Tutorial en Linux

Herman Jaramillo

February 12, 2024

1 Introducción

1.1 Objetivos

Estas notas pretenden ilustrar algunos aspectos importantes para usuarios del sistema `Unix` (más concretamente `Linux`). El objetivo es que el estudiante maneje el sistema con cierto conocimiento que le permita navegar por el mismo mediante la ejecución de comandos y la elaboración de `shell scripts`.

1.2 Historia

Los inicios del sistema operacional Unix [1] se dieron en el Instituto Tecnológico de Massachussets (MIT, for sus siglas en inglés), Laboratorios Bell (Bell Laboratories) y General Electric (GE) como soluciones de multiusuario y multitarea para mainframes. No fue hasta los 1970s que el sistema se desarrolló en laboratorios Bell de AT&T (American Telephone and Telegraph Company) con el trabajo de Ken Thompson, Dennis Ritchie (inventor de C) y otros. Aunque inicialmente el sistema estaba supuesto a ser usado internamente dentro de AT&T se extendió la licencia a otras entidades académicas y comerciales, tales como la Universidad de Berkely (BSD), Microsoft (Xenix), IBM (AIX), HP (UX) , Sun Mucrosystems (Solaris), Apple (MacOS). La licencia de Unix ha pasado por muchas manos. En este momento está en manos del Open Group [2]

1.3 Typos de Unix

Desde su invención, el número de tipos de sistemas Unix ha crecido considerablemente. Dentro de ellos los más comunes, usados para computadores personales, son MacOS X,

[1] https://en.wikipedia.org/wiki/Unix
[2] https://en.wikipedia.org/wiki/The_Open_Group

Solaris (para SUN) y GNU/Linux. Dentro de los sistemas Linux [3], (Linux-Wikipedia) que tiene a Linus Torvalds como su principal desarrollador del "kernel" existe una lista considerable de tipos de Linux. Por ejemplo, encontramos CentOS, Fedora(Red Hat), OpenSUSE y Ubuntu entre otros. Este documento se preparó con el sistema operacional de Ubuntu. La página en este enlace [4] muestra con detalle una lista de distribuciones de Linux y componentes dentro de las distribuciones.

El código (libre) de Linux se encuentra en GitHub. [5]

1.4 Componentes

Las principales componentes de Unix son el Kernel y la Shell que describimos a continuación.

1.4.1 Kernel

En el kernel se ubica el corazón de Linux. Manejo de memoria, comunicaciones, manejo de procesos y almacenamiento de archivos, entre otros.

1.4.2 Shell

La "Shell" es la interface entre el usuario y el kernel. Una vez el usuario hace login y el sistema verifica la información la "shell" entra en acción. La "shell" se ejecuta en modo comando a través de una terminal. Hay muchos tipos de "shells" disponibles in Linux. Por ejemplo, ksh, bsh, bash, csh, tcsh, etc. Aunque shell es una palabra en inglés que traduce "concha" nos seguimos refiriendo con su nombre en inglés por ser altamente usado en la comunidad de Linux. En este tutorial solo nos concentraremos en la shell tipo "bash" (Born again shell). Esta shell es la que se usa por defecto en el sitema Ubuntu, el cual usamos en esta clase. Los comandos ejecutados en una shell se pueden completar en algunas ocaciones usando la tecla `TAB` . A esto se le conoce en inglés como *command completion*, es decir, completación de comandos. La completación de comandos no solo ahorra escritura, y por lo tanto tiempo de trabajo, sino que sirve de control de calidad en la escritura del comando. Si al presionar `TAB` no sale ninguna opción probablemente halla un error en lo que se ha escrito del comando hasta el momento o, por ejemplo, un archivo sobre el cual se quiere trabajar no existe. También puede ocurrir que el comando que se quiere ejecutar no está disponible para el tipo de archivo que se quiere usar. Si

[3] https://www.linux.org/

[4] https://en.wikipedia.org/wiki/List_of_Linux_distributions

[5] https://github.com/torvalds/linux

hay archivos que comienzan con el mismo nombre, `TAB` lista todas las opciones para que el usuario escoja la deseada.

1.4.3 Escritorios (desktops)

Los escritorios son ambientes donde se trabaja. Es la forma como la pantalla se presenta al usuario. Color de fondo, íconos, barras de herramientas, diseño de ventanas etc. Hay una colección de escritorios para Linux. Los más comunes son el escritorio KDE [6] y el GNOME. [7] El escritorio KDE se desarrolla sobre las herramientas para desarrollo de interfaces gráficas (GUI=Graphical User Interface) Qt [8] el cual es de uso comercial, mientras que Gnome se desarrolla sobre GTK [9] que es de uso libre.

La página Best Desktop Environments for Linux [10] muestra un conjunto de escritorios usados en Linux.

1.5 Cree un nuevo usuario

Antes de comenzar el curso vamos a crear un nuevo usuario sobre el cual va a trabajar durante el transcurso de la clase.

Una vez hace `login` en el sistema, usted debe tener una cuenta de superusuario, de otra forma no puede crear un nuevo usuario y le toca trabajar con la cuenta que se le asigne en el curso. Vamos entonces a asumir que usted tiene una cuenta de superusuario.

Para crear una nueva cuenta de usario abra una terminal (por ejemplo con Ctrl+Alt+T. Luego explicamos más acerca de como abrir una terminal). Escribimos

```
>sudo adduser usuario
```

donde "`>`" es el prompt (no se escribe este) y `usuario` es el nombre de usuario que le quiere dar al usuario. El sistema le hace algunas preguntas sobre la contraseña y datos personales. Por ejemplo:

```
:home>sudo adduser pepito
Adding user 'pepito' ...
Adding new group 'pepito' (1001) ...
Adding new user 'pepito' (1001) with group 'prueba' ...
Creating home directory '/home/pepito' ...
```

[6] https://en.wikipedia.org/wiki/KDE
[7] https://en.wikipedia.org/wiki/GNOME
[8] https://www.qt.io/
[9] https://www.gtk.org/
[10] https://itsfoss.com/best-linux-desktop-environments/

```
Copying files from ‘/etc/skel’ ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for prueba
Enter the new value, or press ENTER for the default
Full Name []: Pepito Perez
Room Number []: 109
Work Phone []: 3032242656
Home Phone []: 3031111111
Other []: nada
Is the information correct? [Y/n] y
```

Se puede verificar que el usuario está en el sistema mediante los siguientes comandos:

```
>cd \home
>ls
```

En mi caso obtengo

```
herman   pepito
```

A partir de este momento cada uno va a trabajar sobre su cuenta de usuario personal.

1.6 Conozca su sistema

En este curso usamos, para casi todas las tareas, la terminal y el modo comando. El símbolo `>` representa el "prompt". Lo que está a la izquierda de donde se digita el comando. A continuación mostramos algunos comandos útiles para conocer su systema.

- El nombre de su máquina mediante el comando `>hostname`. Por ejemplo, mi sistema reporta:

  ```
  >hostname
  herman-Lenovo-V510z
  ```

  Una extensión de este comando es `hostnamectl`. Por ejemplo,

```
>hostnamectl
Static hostname: herman-Lenovo-V510z
Icon name: computer-desktop
Chassis: desktop
Machine ID: 8abc91df838141fd8249b5737dec28bb
Boot ID: ef3587ab81b845b382cc4d30ac1302a7
Operating System: Ubuntu 20.04.6 LTS
Kernel: Linux 5.15.0-89-generic
Architecture: x86-64
```

- Sistema operativo mediante el comando `>uname`. Por ejemplo, en mi sistema se obtiene

```
>uname
Linux
```

  Estos comandos tienen opciones. Por ejemplo,

```
>uname -a
Linux herman-Lenovo-V510z 5.15.0-89-generic #99~20.04.1-Ubuntu SMP
Thu Nov 2 15:16:47 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
```

  en este caso la opción "`-a`" indica `all` o todo. Simpre se puede usar el manual en linea, por ejemplo `man uname` para averiguar que opciones tiene un comando y más información del comando.

- Otra forma de verificar su versión del sistema es con el comando `lsb_release -a`. Por ejemplo,

```
>lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 20.04.6 LTS
Release: 20.04
Codename: focal
```

- El archivo `/etc/os-release` también proporciona información acerca de su versión de Linux. Por ejemplo,

```
>cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.6 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.6 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies
/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

- Si desea conocer escritorio (desktop) que está usando use

