes_and_ftp_paths.txt empty_file Pseudomonas_spe
```

6 Reto de programación - ejercicio de parseo de archivos FASTA

Como ejercicio, para repasar lo que hemos aprendido en esta sesión les propongo repetir el ejercicio de parseo de archivos FASTA pero con secuencias del gen *rpoB* de *Bradyrhizobium*

Los que no tengan instalado MobaXterm, tendrán el reto adicional de instalarlo y familiarizarse con él.

6.1 Inspección y estadísticas básicas de las secuencias descargadas

1. Descargar las secuencias de NCBI usando el portal ENTREZ nucleotides con: ‘*Bradyrhizobium*[orgn] AND *vinuesa*[auth] AND *rpoB*[gene]’
2. Renombra el archivo descargado a *rpoB_Bradyrhizobium_vinuesa.fna*
3. ¿Cuántas secuencias hay en el archivo *rpoB_Bradyrhizobium_vinuesa.fna*?
4. Explora la cabecera y cola del archivo con `head` y `tail`
5. Despliega las 5 primeras líneas de cabeceras fasta usando **grep** y **head** para explorar su estructura en detalle
6. Calcula el número de generos que contiene el archivo FASTA
7. Calcula el número de especies que contiene el archivo FASTA
8. Imprime una lista ordenada de mayor a menor, del numero de especies que contiene el archivo FASTA

6.2 Edición de las cabeceras FASTA mediante herramientas de filtrado de UNIX

1. Explora nuevamente todas las cabeceras FASTA del archivo *rpoB_Bradyrhizobium_vinuesa.fna* usando **grep** y `less`
2. Simplifica las cabeceras FASTA usando el comando **sed** (stream editor)

El objetivo es eliminar redundancia y longitud excesiva de las cabeceras FASTA, así como todos los caracteres ‘(, ; :)’ que impedirían el despliegue de un árbol filogenético, al tratarse de caracteres reservados del formato NEWICK. Dejar solo el numero de accesoión, así como el género, especie y cepa indicados entre corchetes.

3. Cuando estén satisfechos con el resultado, guarden la salida del comando en un archivo llamado *rpoB_Bradyrhizobium_vinuesa.fnaed*