Sesión 1. Introducción al biocómputo en sistemas GNU/Linux

Pablo Vinuesa, Centro de Ciencias Genómicas - UNAM

2019-08-09

Contents

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: 2019-08-09

1.1 Licencia y términos de uso

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
```

```
## 800px-Evolución_UNIX.png
## assembly_summary.txt.gz
## Evolución_UNIX.png
## fetch_recA_bradys_vinuesa_nuccore_screenshot.png
## github_TIB-filoinfo_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
```

```
# mostrar todos (-a all) los archivos, incluidos los ocultos
ls -a | tail
```

```
## sesion_remota_bonampak_capt_pantalla1.png
## sesion_remota_bonampak_capt_pantalla2.png
## sesion_remota_bonampak_capt_pantalla.png
## Taller_CG-FC_pagina_web_CCG.png
## TIB2019-T3
## working_with_linux_commands.code
## working_with_linux_commands.html
## working_with_linux_commands.log
## working_with_linux_commands.Rmd
## working_with_linux_commands.tex
```

2.2.2.1 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  9 17:57 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 349 root root          0 ago  9 17:55 proc
## drwx------   5 root root       4096 jul  4 12:57 root
## drwxr-xr-x  35 root root       1060 ago  9 17:56 run
## drwxr-xr-x   2 root root      12288 ago  9 09:09 sbin
```

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

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

```
## bash
## brltty
## bunzip2
## busybox
## bzcat
```

```
# 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 fina
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
```

2.2.3 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/brltty
## /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
```

```
## 800px-Evolución_UNIX.png
## assembly_summary.txt.gz
## Evolución_UNIX.png
## fetch_recA_bradys_vinuesa_nuccore_screenshot.png
## github_TIB-filoinfo_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
```

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 fisicas 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 [ "$hostn" == "buluc" ]; then
     ln -s /home/vinuesa/cursos/TIB2019-T3/sesion1_linux/data/linux_basic_commands.tab comandos_de_linu
elif [ "$hostn" == "alisio" ]; then
  ln -s /home/vinuesa/Cursos/TIB/TIB19-T3/sesion1_intro2linux/linux_basic_commands.tab comandos_de_linu
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 19440
## -rw-r--r-- 1 vinuesa vinuesa  110128 ago  8 18:05 800px-Evolución_UNIX.png
## -rw-r--r-- 1 vinuesa vinuesa 6780296 ago  8 18:05 assembly_summary.txt.gz
## lrwxrwxrwx 1 vinuesa vinuesa      78 ago  9 19:46 comandos_de_linux.tab -> /home/vinuesa/Cursos/TIB/
## -rw-r--r-- 1 vinuesa vinuesa  438308 ago  8 18:05 Evolución_UNIX.png
## -rw-r--r-- 1 vinuesa vinuesa  222602 ago  8 18:05 fetch_recA_bradys_vinuesa_nuccore_screenshot.png
## -rw-r--r-- 1 vinuesa vinuesa   87065 ago  8 18:05 github_TIB-filoinfo_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
```

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 A
## 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 A
## 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    Vers
##      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 A
## 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 simiar 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 xpresiones 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 omiti
-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, done:
  - // 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
  - \m 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 tuberias 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 ensambles 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 colu
## # assembly_accession bioproject  biosample   wgs_master  refseq_category taxid   species_taxid   orga
## GCF_000010525.1  PRJNA224116 SAMD00060925        representative genome   438753  7   Azorhizobium cau
## 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 (sor -u; sort -nrk2; sort -dk1)
- wc cuenta líneas, palabras y caracteres (wc -l)
- uniq regresa listas de valores únicos (uniq -c)
- grep Filtra las lineas 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 especie
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 lineas que contienen Acinetobacter o Stenotrophomonas

```
# filtra y cuenta las lineas 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 orden
zgrep -E 'Acinetobacter|Pseudomonas|Klebsiella' assembly_summary.txt.gz | cut -f 8 | cut -d' ' -f1 |sor
```

```
##    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 lineas que contienen Filesystem o Text processing y ordénalas alfabéticamente según las en
# eliminando las entradas de Candidatus y [Pseudomonas]
zgrep -E 'Acinetobacter|Pseudomonas|Klebsiella' assembly_summary.txt.gz | cut -f 8 | cut -d' ' -f1 | gr
```

```
##    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_0000
## Stenotrophomonas maltophilia K279a   ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/072/485/GCF_0000
## Stenotrophomonas maltophilia JV3 ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/223/885/GCF_00022388
```

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

Vermos aquí unas pocas construcciones muy básicas de probramación Shell

3.1 Asignación de variables

- La sintaxis básica de asignación es:

```
varName=VALUE
```

- para recuperar el valor de una varialbe, 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
```

- para capturar la salida de un comando usammos $(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 <9ago2019>
```

- 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 (cualqier 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 escibimos 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  9 19:46 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
```

3.2.3.1 La versión corta de test [condición] && ejecuta orden1 && ejecuta orden2 ...