```
>echo $XDG_CURRENT_DESKTOP
ubuntu:GNOME
```

  Más abajo explicamos el comando `echo` junto con las variables ambientales como, por ejemplo, `XDG_CURRENT_DESKTOP`.

- Familia con prefijo `ls` (list). Si se escribe en el prompt `ls` y se usa la tecla `TAB` se ven más de una veintena de comandos que comienzan con `ls` . Listamos acá algunos que nos dicen sobre el sistema.
  - lscpu : Lista información acerca de o los CPU del sistema. use conexión con `more` para paginar la salida.
  - lshw : Lista de información sobre el hardward del equipo. Para mayor información sobre la salida de este comando se requiere contraseña de super-usuario.
  - lsmem : Lista información de la memoria del sistema.
  - lsof : Lista archivos abiertos (open files) en el sistema.
  - lstopo : Lista la topología del sistema.
  - lsusb : Lista los dispositivos USB .

Actividad # 1: Use los comandos `lshw` y `lscpu` para encontrar la memoria de su máquina, el número de CPUs y la velocidad (frecuencia del relog en GHz). Ayuda: Puede almacenar el resultado de su comando en un archivo. Por ejemplo, el comando "`lshw > ~/myHardware`" almacena la información sobre su sistema en el archivo `myHardware` en su `Home directory`. Más abajo hablamos del `Home directory`.

2 Estructura de Archivos en Linux

La estructura en que los archivos se almacenan en Linux está representada como un árbol y se muestra en la Figura 1. En la parte superior está la raíz (root) de donde se desprende todo el árbol . Aunque en la raíz pueden haber muchos más directorios los que se muestran en la figura son los principales. Explicamos la lista en la figura.

(i) El directiorio `bin` indica donde se almacenan los archivos ejecutables.

(ii) El directorio `tmp` indica es un área para el almacenamiento de archivos temporales. Normalmente cuando el computador se reinicia (reboot) el contenido de `tmp` se borra. Es muy útil cuando se desea almacenar archivos grandes que solo duran mientras se tiene una sesión (prendida y apagada del equipo) de trabajo. Si el usuario piensa que los archivos que va a crear no son de gran importancia pero quiere que no se pierdan mientras los está editando es mejor crear un directorio de basura (por ejemplo `Scratch`) donde haga sus pruebas. La razón es que si hay un corte de luz, se pierde lo que se está haciendo en `tmp`.

(iii) El directorio `home` de usuario es el área de trabajo de los usuarios. Acá es donde nosotros desarrollamos nuestros códigos y documentos. En el ejemplo vemos dos usuarios "`Juan`" y " `Pedro`". De ahora en adelante escribimos home de usuario, o simplemente home, para indicar el directorio donde nosotros como usuarios entramos cuando hacemos login.

(iv) El directorio `media` ofrece el lugar donde se detectan los dispositivos de almacenamiento externo tales como memorias USB, DVDs, etc.

(v) El directorio `etc` ofrece, entre otros, los nombres de la máquina y aquellas que se conectan con ella, las claves de acceso para todos los usuarios y otros. Este directorio se creó históricamente para localizar todos los archivos que no tenían un lugar bien definido. Es decir, es un directorio de misceláneos.

(vi) El directorio `usr` muestra el lugar donde se instalan la mayoría de las librerías (aplicaciones) del usuario.

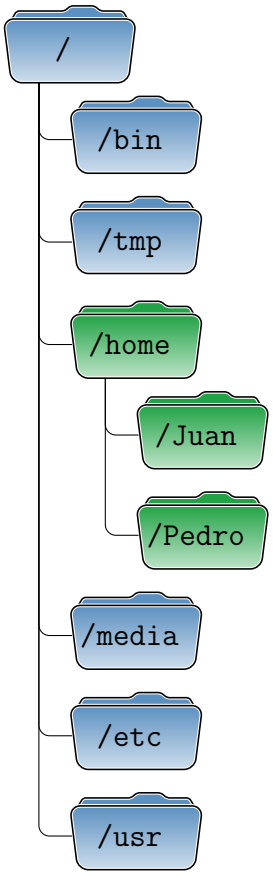

Figure 1: Estructura de los Archivos en Linux en forma de árbol

Existen otros directorios tales como `dev` , `mnt` los cuales no estudiamos en este curso. Sugerimos visitar la página en este enlace [11] para una información mucho más completa acerca de la estructura de los archivos en Linux.

Cuando ingresamos al sistema, automáticamente quedamos localizados en nuestra área de trabajo dentro del directorio `home`. Por ejemplo, si nuestro nombre de usuario es `Pedro`, al prender el equipo estamos localizados en el directorio `/home/Pedro`.

Actividad de creación de archivos #2: Los archivos tiene dos formas fundamentales. Un archivo es un directorio (es decir contiene otros archivos, también se conoce como carpeta o, en inglés directory or folder) o es una rama terminal del árbol (es decir, es

[11] https://www.linux.com/blog/learn/intro-to-linux/2018/4/linux-filesystem-explained

un documento. Podemos simplemente decir archivo directorio versus archivo regular o rama versus hoja). Iniciamos el equipo y abrimos una terminal. Esta se puede buscar dentro de las herramientas en la barra lateral o abriendo "`Activities`" en la esquina izquierda. Se abre una pestaña de búsqueda donde puede escribir `xterm`. Otra forma de abrir una terminal es con la combinación `Ctrl+Alt+T`. Una vez se tiene una terminal abierta y con el ratón enfocado en esta terminal, se puede generar una nueva con el comando `Ctrl+Shift+n`. El primer comando que vamos a ejecutar es

```
> man man
```

Acá "`>` " representa el "prompt", es decir, el indicador que espera por una entrada desde el usuario. En todos los comandos mostrados en este tutorial el "prompt" no se escribe, el "prompt" lo muestra el sistema. El "prompt" se puede configurar para que aparezca, por ejemplo, el nombre del directorio sobre el cual se está parado. Más adelante creamos una actividad para esto. El comando de arriba, es el "manual" del "manual" es decir indica como se usa el manual en Linux. Solo los muy dedicados leen el manual del manual. La estructura del manual para cada comando es la siguiente:

- **Name**: Nombre del comando.
- **Synopsis**: el comando con opciones, vista rápida.
- **Description**: Descripción del comando.
- **Examples**: Ejemplos del uso.
- **Overview**: Vista general del comando.
- **Defaults**: Parámetros por defecto.
- **Options**: Descripción detallada de las opciones, una a una.

Esta lista no es exhaustiva, y algunas veces el orden no es el mismo. Por ejemplo, aparecen, a veces, descripciones tales como: **author** (autor), **reporting bugs** (reporte de errores), **copyright** (derechos de copia) y **see also** (ver además), etc.

El siguiente comando a ejecutar es

```
> ls
```

Este comando, por sus siglas en inglés "List Directory" , muestra una lista del contenido del directorio. Para consultar el manual de este comando escribimos

```
> man ls
```

Vemos que hay muchas opciones que se pueden ejecutar con `ls` . Por ejemplo, `ls -a` lista todos los archivos. Aquellos archivos que comienzan con punto "." están normalmente ocultos. Finalmente, `ls --color` muestra aquellos archivos con colores indicando cuales son directorios y cuales son archivos simples. Otra forma de averiguar el significado de comandos es mediante el uso del comando "info". Por ejemplo,

```
> info ls
```

produce otro listado diferente con información del comando `ls`. Para pasar a la siguiente página se pulsa la barra espaciadora. Si se quieren hacer una búsqueda se puede hacer oprimiento la tecla "`/`" y escribiendo la palabra que se quiere buscar (por ejemplo `Examples`). Para buscar hacia atrás se oprime la tecla "`?`". Para salirse del pantallazo de "info" se oprime la letra "`q`". Estos comandos también aplican para la visualización con el comando `man`.

Normalemente el sistema Linux no trae el comando `tree` . Este comando permite ver los directorios y subdirectorios como un árbol. Si su sistema no trae el comando `tree` lo podemos instalar con la siguiente instrucción.

```
> sudo apt install tree
```

Acá:

1. `>` representa el "prompt" es decir el indicador que espera por una entrada del usuario.

2. `sudo` es un comando que pone al usuario en modo administrador. Es decir, le da privilegios para instalar archivos en áreas reservadas para el administrador. En inglés `sudo` significa *superuser do* o haga las veces de superusuario.

3. `apt` Significa herramienta de paquetes avanzados por sus siglas en inglés *Advance Package Tool* . Es una herramienta supremamente útil y segura para instalar nuevo software en el computador. Existen otras herramientas para instalar software. Por ejemplo, synaptic [12] y Gnome Software [13]. Estas herramientas proveen una interface gráfica (GUI que en inglés es Graphical User Interface) y también son seguras.

4. `tree` significa que se muestra el directorio y todos los subdirectorios en forma de árbol.

Vamos a crear en el directorio home un subdirectorio con el nombre `Herramientas_Computacionales` . Esto lo hacemos mediante el comando

[12] https://www.nongnu.org/synaptic/
[13] https://wiki.gnome.org/Apps/Software

```
> mkdir Herramientas_Computacionales
```

El comando `mkdir` significa cree directorio (del inglés "make directory "). Revisemos que el nuevo directorio `Herramientas_Computacionales` está allí. Lo vamos a hacer con varias herramientas. No recomendamos palabras con espacios. En Linux se prefiere usar el guion bajo "`_` " para unir palabras. Tampoco se sugiere usar guion simple "`-` " para unir palabras. y buscamos el nuevo directorio

```
> ls --color
```

Debe verse más fácil puesto que es de color azul.

```
> ls -lt
```

Está listado de primero, pues el argumento `-lt` significa de forma larga (esto lo explicamos más adelante) y en orden reverso. Es decir, el último creado se muestra de primero. Por último corremos

```
> tree | more
```

para ver por pantallazos todo el árbol a partir del punto donde estamos localizados. Para salirse de la aplicación oprima la tecla `q` (quit). El comando

```
> ls -R
```

también lista los archivos en forma recursiva.

Actividad # 3: Trate con el comando `tree` y el comando `ls -R` en varios directorios del sistema.

En este momento nos desplazamos al directorio `Herramienas_Computacionales` con el comando

```
> cd Herramientas_Computacionales
```

El comando `cd` indica cambio de directorio (del inglés "change directory"). En este momento estamos parados en el directorio `Herramientas_Computacionales` . Si corremos el comando

```
> ls
```

no encontramos información, pues el directorio está vacío.

Quien soy y donde me encuentro: Hay un par de comandos que nos ayudan a reconocer el entorno donde estamos parados. El primero es

```
> whoami
```

(es decir "quien soy yo") y el otro comando es

```
> pwd
```

que significa "en que directorio estoy parado". Del inglés "path working directory" En este momento vamos a crear tres directorios como sigue

```
> mkdir animales
> mkdir vegetales
> mkdir minerales
```

Estos tres comandos se pudieron haber ejecutado con un simple comando

```
>mkdir animales vegetales minerales
```

Luego revisamos el trabajo con los dos comandos

```
> ls
> tree
```

Ahora nos vamos a dirigir al directorio "animales" con el comando

```
> cd animales
```

Vamos a crear tres archivos de texto en este directorio. Una forma rápida de crear archivos vacíos, es mediante el comando "`touch`". Por ejemplo, el comando

```
> touch nada
```

crea un archivo llamado "`nada`". Hagamos ahora los comandos

```
> ls
> ls -l
```

El argumento `-l` indica que se lista el archivo en forma larga. Vamos ahora a explicar el significado de la salida de este comando. Por ejemplo, en mi caso obtengo

```
:animales>ls -l
total 0
-rw-r--r-- 1 hjaramillo hjaramillo 0 feb 25 14:54 nada
```

(i) El primer guión `-` indica que el archivo es regular (es decir, no es directorio). Si fuese directorio tendría la letra `d` .

(ii) Luego siguen campos para 9 caracteres. Los tres primeros caracteres ofrecen información acerca del usuario. Los segundos tres caracteres acerca del grupo al que pertenece el usuario y por último, los tres últimos caracteres ofrecen información para cualquier usuario del sistema en la red. En cada uno de estos espacios puede ir una de las letras `r` ,`w` , `x` . La letra `r` significa que el archivo puede ser leido, la letra `w` significa que el archivo se puede sobre-escribir (o borrar) y la letra `x` que el archivo se puede ejecutar.

(iii) Luego aparece el dueño del archivo y seguido el grupo dueño del archivo.

(iv) Seguidamente se muestra el tamaño del archivo en bytes (0 en este caso dado que el archivo está vacío por la forma como se creo).

(v) Luego se muestra la fecha de creación. Existe el comando

```
> date
```

que muestra la fecha y hora al momento de correr el comando. Se puede usar el comando `date` para verificar la información suministrada arriba.

(vi) La hora de creación.

(vii) Finalmente el nombre del archivo.

Existen, al menos, tres formas de crear archivos nuevos:

- mediante el comando `touch`. Ya vimos ejemplos.
- usando el comando `echo`. Por ejemplo,

```
>echo ‘‘soy el rey leon’’ > leon
```

Este comando crea un archivo con el nombre `leon`. Para ver el contenido puede correr el comando `>cat leon`. Si quiere agregar lineas al final del archivo debe cambiar el signo "`>`" por "`>>`" (append).

- Mediante un editor. Hablamos de editores más adelante.

Actividad # 4: Agregue una línea al archivo `leon` con la frase "`.soy el rey de la selva`".

El privilegio de un archivo (leer, escribir, ejecutar) para el usuario, el grupo, o todos, se puede modificar con el comando `chmod` (change mode). Mostramos algunos ejemplos de este comando asumiendo que estamos parados en el directorio `animales`.

Inicialmente observamos los permisos del archivo `leon` mediante el comando :

```
> ls -l leon
> -rw-rw-r-- 1 herman herman 9 dic 14 08:10 leon
```

Observamos que, por defecto, el archivo nuevo se crea de forma que lo puede leer cualquier usuario, escribir el dueño del archivo y un miembro del grupo solamente. El archivo no es ejectuable (nadie lo puede ejecutar).

```
animales>chmod a-r leon
:animales>ls -l leon
--w--w---- 1 herman herman 9 dic 14 08:10 leon
```

La opción "`a`" significa "todos" (en inglés "all"'). El signo "`-`" significa suprimir privilegios. A todos, incluso el usuario quien creó el archivo, se les prohibe leer este archivo. Observe que el "prompt" acá es `animales`. Más adelante creamos una actividad para que en el "prompt" aparezca el nombre del directorio sobre el cual se está parado.

```
:animales>chmod g+r  leon
:animales>ls -l leon
--w-rw---- 1 herman herman 9 dic 14 08:10 leon
```

Al grupo (`g`) se le agregó (con signo "`+`") la opción leer "`r`".

```
:animales>chmod a+r leon
:animales>ls -l leon
-rw-rw-r-- 1 herman herman 9 dic 14 08:10 leon
:animales>chmod o-r leon
:animales>ls -l leon
-rw-rw---- 1 herman herman 9 dic 14 08:10 leon
```

Se usa el comando y se ejecuta `ls -l` para verificar que el modo fue modificado de forma adecuada. Sobre el archivo `leon` se suspende forma de lectura para todos, luego se agrega forma de lectura para el grupo, se agrega forma de lectura a todos, se suspende forma de lectura a otros. Una forma rápida de modificar los privilegios es usando la forma `XYZ` donde `X, Y, Z` son números en base 8 entre 0 y 7. Por ejemplo, el comando

```
:animales>chmod 751 leon
:animales>ls -l leon
-rwxr-x--x 1 herman herman 9 dic 14 08:10 leon
```

deja al usuario con las tres opciones `rwx` ,pues $7 = (111)_2$, al grupo con las opciones de `r` y `x` ,pues $5 = (101)_2$ y a los otros con la opción `x` solamente, pues $1 = (1)_2$.

Si se usa la opción de privilegio sin indicar el usuario, o grupo u otros este privilegio se expande para todos. Observe el siguiente script.

```
:Animales>ls -l
total 4
-rw-rw-r-- 1 herman herman 34 dic 25 13:03 leon
:Animales>chmod -r leon
:Animales>ls -l
total 4
--w--w---- 1 herman herman 34 dic 25 13:03 leon
:Animales>chmod +r leon
:Animales>ls -l
total 4
-rw-rw-r-- 1 herman herman 34 dic 25 13:03 leon
```

Observe que con la opción "`-r`" quitamos todos los privilegios de leer al usuario, grupo y otros, y con la opción `+r`" agregamos privilegios a todos (usuario, grupo y otros).

Actividad # 5: Explique que quiere decir `total 4` cuando se ejecuta el comando `ls -l`. Encuentre un comando que determine el tamaño de un bloque (block) en Linux.

Actividad # 6: El comando `df` ayuda a determinar el uso de los discos montados en el sistema. Encuentre, mediante el uso de este comando, como saber en que partición está su directorio `home` y qué pocentaje se está usando, también el uso en `blocks`, el tamaño de cada block, y el espacio libre en `blocks`.

Ahora vamos a ver como cambiar el dueño de un archivo. Esto se puede solo si el usuario tiene una contraseña de super-usuario. Observe los siguientes comandos y sus salidas

```
>sudo touch basura
[sudo] password for herman:
>ls -l basura
-rw-r--r-- 1 root root 0 dic 14 08:46 basura
```

```
>sudo chown herman basura
>ls -l basura
-rw-r--r-- 1 herman root 0 dic 14 08:46 basura
```

Se creó un archivo con el nombre `basura` bajo la identidad del super-usuario (con `sudo`). Bajo esta identidad el dueño de este archivo es `root` . Para cambiar de dueño corremos el comando `sudo chown herman basura` bajo privilegios de super-usuario (de otra forma, sin usar `sudo` , no tendríamos la autoridad para modificar de dueño de super-usuario a usuario). El comando `chown herman basura` (`chown` significa "change owner" o cambio de dueño) modifica el archivo basura de forma que el nuevo dueño es `herman` .

Enseguida vamos a "llenar" los directorios creados. En este momento estamos sobre el subdirectorio llamado `animales`. Vamos a buscar un editor para crear un documento con el nombre `leon`. Los editores por exelencia y tradición para Linux son el editor vi [14] y el editor emacs. Estos editores, aunque poderosos, requieren de un tiempo para su aprendizaje, del cual no disponenmos en el momento. Por lo tanto vamos a usar un editor de fácil manejo como es `gedit`. Corremos el comando

```
> gedit leon
```

No vamos a editar el contenido palabra por palabra con el fin de optimizar el tiempo de clase. En **G*o*og*le*** escribimos `leon africano wiki` y copiamos el primer párrafo. Pegamos el contenido en el documento que se acaba de abrir y está vacío en el momento. Se copia con el botón izquierdo del ratón y se pega con el botón derecho (también se puden usar los comandos `Ctr+c` para copiar y `Ctr-v` para pegar. Hacemos "click" en "Save" y cerramos el documento, mediante la selección del menú en la parte superior derecha (después de la palabra "`Save`").

Vamos a ver que el archivo está allí. Para esto los comandos

```
> ls leon
> ls
```

el primero solo corrobora la existencia mientras que el segundo comando muestra los archivos existentes en el directorio donde estamos ubicados.

Para verificar el contenido del archivo, lo podemos abrir con cualquier editor o usar posibles comandos tales como

```
> cat leon
```

[14] https://es.wikipedia.org/wiki/Vi

o

```
> more leon
```

o

```
> less leon
```

Ahora vamos a crear otro documento, que llamaremos `tigre`.

```
> gedit tigre
```

En **Google** escribimos `tigre africano wiki` y seleccionamos la página de `Panthera tigris`. Allí copiamos y pegamos el primer párrafo. Salvamos (`Save`) y cerramos (`Close`, o oprimiendo la `x` en la ventana) el archivo. De nuevo verificamos que el archivo está ahí con el comando

```
> ls
```

Es decir,

```
:animales>ls
leon  nada  tigre
```

Nos vamos a mover ahora al directorio de vegetales. Para moverse de un directorio al padre de este directorio se usa el comando

```
cd ..
```

En este momento debemos estar en el directorio llamado :
`Herramientas_Computacionales` . Si el nombre no aparece en el "prompt" entonces podemos hacer el comando

```
>pwd
```

para que nos muestre donde estamos parados. Acá el resultado, en mi caso.

```
:Herramientas_Computacionales>pwd
/home/herman/Herramientas_Computacionales
```

Actividad # 7 : Ya se tienen los elementos para navegar al directorio padre (y por supuesto abuelo y bisabuelo, etc, mediante el uso repetido del comando "`cd ..`". Ahora el estudiante debe moverse (usando la terminal) al directorio `vegetales` y `minerales` y crear en el directorio `vegetales` tres archivos con el nombre `ajo`, `manzana`, `naranja`. Busquen el contenido en Wikipedia y copien el primer párrafo línea por línea. Luego visitan el directorio `minerales` y crean allí los cuatro minerales `hierro, oro, cobre, carbón`. Usen la misma técnica usada para el caso de animales y vegetales con el objeto de tener todos archivos unificados.

Actividad de navegación por el sistema # 8: En esta sección vamos a mostrar como navegar por el sistema. Para irnos a la raíz simplemente corremos el comando

```
>cd /
```

Para verificar que estamos en la raíz corremos el comando

```
>ls
```

No es recomendable correr el comando

```
>tree
```

en la raíz por su gran extensión[15]. Sin embargo, si lo tenemos que hacer debemos acompañarlo del comando `more` como sigue

```
>tree | more
```

Allí puede ver los directorios que cuelgan de la raíz.

Para moverse al "home" (donde usted tiene su cuenta de usuario) simplemente usas el comando

```
>cd
```

Existen dos tipos de direcciones:

[15]puede tratarlo si quiere, pero podría tener que usar `Ctrl+C` para parar el proceso si se demora mucho

(i) **Absolutas:** Las direcciones absolutas se escriben desde la raiz. Una vez en un directorio, la dirección absoluta se puede averiguar con el comando

```
> pwd
```

por ejemplo si estamos parados en el directorio creado por nosotros `animales`, la salida del comando anterior arroja

```
:animales>pwd
/home/herman/Herramientas_Computacionales/animales
```

De forma que la dirección absoluta es
`/home/herman/Herramientas_Computacionales/animales`. Para moverse a una dirección absoluta basta con usar el comando

```
> cd direccion_absoluta
```

donde `direccion_absoluta` corresponde con la dirección a la cual nos queremos mover y comienza siempre con `/` indicando que se lista desde la raíz (root). Por ejemplo, vamos a navegar por dos lugares. Vamos al directorio "`home`" y de allí al directorio "`animales`". Hacemos `ls` para verificar los comandos. Es decir, tenemos el proceso

```
> cd
> ls
> cd /home/herman/Herramientas_Computacionales/animales
> ls
```

Este proceso debe producir los siguientes resultados (en mi cuenta).