```
# 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
```

3.2.4 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 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 un comando como el siguiente:

for file in *.faa; do clustalo -i $file -o ${file%.faa}_cluoAln.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
## getopts  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
```

---

4 El lenguaje de procesamiento de patrones AWK

AWK es un lenguaje de programación diseñado para procesar datos de texto, ya sean ficheros 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 $wile

  - operadores aritméticos +, -, *, /, %, =, ++, –, +=, -=, . . .)
  - operadores boleanos ||, &&
  - 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 definer FS="\t", para que tome como campos sólo a aquellos separados por tabul
# 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 enviar
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 accesión revisado v2, publicados en 2019 para genomas en estado Scaffold

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

```
## 31
```

- veamos las entradas de la tabla que tienen un número de accesión 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 && condición2 && 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_ASM23
## Helicobacter pylori GAM100Ai ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/310/005/GCF_000310005.2_A
## Helicobacter pylori GAM101Biv    ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/344/945/GCF_000344945
```

- veamos las entradas de la tabla que tienen un número de accesión 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 && condición2 && condición3, acción
zcat assembly_summary.txt.gz | awk 'BEGIN{FS="\t"; OFS=" ~~~ "} $16 ~ /v2$/ && $15 ~ /201./ && $12 == "S

## 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/Pseudo

## 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 genarar 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/Pseudo
```

- 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
```

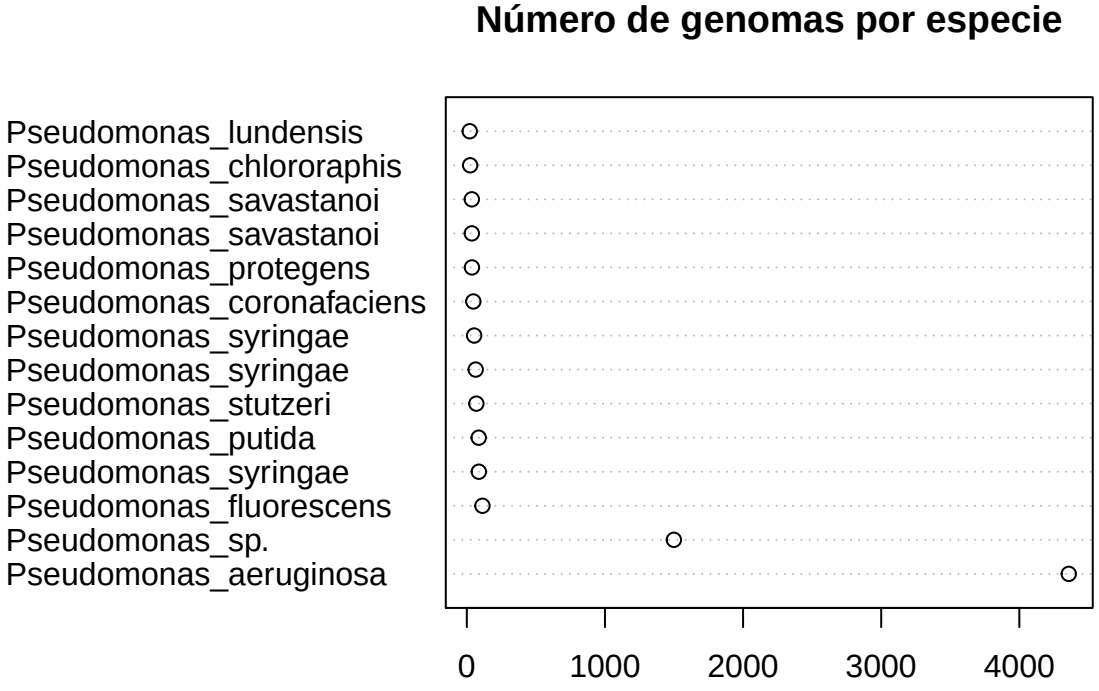
```
# 1. leemos el archivo a una estructura de datos tipo dataframe
dfr <- read.csv(file = "Pseudomonas_species_with_gt_20_genomes.csv", header = TRUE)

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")
```



5.1.1 Reto de programación *awk* y *R*

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

```
## >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:]]/_/g
```

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

```
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]    ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574318.1_[Byuanmingense_strain_ViHaG4]    ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574297.1_[Byuanmingense_strain_InRo02]    ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574296.1_[Byuanmingense_strain_InKo02]    ATGAAGCTCGGCAAGAACGATCGCTCCATGGACATCGAGGCGGTCTCCTCCGGCTC
## >EU574295.1_[Byuanmingense_strain_InKo01]    ATGAAGCTCGGCAAGAACGATCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
```

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

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

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

```
## >EU574319.1_[Byuanmingense_strain_ViHaG5]
## ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTCGCTCGGGCTCGATATCGCGCTCGGCATCGGCGGCTTGCCCAAGGC
## >EU574318.1_[Byuanmingense_strain_ViHaG4]
## ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTCGCTCGGGCTCGATATCGCGCTCGGCATCGGCGGCTTGCCCAAGGC
## >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]   ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCGGGTTCTCTC
## >EU574315.1_[Bjaponicum_strain_NeRa15]   ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTC
## >EU574314.1_[Bjaponicum_strain_NeRa14]   ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTC
## >EU574313.1_[Bjaponicum_strain_NeRa12]   ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTC
## >EU574312.1_[Bjaponicum_strain_NeRa11]   ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTC
```

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]
## ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCGGGTTCTCTCGGGCTCGACATTGCACTGGGGATCGGCGGTCTGCCCAAGG
## >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
```

---

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 lineas 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 accesión, 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