```
:animales>cd
:herman>ls
'BAYLOR COLLEGE OF MEDICINE.pdf'   Dropbox   snap
chapter7.tex    examples.desktop   SU
```

```
debian    Herramientas_Computacionales   Templates
Descargas    leona   Videos
Desktop    Music   Virtual
Documents    Pictures
Downloads    Public
:herman>cd /home/herman/Herramientas_Computacionales/animales
:animales>ls
leon  nada  tigre
```

(ii) **Relativas:** Las direcciones relativas son direcciones que parten del punto donde estamos parados. Una vez parados en un directorio nos podemos mover en dos posibles direcciones. Si el directorio no es la raíz nos podemos mover al padre con el comando

```
> cd ..
```

Si el directorio no es una hoja (pues no podríamos navegar más allá de una hoja), podemos movernos desde el punto donde estamos parados hacia adelante usando el nombre del directorio, arrancando de donde estamos parados. Por ejemplo, si estamos en nuestro directorio "`home`" y queremos movernos al directorio `animales`, creado en esta actividad, podemos simplemente correr el comando

```
> cd Herramientas_Computacionales/animales/
```

Note que **no** se escribe el "slash" `/` al comienzo del comando.

Por ejemplo, acá está el ejercicio en mi cuenta de usuario

```
:Unix>cd
:herman>cd Herramientas_Computacionales/animales/
:animales>pwd
/home/herman/Herramientas_Computacionales/animales
```

donde `Unix` representa el directorio de arranque.

Si queremos regresar al directorio donde estabamos parados la última vez simplemente corremos el comando

```
> cd -
```

Este comando actúa como un interruptor (switch). Es decir, si antes estábamos parados en el directorio `A` y nos movimos al directorio `B` . El comando `cd -` nos retorna a `A` y si corremos el comando una vez más retornamos al directorio `B` .

Actividad # 9 : Navege por los siguientes directorios en el orden indicado: `root, home, vegetales, minerales, animales` . Use direcciones:

(i) absolutas

(ii) relativas

2.1 Concatenación , visualización de contenido de archivos y atributos de archivos.

Se puede usar el comando `cat` para concatenar u observar archivos. Por ejemplo, navegemos hasta el directorio `animales` y ejecutemos el comando

```
> cat leon tigre
```

Este comando simplemente muestra los archivos `leon` y `tigre` uno después del otro. Los archivos se pueden escribir concatenados en un tercer archivo como sigue

```
> cat leon tigre > felinos
```

El archivo `felinos` consiste en la unión de los dos archivos `leon` y `tigre` . Observe las medidas de los tres archivos con el resultado de contar (lineas, palabras y caracteres) en el siguiente comando

```
> ls -l leon tigre felinos
```

El resultado en mi sistema es:

```
:animales>ls -l
total 12
-rw-r--r-- 1 herman herman 1326 feb 26 14:16 felinos
-rw-r--r-- 1 herman herman  931 feb 25 16:39 leon
-rw-r--r-- 1 herman herman    0 feb 25 14:54 nada
-rw-r--r-- 1 herman herman  395 feb 25 16:35 tigre
```

Cuente el número de bytes en cada archivo y verifique que la suma corresponde con lo que se espera. De otra forma más detallada use el comando `wc` (word count), como sigue.

```
> wc leon tigre felinos
```

La respuesta est'a dada en mi ambiente como

```
:animales>wc leon tigre felinos
9  152  931 leon
5   62  395 tigre
14  214 1326 felinos
28  428 2652 total
```

Note que `leon` tiene 9 líneas, `tigre` tiene 5 líneas y `felinos` tienes 9+5=14 líneas, de igual forma se verifica el número de palabras y caracteres. Observe que un caracter coincide en tamaño con un byte (comparando la salida de los comandos `ls -l` y `wc`). El comando también arroja el total de lineas, palabras y caracteres de los tres archivos, que es el doble del tamaño de los `+felinos+ . Podemos observar m\'as atributos del archivo \verb +fe` comandos.

- `> stat felinos`

  Este comando arroja el "status" del archivo felinos. La información es mucho mayor que la del comando `ls -l` . Además de arrojar el tamaño del archivo en bytes y en blocks (un block son 4096 bytes o 4K), produce tiempos de modificación (cambio de contenido), cambio (cambio de dueño o privilegio) y acceso (última vez que se accesó).

- También existe el comando

```
> file felinos
```

que describe el tipo de archivo. Binario, de texto, programa, PDF, comprimido, etc.

Actividad # 10: Por que cuando usted usa `stat felinos` el número de `blocks` no coincide con el tamaño del archivo en `bytes`.

Actividad # 11: Experimente el comando `file` con varios archivos en el sistema. Busque archivos en `Python`, `C`, `PDF`, texto simple (con extensión `txt` o `tex`), etc.

Con el único objeto de crear un archivo de más de una página vamos a correr el comando

```
> cat felinos felinos felinos > felinos3veces
```

El archivo `felinos3veces` contiene ahora 42 líneas que son más de una página para comandos como el comando `more`. Vamos a ensayar los siguientes comandos para visualizar contenido de archivos. Como ya sabemos, el comando `cat` lista el archivo sin parar, hasta el final. Si queremos ver las líneas de más arriba necesitamos usar la barra deslizadora hacia arriba con el ratón. El comando `more` nos permite ver página por página o (usando la barra espaciadora) o línea por línea usando la tecla `return`. Lo podemos tratar con el comando

```
> more felinos3veces
```

Nos salimos del comando con la tecla `q` (quit). Si solo queremos ver las primeras 10 líneas podemos usar el comando `head` . Es decir,

```
> head felinos3veces
```

¿Como sabemos que son 10 líneas? Podemos usar el manual

```
> man head
```

para saber más de este comando, pero sin necesidad de usar el manual podemos correr el comando

```
> head felinos3veces | wc
```

Que conecta el comando `head` con el comando `wc` el cual arroja que estamos mirando las 10 primeras líneas. Si quisiéramos solo 5 lineas decimos

```
> head -n5 felinos3veces
```

(puede verificar con el comando `wc` que son 5 líneas). En vez de 5 líneas podríamos ver 15 líneas. Dejamos esto como ejercicio para el estudiante. De esta forma podemos mirar el número de líneas que queramos pero a diferencia del comando `more` el comando `head` no para. Así que si corrimos más líneas del campo visual de la pantalla debemos usar la barra deslizadora para ver texto oculto. Así como el comando `head` también existe un comando `tail` para ver las últimas líneas de un archivo. Por ejemplo, el comando

```
> tail felinos3veces
```

muestra las últimas líneas del archivo `felinos3veces`. Pueden verificar con el comando `wc` que son 10 líneas las que arroja por defecto. Si, por ejemplo, queremos ver solo las 3 últimas líneas podemos correr el comando

```
> tail -n3 felinos3veces
```

Por último, el comando `less` también permite la visualización de contenidos de archivos y la búsqueda de palabras mediante el uso del slash " \" hacia adelante o el signo de interrogación "`?`" hacia atrás, usando la letra "n" para la búsqueda de la próxima aparición. Por ejemplo, usemos el comando

```
> less  felinos3veces
```

Prueba buscando palabras, hacia atrás, hacia adelante y recorrer varias instancias. El comando "`more`" no es conveniente para búsqueda. Trátalo.

2.1.1 Comparación de archivos

Para acabar esta sección vamos a describir comandos útiles para comparar contenidos de archivos. Copiemos el archivo `felino3veces` al archivo
`felino3vecessim` con el comando

```
> cp  felinos3veces  felinos3vecessim
```

Luego abra el archivo `fielinos3vecessim` (por ejemplo con el editor `gedit`). Cambie la primera `A` de la primera palabra `Asia` de forma que esta `A` es ahora minúscula. Es decir, la primera palabra `asia` viene con minúsculas. Existe del comando `diff` que muestra la diferencia entre dos archivos. La comparación se hace línea por línea.

Este es el resultado de este comando en mi ambiente

```
:animales>diff felinos3veces felinos3vecessim
5c5
< desaparecido del resto de Asia del Sur, Asia Occidental,
África del Norte y la península balcánica en tiempos históricos
---
> desaparecido del resto de asia del Sur, Asia Occidental,
África del Norte y la península balcánica en tiempos históricos
```

Vemos que se muestra la única línea que difiere en el código donde en el primer caso la `A` es mayúscula y en el segundo minúscula . La cadena de caracteres `5c5` indica que la diferencia se encuentra en la línea 5 y que lo que ocurrio fue un cambio (`c`). Si en vez de `c` tuviésemos `d` hubiera ocurrido que borramos algo y si hubiese sido `a` entonces estaríamos agregando algo. El segundo número (en este caso también es 5) indica el número de la línea en el segundo archivo donde se encontraron diferencias.

El comando `diff` solo es útil en archivos de texto. Sin embargo si los archivos son binarios el comando `diff` indica que son diferentes. Por ejemplo,

```
:Unix>diff /bin/cp /bin/mv
Binary files /bin/cp and /bin/mv differ
```

Este comando muestra que los archivos binarios `cp` y `mv` son distintos.

Algunas banderas para mejorar los resultados del comando `diff` son

(i) `i : Ignore`case. Es decir, que la diferencia ignora si la letra es mayúscula o minúscula. En el ejemplo anterior los archivos serían idénticos y el resultado de `diff` no arroja nada (no hay direrencias).

(ii) `-w` Ignore espacios en blanco (white) .

(iii) `-r` Comparación recursiva en todos los directorios y subdirectorios de los archivos a comparar. Este comando es muy útil para detectar fraudes en estudiantes. Si la bandera `-r` está activa probablemente es útil usar la bandera `--ignore-file-name-case` .

(iv) `-B` Ignore líneas en blanco.

(v) `--color` Muestra líneas de distinto color para cada archivo.

Para mayor información sobre este comando el usuario puede escribir `man diff`.

Otro comando útil para comparar archivos es `cmp` (del inglés *compare* que traduce *comparar*). Este comando es más poderoso que `diff` en el sentido en que compara byte por byte y funciona en archivos binarios. Sin embargo `cmp` no trabaja de forma recursiva a diferencia de `diff` . Como ejemplo vemos que

```
:animales>cmp felinos3veces felinos3vecessim
felinos3veces felinos3vecessim differ: byte 501, line 5
```

muestra que los archivos `felins3veces` y `felinos3vecessim` difieren en el byte 501 (línea 5). El comando `cmp` ofrece muchas opciones que se pueden estudiar mediante el comando `man cmp`.

Por último, el comando `kompare` (si no está instalado en su sistema lo puede hacer con el comando `sudo apt-get install kompare`) compara archivos en forma gráfica. La `k` de `kompare` se debe a que el programa se desarrolló para un escritorio del tipo KDE . La Figura 2 muestra el resulado de correr el comando

```
:animales>kompare felinos3veces felinos3vecessim
```

La interfaz gráfica muestra, a la izquierda, el primer archivo y,, a la derecha, el segundo archivo. En el bloque encima del texto aparecen las líneas que son diferentes y se puede navegar sobre ellas. En el ejemplo solo hay una línea.

Actividad # 12: Escoja o cree dos archivos donde halla varias líneas de diferencia. Use el comando `kompare` y navegue usando el interfaz gráfico por las líneas con diferencias.

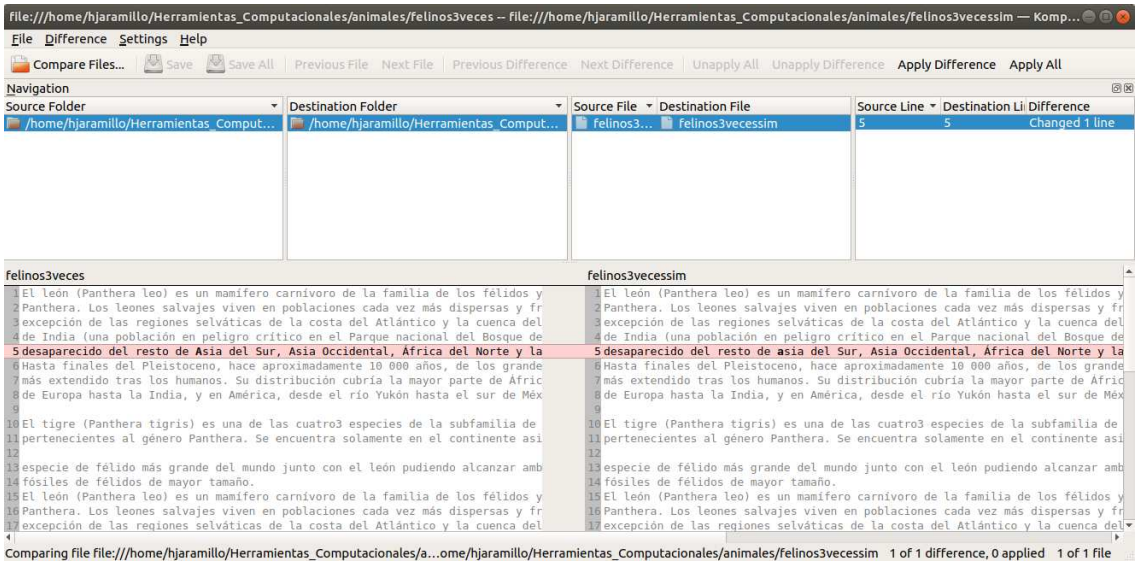

Figure 2: Comparación gráfica de dos archivos con el comando `kompare` .

2.2 Renombrar, remover archivos (y directorios)

Un archivo se puede renombrar con el comando `mv` (move del inglés mover). Por ejemplo, creamos un archivo vacío con el comando `touch` lo observamos con el comando `ls` y luego le cambianos el nombre como sigue. Estando parados en el directorio `animales`: Observe el resultado de correr los cuatro comandos anteriores en su sistema. Estando parados en el directory `animales`:

```
> touch cambieminombre
> ls
> mv cambieminombre cambiado
> ls
```

Observe el resultado de correr los cuatro comandos anteriores en su sistema. Para borrar el archivo simplemente usamos el comando `rm` (remover, del inglés remove). Es decir, usamos el comando

```
> rm cambiado
```

Vamos ahora a mostrar como remover un directorio. Creamos un directorio con el nombre `removerdir` . Luego entramos a ese directorio con el comando `cd removedir`. Allí creamos un archivo de nombre `a` (use el comando `touch`). Nos devolvemos al directorio padre con el comando `cd ..` y tratamos de remover el directorio con el comando `rm removerdir` el sistema arroja error por que el comando `rm` es solo para remover archivos tipo hoja (terminales). Tratamos el comando `rmdir` (remover directorio). Vemos que el sistema no lo permite puesto que el directorio no está vacío. El conjunto de comandos acá indicado y el resultado, en mi sistema, es

```
:animales>mkdir removerdir
:animales>cd removerdir
:removerdir>touch a
:removerdir>cd ..
:animales>rm removerdir/
rm: cannot remove 'removerdir/': Is a directory
:animales>rmdir  removerdir/
rmdir: failed to remove 'removerdir/': Directory not empty
```

Debemos ingresar de nuevo al directorio `removerdir` , borrar el archivo `a` , devolvernos al directorio padre `cd ..` y de allí si correr el comando
`rmdir removedir`.

Si estamos seguros de lo que estamos haciendo podemos usar el comando `rm -rf removedir` que remueve directorios en forma recursiva (`r`) y forzando (`f`) la remoción. Acá el ejemplo en mi sistema:

```
:animales>ls
basura  cambiado  felinos  felinos3veces  leon  nada  removerdir  tigre
:animales>rm -rf removerdir
:animales>ls
basura  cambiado  felinos  felinos3veces  leon  nada  tigre
```

Actividad # 13 : Cree el árbol mostrado en la Figura 3. Mueva el archivo hoja `g` al directorio `b` . Corra el comando `tree` para mostrar el resultado de su trabajo.

3 Herramientas de Búsqueda, filtros, Grep, SED, AWK e Historia

Linux posee algunas herramientas de búsqueda poderosas. En esta sección exploramos los comandos `find` y `grep` .

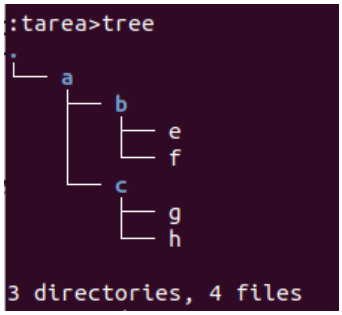

Figure 3: Árbol para actividad en la sección 2.2

3.1 Comando `find`

Nos ubicamos en el directorio home. Asumamos que sabemos que existe un directorio dentro del árbol de archivos que se desprenden de mi directorio home que se llama `animales` . El comando

```
> find
```

simplemente lista todos los archivos debajo del directorio actual (es como el comando `tree` pero no tan organizado).

Para buscar el archivo `animales` usamos el comando `find` de la siguiente manera.

```
:herman>find . -name animales
./Herramientas_Computacionales/animales
find: ‘./.dbus’: Permission denied
```

Al punto `.` después de `find` indica que se busca a partir del directorio local. Vemos que aparecen dos líneas. La primera es
`./Herramientas_Computacionales/animales` indicando que encontró el archivo dentro del directorio `Herramientas_Computacionales`.

Actividad # 14: Para que es la opción “`name`” ? Corra el comando sin la opción “`name`” y observe la diferencia. Comente acerca de esto.

Veamos otro ejemplo.

```
>find /etc/polkit-1 -type f
find: ‘/etc/polkit-1/localauthority’: Permission denied
/etc/polkit-1/localauthority.conf.d/51-ubuntu-admin.conf
/etc/polkit-1/localauthority.conf.d/50-localauthority.conf
```

La opción `f` es para indicar que busque archivos (files) no directorios. Note una línea de error con el mensaje `Permission denied`. Si queremos dirigir las líneas de error a un archivo lo podemos hacer mediante el comando

```
>find /etc/polkit-1 -type f  2> errores
```

Una mirada al archivo `errores` produce

```
>cat errores
find: ‘/etc/polkit-1/localauthority’: Permission denied
```

Asumamos que no estamos interesados en los errores y no los queremos guardar en ningún archivo. Usamos la siguiente sintaxis

```
>find /etc/polkit-1 -type f  2> /dev/null
/etc/polkit-1/localauthority.conf.d/51-ubuntu-admin.conf
/etc/polkit-1/localauthority.conf.d/50-localauthority.conf
```

Si el archivo no existe simplemente no hay salida de ningún tipo.

Actividad # 15 : Pruebe el comando con `animals` en vez de `animales` .

3.2 Comando `grep`

Para estudiar el comando `grep` nos paramos en el directorio `animales` . Vamos primero a ver si el documento `leon` tiene la palabra `Asia`.

```
>grep Asia leon
Distribución histórica (en rojo) y actual (en azul) del león africano en África,
Asia y Europa.
Distribución histórica (en rojo) y actual (en azul) del león africano en África,
Asia y Europa.
Gir y alrededores), habiendo desaparecido del resto de Asia del Sur, Asia Occidental,
```

Vemos que aparecen las líneas donde la palabra `Asia` está localizada. Para más información quisieramos saber el número de las correspondientes líneas. Para esto corremos el comando de forma que encontramos

```
    >grep -n Asia leon
    42:  Distribución histórica (en rojo) y actual (en azul) del león africano
    en África, Asia y Europa.
    43:  Distribución histórica (en rojo) y actual (en azul) del león
    africano en África, Asia y Europa.
  58:  Gir y alrededores), habiendo desaparecido del resto
de Asia del Sur, Asia Occidental,
```

que la palabra `Asia` está localizada en las línea 42, 43 y 58.

Podemos buscar dos cadenas de caracteres distintas que pueden estar en líneas diferentes. Por ejemplo usando la opción "`E`".

```
  >grep -E -n 'sobreviven|sociales' leon
  65:  Si sobreviven a las dificultades de la infancia, las leonas que viven en un hábitat
  71:  y boscosas. Los leones son animales especialmente sociales en comparación con otros
```

El símbolo `|` (pipe) que para los comandos es una conexión lo vamos a usar en este caso como el símbolo lógico `o` .

Si se quiere usar la conjunción "`y`" Por ejemplo

```
  >grep -E -n 'leon.*compone' leon
  72:  félidos. Una manada de leones se compone de hembras que tienen una relación familiar,
```

donde aca la conjunción está representada por el símbolo "`.*`". La opción "`E`" (opción "E"xtendida) permite muchas más posibilidades no consideradas en este curso.

Claro que otra opción es usar varios comandos unidos por la barra vertical "`|`. Por ejemplo, se buscan todas las líneas con la palabra "`leon`" y de estas aqueyas que también tengan la palabra "`compone`". Así,

```
  >grep -n leon leon | grep compone
  72:  félidos. Una manada de leones se compone de hembras que tienen una relación familiar,
```

Para la búsqueda no es necesario que estemos parados en el directorio local donde el archivo `leon` está localizado. Podemos hacer un `grep` recursivo. Por ejemplo, si nos paramos en nuestro directorio de usuario (home) podemos correr el comando `grep` con los argumentos `-nr` de la siguiente forma.

```
  grep -nr Asia
  animales/leon:42:  Distribución histórica ... del león africano en África, Asia y Europa.
  animales/leon:43:  Distribución histórica  ...  del león africano en África, Asia y Europa.
animales/leon:58:  Gir y alrededores), habiendo desaparecido ... Asia Occidental,
animales/felinos:62:con tigres, pero ... parte de Asia. Al
animales/felinos:165:Subfamilia Pantherinae ... Asia central y no en África.26
animales/felinos:173:Trinil (Java). El tigre ...  de Asia a finales del pleistoceno,
animales/felinos:180:Nueve pecies de ...  Asia tiempos recientes, de las cuales tres están
animales/felinos:302:    en el oeste de Asia, abarcando ... Irán, Irak, Pakistán, Rusia y
animales/felinos:326:    Panthera tigris acutidens: ...   en Asia durante el periodo
animales/felinos:401:Actualmente el tigre es animal ...  países de Asia:[cita requerida]
animales/felinos:443:y "salvadores". BBC. Consultado ... de Asia, en peligro de
```

Estando parados en el directorio `Herramientas_Computacionales` . Los puntos suspensivos "`...`" se usan para recortar el tamaño de las líneas.

Enseguida vamos a ver como obtener una respuesta más rápida para la búsqueda de la palabra "`Asia`" . Esto se logra con la combinación de los comandos `find` y `grep` como sigue:

```
:herman>find .   | xargs grep -n Asia
```

La primera parte "`find .  `" recorre todos los directorios y archivos a partir del directorio local ".". Luego se conecta el comando con `xargs grep -n Asia` el cual busca dentro de cada uno de estos archivos la palabra "`Asia`". Si el archivo es un directorio podría obtener el mensaje " `grep:  ./animales:  Is a directory`". Si se quieren evitar mensajes de error o de advertencia se puede agregar un comando mas, por ejemplo

```
:herman>find .   | xargs grep -n Asia | grep -v directory
```

Este evita mostrar todas las líneas con la palabra "`directoy`". La opción "`v`" significa, execpto.

En el ejemplo anterior nosotros debemos saber que el archivo con nombre `leon` es el que contiene la palabra `Asia` . ¿ Qué tal si queremos buscar solo en cierto tipo de archivos, con el find the acelerar la búsqueda? Para esto abordamos la siguiente sección que llamaremos `filtros` .

3.3 Filtros

Los filtros aplicados acá asumen un sistema operacional Ubuntu, dado que buscamos archivos que están localizados en ese ambiente y probablemente no en otros. Vamos a pararnos en el directorio home con el comando: "`cd`". Buscaremos TODOS los archivos que tengan la extension `c` Es decir, archivos de programas en lenguaje `C` . Esto lo podemos hacer con el comando

```
> find . -name ‘‘*.c’’
```

Si vemos que son muchos los archivos (más de una página) podemos hacer una conexión con el comando

```
> find . -name ‘‘*.c’’  | more
```

y esto nos pasa por páginas todos los archivos con extensión `c` Si queremos contar cuantos archivos en el lenguaje `C` hay en el directorio home corremos el comando

```
> find . -name ‘‘*.c’’  | wc
```

De la misma forma podemos contar los archivos en Python en el directorio home con el comando

```
> find . -name ‘‘*.py’’  | wc
```

Actividad # 16: Cuenta todos los archivos de formato PDF en el directorio home.

Vamos a buscar una “aguja en un pajar ”. Pensemos en buscar la palabra `Germaschewski` en el directorio `usr`src que contiene los códigos fuente de programación en `C` . Es decir, asumimos que la palabra existe en algún código de extensión `c` Inicialmente podemos correr el comando

```
> find  | wc
```

que simplemente nos dice cuantos archivos hay bajo el directorio local que en este caso es `usr`src . En mi sistema, en este momento, hay 88110 archivos.

Desde `/usr/src` buscamos todos los archivos con extensión `.c` y conectamos el comando `find` con el comando `grep` . La siguiente lista de comandos dan un ejemplo del poderío de la combinación entre `find` y `grep` .

```
:src>cd
:herman>cd /usr/src
:src>find . -name "*.c" | wc
330     330   18480
:src>find . -name "*.c" | xargs grep -n Germaschewski
./linux-headers-4.15.0-45/scripts/genksyms/genksyms.c:8:   kernel sources by
Rusty Russell/Kai Germaschewski.
./linux-headers-4.15.0-45/scripts/basic/fixdep.c:6: * Author Kai Germaschewski
./linux-headers-4.15.0-45/scripts/basic/fixdep.c:7: * Copyright    2002
by Kai Germaschewski  <kai.germaschewski@gmx.de>
./linux-headers-4.15.0-45/scripts/kallsyms.c:3: * Copyright 2002   by Kai Germaschewski
./linux-headers-4.15.0-45/scripts/mod/modpost.c:3: * Copyright 2003   Kai Germaschewski
./linux-headers-4.15.0-45/scripts/mod/file2alias.c:6: *           2003  Kai Germaschewski
./linux-headers-4.15.0-44/scripts/genksyms/genksyms.c:8:   kernel sources
```

```
by Rusty Russell/Kai Germaschewski.
./linux-headers-4.15.0-44/scripts/basic/fixdep.c:6: * Author    Kai Germaschewski
./linux-headers-4.15.0-44/scripts/basic/fixdep.c:7: * Copyright    2002
by Kai Germaschewski  <kai.germaschewski@gmx.de>
./linux-headers-4.15.0-44/scripts/kallsyms.c:3: * Copyright 2002   by Kai Germaschewski
./linux-headers-4.15.0-44/scripts/mod/modpost.c:3: * Copyright 2003  Kai Germaschewski
./linux-headers-4.15.0-44/scripts/mod/file2alias.c:6: *           2003   Kai Germaschewski
./linux-headers-4.15.0-43-generic/scripts/basic/fixdep.c:6: * Author   Kai Germaschewski
./linux-headers-4.15.0-43-generic/scripts/basic/fixdep.c:7: * Copyright
2002 by Kai Germaschewski  <kai.germaschewski@gmx.de>
./linux-headers-4.15.0-43-generic/scripts/kallsyms.c:3: * Copyright 2002
by Kai Germaschewski
./linux-headers-4.15.0-43-generic/scripts/mod/modpost.c:3: * Copyright 2003
Kai Germaschewski
./linux-headers-4.15.0-43-generic/scripts/mod/file2alias.c:6: *           2003
Kai Germaschewski
./linux-headers-4.15.0-44-generic/scripts/basic/fixdep.c:6: * Author
Kai Germaschewski
./linux-headers-4.15.0-44-generic/scripts/basic/fixdep.c:7: * Copyright
2002 by Kai Germaschewski  <kai.germaschewski@gmx.de>
./linux-headers-4.15.0-44-generic/scripts/kallsyms.c:3: * Copyright 2002
by Kai Germaschewski
./linux-headers-4.15.0-44-generic/scripts/mod/modpost.c:3: * Copyright 2003
Kai Germaschewski
./linux-headers-4.15.0-44-generic/scripts/mod/file2alias.c:6: *           2003
Kai Germaschewski
./linux-headers-4.15.0-45-generic/scripts/basic/fixdep.c:6: * Author
Kai Germaschewski
./linux-headers-4.15.0-45-generic/scripts/basic/fixdep.c:7: * Copyright
2002 by Kai Germaschewski  <kai.germaschewski@gmx.de>
./linux-headers-4.15.0-45-generic/scripts/kallsyms.c:3: * Copyright 2002
by Kai Germaschewski
./linux-headers-4.15.0-45-generic/scripts/mod/modpost.c:3: * Copyright 2003
Kai Germaschewski
./linux-headers-4.15.0-45-generic/scripts/mod/file2alias.c:6: *           2003
Kai Germaschewski
./linux-headers-4.15.0-43/scripts/genksyms/genksyms.c:8:   kernel sources by
Rusty Russell/Kai Germaschewski.
./linux-headers-4.15.0-43/scripts/basic/fixdep.c:6: * Author
Kai Germaschewski
./linux-headers-4.15.0-43/scripts/basic/fixdep.c:7: * Copyright    2002
by Kai Germaschewski  <kai.germaschewski@gmx.de>
./linux-headers-4.15.0-43/scripts/kallsyms.c:3: * Copyright 2002
by Kai Germaschewski
./linux-headers-4.15.0-43/scripts/mod/modpost.c:3: * Copyright 2003
Kai Germaschewski
./linux-headers-4.15.0-43/scripts/mod/file2alias.c:6: *           2003
Kai Germaschewski
:src>find . -name "*.c" | xargs grep -n Germaschewski | wc
33     207    3360
```

Explicamos las líneas de arriba. Inicialmente nos movemos al directorio `/usr/src` . Desde allí buscamos todos los archivos que tengan la extensión `c` y los contamos Son 330 archivos. Buscar uno-por-uno por la palabra *Germaschewski* es muy dificil y lento. Usamos la conexión con `xargs` el cual indica que luego de este se van a correr varios comandos con diferentes entradas (mirar `man xargs` para el significado de este comando.

Básicamente es como un multiplexado de comandos) para buscar mediante el comando `grep` la palabra *Germaschewski* . Vemos que ocurre en muchos archivos. (Germaschewski es desarrollador). Contamos estos archivos con otra conexión a `wc` y encontramos que Germaschewski desarrolló (junto con Rusty Russell como primer autor) 33 códigos. Es decir, el 10 por ciento de los códigos en `C` bajo el directorio `usr`src .

Si 33 códigos son muchos podríamos filtrar más usando el año. Por ejemplo listar solo los que fueron desarrollados en el año 2002, con el comando

```
find . -name "*.c" | xargs grep -n Germaschewski |  grep 2002
```

y contarlos con el comando

```
find . -name "*.c" | xargs grep -n Germaschewski |  wc
```

para un total de 12. En este momento nuestra búsqueda se redujo de 330 archivos a 12.

No solo podemos buscar palabras solas. Podemos buscar frases que colocamos entre comillas. Por ejemplo, el comando.

```
:src>find . -name "*.c" | xargs grep -n "versions of this tool"
./linux-headers-4.15.0-45/scripts/kallsyms.c:742:  * versions of this tool.
./linux-headers-4.15.0-44/scripts/kallsyms.c:742:  * versions of this tool.
./linux-headers-4.15.0-43-generic/scripts/kallsyms.c:742:  * versions of this tool.
./linux-headers-4.15.0-44-generic/scripts/kallsyms.c:742:  * versions of this tool.
./linux-headers-4.15.0-45-generic/scripts/kallsyms.c:742:  * versions of this tool.
./linux-headers-4.15.0-43/scripts/kallsyms.c:742:  * versions of this tool.
```

encuentra todos los archivos con la frase *versions of this tool* .

Actividad # 17: Vaya al directorio `/usr/share` y haga la siguiente actividad. Cuente todos los archivos en `C` . Encuentre todos los códigos cuyo desarrollador es Mark Adler.

El tipo de actividad descrito arriba es muy importante para desarrolladores de código. Muchas veces solo recordamos cosas inexactas, como decir que el archivo es de `C` o de `Java` y alguna palabra clave pero no sabemos donde está. Mediante estas herramientas podemos encontrar cosas en directorios gigantescos en cuestión de segundos.

3.4 Comando SED (Stream Editor)

Es importante notar que tanto el comando `SED` como el `AWK` (que vemos en la siguiente sección) se usa para archivos de TEXTO. El Stream Editor (SED) es un comando que permite editar archivos sin la necesidad de abrirlos y cerrarlos en modo comando. Esto es muy conveniente cuando se quieren crear "shell scrips" para automatizar procesos. Permite buscar cadenas de caracteres y reemplazarlas en modo comando. Se pueden hacer las operaciones de

- Insertar
- Reemplazar
- Remover

La sintaxis es:

```
> sed OPTIONS ... [SCIPT][INPUTFILE...]
```

Creamos un ejemplo de un archivo simple con el fin de probar el comando `sed`. Copiemos y peguemos parte del texto Pera[16] el segundo párrafo en un editor regular (por ejemplo `gedit prueba.txt`).

3.4.1 Insertar lineas en un archivo

Inicialmente mostramos como insertar nuevas líneas al archivo. Corramos los siguientes comandos:

```
>sed '4a  linea insertada' prueba.txt
```

La línea insertada con el texto "`linea insertada`" es ahora la línea 5 y aparece en la pantalla. Si se quiere guardar el resultado en un archivo se usa el comando.

```
>sed '4a  linea insertada' prueba.txt > insertada.txt
>kompare prueba.txt insertada.txt
```

El comando inserta la línea con el texto "`linea insertada`" luego de la línea 4. La letra `a` significa `append` del inglés agregar.

Es importante notar que si agregamos la opción `i`, el archivo de entrada se sobreescribe. Es decir

```
>sed -i '5 a linea insertada' prueba.txt
```

inserta la línea `linea insertada` luego de la línea 5 en el archivo `prueba.txt` y sobreescribe el archivo. Si planea usar la opción `i` es bueno conservar una copia del archivo por seguridad. Note que la opción `5` está adelante de la la opción `a`.

[16] https://es.wikipedia.org/wiki/Pera

Actividad # 18: Que pasa si en vez de `sed -i '5 a linea insertada' prueba.txt` se usa `sed -i 'a 5 linea insertada' prueba.txt`?

Para insertar una línea al final del archivo se usa el signo `$` en vez del número correspondiente a la última línea (aunque esto también funciona pero implica recordar este número). Por favor inserte la línea "`linea al final`" al archivo `prueba.txt`. Use el comando `kompare` para verificar su trabajo.

También se puede insertar una línea debajo de otra línea que cumple un patrón. Veamos un ejemplo:

```
>sed '/inconfundible/ a Linea insertada luego de la linea con inconfundible'
prueba.txt > prueba2.txt
>kommpare prueba.txt prueba2.txt
```

En este caso se busca la primera línea con el patrón "`inconfundible`" y se inserta (después) la línea con el texto "`Linea insertada luego de la linea con infonfundible`".

3.4.2 Modificar lineas en un archivo

Luego mostramos como reemplazar una cadena de caracteres en el texto. La palabra "`producen`" está en el texto en la línea 8. Veamos

```
>grep -n producen *.txt
8:las que menos alergias producen, tiene un alto contenido en agua (más del 80
```

Vamos a reemplazar la palabra "`producen`" por la palabra "`generan`".

Esto se hace con el comando:

```
sed 's/producen/generan/'  prueba.txt
```

Note que esto genera todo el texto de nuevo (en pantalla) con la palabra `producen` reemplazada por `generan`. Explicamos el comando.

- La opción `s` indica sustiución (substitution en inglés).
- Los slash `son separadores.` M'as abajo observaremos que se pueden usar otros separadores como estpacios, barras verticales o puntos.

- La primera palabra está en el archivo original y la segunda es la que entra a reemplazar.

No es necesario el uso del separador “slash“ ”\” . Por ejemplo

```
sed 's producen generan '  prueba.txt
```

produce el mismo resultado. Cada “slash \ ” es reemplazado por un espacio “ ”. También se puede reemplazar el “slash \ ” con una barra vertical “| ”.

```
sed 's|producen|generan|'  prueba.txt
```

o por un punto

```
sed 's.producen.generan.'  prueba.txt
```

El texto de salida, por defecto, aparece en pantalla, pero se puede direccionar a un archivo. Por ejemplo,

```
sed 's/producen/generan/'  prueba.txt > pruebaCambiada.txt
```

Verifique que el archivo `pruebaCambiada.txt` corresponde a la salida del comando `sed` de arriba. Use, por ejemplo, `kompare prueba.txt pruebaCambiada.txt`.

La cadena de caracteres `su` aparece 4 veces en el texto `prueba.txt`. Esto lo verificamos con el comando

```
> grep su prueba.txt | wc
4      27     156
```

Ahora bien observe el siguiente comando y su salida

```
>grep -n su prueba.txt
1:En China son consideradas como un símbolo de longevidad porque, aunque sus
2:flores sugieren fragilidad, crecen en el peral,un árbol caracterizado por su
5:y frutos de su familia |la de las rosáceas|: rosas, fresas, melocotones y
6:cerezas. Además, su inconfundible sabor resiste la destilación para elaborar
```

Observamos que la segunda línea tiene dos ocurrencias de la cadena de caracteres `su`. Si quisíeramos cambiar solo la primera ocurrencia en cada línea de la cadena `su` por su correspondiente mayúscula `SU` corremos el comando

```
>sed 's/su/SU/1'  prueba.txt > SU.txt
```

Revise con `kompare prueba.txt SU.txt` el resultado es el esperado. Solo la primera ocurrencia en cada línea es cambiada y para la línea 2, particularmente, solo la primera ocurrencia. Como las otras líneas tienen solo una ocurrencia el cambio se da en esta ocurrencia. Por defecto esto es lo que pasa en el sistema. Es decir los dos comandos

```
>sed 's/su/SU/1'  prueba.txt > SU.txt
>sed 's/su/SU/'  prueba.txt  > SU2.txt
```

producen el mismo reultado.

Actividad # 19: Corra el comando:

```
>sed 's/su/SU/2'  prueba.txt > SU.txt
```

"K"ompare `prueba.txt` con `SU.txt` y comente sobre el resultado.

Se pueden reemplazar TODAS las ocurrencias de `su` por `SU` con el comando

```
>sed 's/su/SU/g'  prueba.txt > SUtodas.txt
```

Verifique el resultado comparando con `kompare prueba.txt SUtodas.txt`

Se puede hacer la sustitución en solo la línea que se desee. Por ejemplo, La cadena `su` aparece dos veces en la segunda línea. La vamos a reemplazar, sólo en esa línea por `SU` con el comando

```
>sed '2s/su/SU/g'  prueba.txt > second.txt
```

De nuevo, la `g` al final es necesaria si se quiere que el reemplazo sea a lo largo de toda la línea y no de la primera ocurrencia de `su`. Observe el resultado con `kompare prueba.txt second.txt`.

La opción `p` al final imprime cada línea reemplazada dos veces.

Actividad # 20 : Pruebe el comando de reemplazo usando la opción `p` al final. Observe los resultados.

3.4.3 Borrar lineas en un archivo

Podemos remover una línea con el comando simple

```
>sed '2d' prueba.txt > deleted.txt
```

Este comando remueve la segunda línea del archivo `prueba.txt` y escribe el resultado en `deleted.txt`. Use `kompare prueba.txt deleted.txt`. En el caso de que se quiera borrar la última línea se usa el comando

```
>sed '$d' prueba.txt > ultima.txt
```

donde `$` representa la última línea. Verifique con `kompare prueba.txt ultima.txt`.

Actividad # 21: Use la sintaxis:

```
>sed 'n,md' filename.txt
```

para borrar desde la línea n hasta la m.

```
>sed 'n,$' filename.txt
```

para borrar desde la línea n hasta el final. Revise sus resultados.

Si se quiere borrar líneas que tengan algún patrón se puede usar un comando como

```
>sed '/su/d' prueba.txt > patrones.txt
```

Revise los resultados con `kompare prueba.txt patrones.txt`. Se debe observar que todas aquellas líneas con el patrón "`su`" fueron borradas.

3.4.4 Casos especiales

Un caso interesante es cuando el patrón que estamos buscando tiene símbolos especiales. Por ejemplo el slash \ . Retomemos el ejemplo de extraer archivos en el directorio `/etc/polkit-1`

```
>find /etc/polkit-1 -type f  > etcfiles.txt
find: ‘/etc/polkit-1/localauthority’: Permission denied
>cat etcfiles.txt
/etc/polkit-1/localauthority.conf.d/51-ubuntu-admin.conf
/etc/polkit-1/localauthority.conf.d/50-localauthority.conf
```

Que pasa si queremos remover la cadena de caracteres `/etc` mediante SED? El problema es que dos slash uno tras otro puede decir que borren la palabra, pero como borramos los slashes? Por ejemplo

```
>sed 's/etc//' etcfiles.txt
//polkit-1/localauthority.conf.d/51-ubuntu-admin.conf
//polkit-1/localauthority.conf.d/50-localauthority.conf
```

Mientras que el `etc` se borró, no se pueden borrar los slash \ . Entonces como hacemos? Una idea es escapar el slash con otro slash, es decir

```
>sed 's//etc//' etcfiles.txt
sed: -e expression #1, char 8: unknown option to ‘s’
```

Claramente esto no funciona.

En general las expresiones pueden ser complicadas debido a que tienen caracteres especiales. Al término "`regex`" se le conoce como expresiones regulares (en inglés regular expressions). Por ejemplo la página Regex[17]Regex explica acerca de este tema que no solo es importante a nivel del sistema operativo sino de la programación de lenguajes (como `Python`, por ejemplo). Nosotros no vamos a profundizar acerca de esto, solo nos limitamos a resolver este problema específico.

Recuerde que no necesariamente debemos usar slash \ y podemos usar espacios, o barras verticales o puntos. Reemplacemos el primer slash \ que sirve como delimitador y los dos últimos slashes con puntos. Es decir,

[17]https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html

```
>sed 's./etc..' etcfiles.txt
/polkit-1/localauthority.conf.d/51-ubuntu-admin.conf
/polkit-1/localauthority.conf.d/50-localauthority.conf
```

Claramente esto funciona.

3.5 Comando AWK (Alfred Aho, Peter Weinberger, y Brian Kernighan

Más que un comando, AWK (creado for los autores en el título de esta sección) es un lenguaje de programación (scripting language). Como tal no vamos a profundizar mucho en estas notas. Solo presentamos algunos aspectos fundamentales del lenguaje.

AWK está diseñado para trabajar sobre archivos. Algunas operaciones que AWK hace son:

- Revisa línea por línea
- Entiende el archivo como una matriz[18]donde las filas son líneas y las columnas son campos (palabras) en el archivo.
- Compara líneas con patrones
- Ejecuta acciones de acuerdo a patrones especificados.

El lenguaje AWK es útil para transformar archivos de datos y producir reportes formateados. Puede ejecutar el comando

```
>man awk
```

para aprender cosas sobre el comando. También se puede simplemente ejectuar el comando

```
>awk
```

que produce un resumen de las cosas más básicas sobre el comando. Por ejemplo, en mi sistema

[18]En realidad no es una matriz por que cada fila puede tener un número distinto de columnas, pero es como si lo fuera. Quizás el término más adecuado es un agreagado.

```
>awk
Usage: mawk [Options] [Program] [file ...]
```

donde solo capturo la primera línea. Note que en el uso aparece `mawk` en vez de `awk`. La razón es que `mawk` es una nueva versión de `awk`. Igualmente existen `nawk` y `gawk`. El primero conocido como **new** `awk` (una nueva versión) y el segundo como **gnu** `awk`. En este texto solo nos limitamos al `awk` tradicional que debe funcionar en la mayoría de los sistemas Unix.

Descarguemos el archivo de prueba (o copiemos y peguemos) empleados. Lo grabamos con el nombre `empleados.txt`. Sobre este archivo vamos a realizar algunas pruebas. Por defecto la opción "`print`" imprime todas las líneas del texto. Así

```
>awk '{print}' empleados.txt
Jorge jefe de contabilidad 60000
Pedro jefe de negocios 70000
Jose jefe de cartera 300000
Julio auxiliar de contabilidad 300000
Ernesto auxiliar de limpieza 320000
Hernando corredor de bolsa 370000
Maria jefe de personal 40000
Amparo jefe de ventas 40000
```

Similar al comando `sed` se puede buscar líneas con un patrón. Por ejemplo, para buscar e imprimir las líneas con la palabra `jefe` corremos:

```
>awk '/jefe/ {print}' empleados.txt
Jorge jefe de contabilidad 60000
Pedro jefe de negocios 70000
Jose jefe de cartera 300000
Maria jefe de personal 40000
Amparo jefe de ventas 40000
```

Si queremos imprimir solo la primera columna (nombre) y quinta columna (costos) ejecutamos:

```
>awk '{print $1, $5}' empleados.txt
Jorge 60000
```

```
Pedro 70000
Jose 300000
Julio 300000
Ernesto 320000
Hernando 370000
Maria 40000
Amparo 40000
```

La coma "," es un separador. Pruebe el comando sin coma.

Antes de continuar es importante reconocer algunas variables que están predefinidas (built in) para el `AWK`.

- Los símbolos `$1, $2` , etc., indican el campo (columnas de la "matriz" de datos). El sḿbolo `$0` indica TODA la línea.
- `NR`: Número de registro (Number of record). En archivos de texto el registro es la línea. Es decir, durante el proceso NR cambia indicando el número de la línea que, en el momento se está ejecutando.
- `NF`: Número de campo (number of field). Es el número de elementos de cada fila. Se usa `$NF` para indicar la última columna de esa fila.
- `FS`: Separador de campo (Field Separator). Por defecto el separador entre campos (columnas) es el espacio en blanco (uno o mas), pero se pueden usar otros como comas, barras verticales etc.
- `RS`: Separador de líneas (Record Separator). Por defecto los registros están separados por el indicador de nueva línea `\n`. Sin embargo podría ocurrir que los datos estén separados por doble línea. Más abajo mostramos ejemplos donde esto sucede.
- Existen otras variables importantes como `OFS` salida del campo separador (Output Field Separator) y `ORD` salida del registro de separación que no vemos acá. Se dejan al estudiante para que las investiguen como una tarea.

Vamos a ilustrar cada una de estas opciones en los siguientes ejemplos.

3.5.1 Ejemplos de AWK

`$0`:

```
awk '$0' empleados.txt
Jorge jefe de contabilidad 60000
Pedro jefe de negocios 70000
Jose jefe de cartera 300000
Julio auxiliar de contabilidad 300000
Ernesto auxiliar de limpieza 320000
Hernando corredor de bolsa 370000
Maria jefe de personal 40000
Amparo jefe de ventas 40000
```

Esto muestra que el `$0` procesa las líneas imprimiendo el contenido completo.

`NR:`

```
awk 'print NR, $0' empleados.txt
1 Jorge jefe de contabilidad 60000
2 Pedro jefe de negocios 70000
3 Jose jefe de cartera 300000
4 Julio auxiliar de contabilidad 300000
5 Ernesto auxiliar de limpieza 320000
6 Hernando corredor de bolsa 370000
7 Maria jefe de personal 40000
8 Amparo jefe de ventas 40000
```

La opción `NR` muestra la línea que se procesa.

Otro ejemplo se da cuando queremos escoger solo ciertas líneas como se muestra a continuación.

```
awk 'NR==3, NR==6 {print NR, $0}' empleados.txt
3 Jose jefe de cartera 300000
4 Julio auxiliar de contabilidad 300000
5 Ernesto auxiliar de limpieza 320000
6 Hernando corredor de bolsa 370000
```

Observamos que solo se procesan las líneas desde la 3 hasta la 6 (incluidas ambas). Se usa la coma `,` como separador entre los dos extremos del intervalo.

Actividad # 22: Que pasa si se usa un solo = en vez de dos == ? Ensaye y comente sobre el resultado.

`NF:` Con el objeto de probar la variable `NF` creamos un archivo pequeño:

```
>cat small
campo1  campo2

campo1 campo2 campo3
```

al cual llamamos `small` por que solo tiene tres líneas (la segunda en blanco).

```
>awk '{print NF}' small
2
0
3
```

Vemos como se imprime el número de campos en cada línea. Como la segunda línea está en blanco, el número de campos es 0.

La variable `NF` es útil, por ejemplo, para borrar líneas en blanco. Por ejemplo,

```
awk 'NF>0' small
campo1  campo2
campo1 campo2 campo3
```

vemos que con esto se borra la línea 2 que es la única que no tiene campos. Como último ejemplo en la variable `NF` observamos que `$NF` representa el último campo. Ası, el ejemplo

```
>awk '{print $NF}' small
campo2

campo3
```

imprime el último campo de cada línea.

`FS:` Observe los siguientes datos:

```
:herman>cat datos.txt
Jorge | 23 | Pereira
Juan | 22 | Medellin
Julio | 19 | Bucaramanga
Alicia | 39 |  Cali
:herman>awk -F'|' '{print "Nombre: " $1", Edad: " $2 ", Ciudad: "$3}' datos.txt
Nombre: Jorge , Edad:  23 , Ciudad:  Pereira
Nombre: Juan , Edad:  22 , Ciudad:  Medellin
Nombre: Julio , Edad:  19 , Ciudad:  Bucaramanga
Nombre: Alicia , Edad:  39 , Ciudad:   Cali
```

Cree el archivo `datos.txt` y pruébelo con este ejemplo.

Actividad # 23 : Altere el archivo anterior colocando una coma entre `23` y `Pereira` en la primera línea en vez de la barra vertical `|`. Comente sobre el resultado del comando en el script de arriba.

RS: Copie el archivo `datos.txt` en `datos2.tex` y edítelo de forma que obtengamos

```
>cat datos2.txt
Jorge
23
Pereira


Juan
22
Medellin


Julio
19
Bucaramanga
```

```
Alicia
39
Cali
```

Observamos que ahora los registros están separados por doble línea. Escribimos un script que produzca el resultado del ejemplo anterior. Esto es:

```
>awk -v RS='\n\n' '{print "Nombre: "$1 ", Edad: " $2", Ciudad: " $3}' datos2.txt
Nombre: Jorge, Edad: 23, Ciudad: Pereira
Nombre: Juan, Edad: 22, Ciudad: Medellin
Nombre: Julio, Edad: 19, Ciudad: Bucaramanga
Nombre: Alicia, Edad: 39, Ciudad: Cali
```

Acá la opción `v` se usa para asignar a la variable `RS` el valor `nn` antes de su ejecución. En este sentido los registros están separados por doble línea y los campos están separados por una sola línea.

Actividad # 24 : Mostrar ejemplos del uso del `OFS` y `ORS` en el comando `AWK`.

3.5.2 El uso de `if`

Creamos un archivo con el siguiente contenido:

```
>cat datos3.txt
a , 3, 1
b , 3, 2
c , 3, 3
d , 5, 3
d , 0, 2
e , 0, 0
```

Vamos a imprimir aqueyas líneas que tienen números duplicados y omitimos las que no tienen duplicados arrojando el mensaje `no hay duplicados`. El script con los resultados es:

```
awk -F ',' '{if($2==$3){print $1","$2","$3} else {print "no hay duplicados"}}'
datos3.txt
no hay duplicados
no hay duplicados
c , 3, 3
no hay duplicados
no hay duplicados
e , 0, 0
```

Se usa la opción `F` por que el separador son comas ",". El `if` y `else` son obvios para programadores de otros lenguajes.

Además del `if-else`, `AWK` maneja las siguientes operaciones de control:

- `while`,
- `do-while`,
- `do`,
- `for`.

Dejamos como actividad al estudiante que investigue ejemplos del uso de estas operaciones de control.

Actividad # 25: Construya ejemplos con el uso de las operaciones de control mostradas arriba.

3.5.3 Operaciones aritméticas entre columnas

Usamos el mismo archivo de arriba para esta sección.

Con `AWK` es muy fácil intercambiar el orden de las columnas. Por ejemplo

```
awk -F ',' '{print $3 $2}' datos3.txt
1 3
2 3
3 3
3 5
2 0
0 0
```

El script elimina la primera columna e intercambia las columnas 2 y 3.

Si además queremos multiplicar, en la salida anterior, la primera columnna por 7 obtnemos

```
>awk -F ',' '{print 7*$3 $2}' datos3.txt
7 3
14 3
21 3
21 5
14 0
0 0
```

Si queremos sumar las columnas dos y tres encontramos

```
awk -F ',' '{print $2+$3}' datos3.txt
4
5
6
8
2
0
```

Es claro que las operaciones sobre columnas son fáciles con `AWK`. Además de las operaciones aritméticas simples que hemos ilustrado también existen operaciones tales como $\cos(x), \exp(x), \log(x), \sqrt{x}$, etc.

Un ejemplo con el uso de la función $\sqrt{x}$ y $\cos(x)$ es:

```
awk -F ',' '{print $2 "   " sqrt($2) "    " $3 "  " cos($3*3.14/180)}' datos3.txt
3   1,73205     1  0,999848
3   1,73205     2  0,999391
3   1,73205     3  0,998631
5   2,23607     3  0,998631
0   0     2  0,999391
```

El uso de las comillas es solo para generar espacios. Note que la función `cos` asume ángulos en radianes y por eso necesitamos la conversión a radianes. Es decir, asumimos que los

datos se dan en grados. Por esto multiplicamos por $\pi/180$. Usamos la aproximación $\pi \approx 3.14$.

La página tutorialspoint [19] muestra una colección de ejemplos con el uso de estas funciones. Vemos en esta página el uso de códigos multilínea (con el uso de BEGIN) aunque no hay ejemplos que usen archivos. En mi opinión el comando `AWK` es poderoso usándolo en una línea. El uso de más de una línea, pienso, es mejor hacerlo en otros lenguajes como `C` o `Python`.

La combinación de las operaciones de control indicadas arriba, junto con las operaciones matemáticas presentan un recurso poderoso para procesar archivos de datos con comandos de una sola línea. Este tipo de operaciones es de suma importancia en la elaboración de shell scripts. Este tema lo estudiamos más adelante. La próxima sección ilustra la combinación de varios comandos `AWK, sort, sed` para resolver problemas con datos.

3.5.4 Interacción entre distintos comandos

Estudie el siguiente script:

```
>cat datos3.txt
a , 3, 1
b , 3, 2
c , 3, 3
d , 5, 3
d , 0, 2
e , 0, 0
>awk -F, '{print $3, $0}' datos3.txt  | sort | sed 's/e/E/'
0 E , 0, 0
1 a , 3, 1
2 b , 3, 2
2 d , 0, 2
3 c , 3, 3
3 d , 5, 3
```

- Se imprimen todas las líneas con el campo `3` primero y la línea completa después.
- Se ordena (`sort`) de acuerdo a la primera columna (que es la `3` del archivo `datos3.txt`, luego

[19] https://www.tutorialspoint.com/awk/awk_arithmetic_functions.htm

- se le cambia la `e` por `E` a la salida de lo anterior.
- Note que el separador "`,`" no necesariamente tiene que ir entre comillas.

para saber más sobre el comando `sort` use
`>man sort`

3.6 Comando `history`

El comando `history` nos muestra la historia de comandos en modo reverso. El número de comandos que muestra `history` está definido en la configuración del sistema (que normalmente se localiza en el archivo `.bashrc`) el cual detallamos en la siguiente sección. El número de comandos recordados por el comando `history` se almacena en la variable ambiental llamada `HISTSIZE` (en mi caso esta variable contiene el número 1000). Miremos los siguientes comandos.

```
> echo $HISTSIZE
1000
```

Este comando muestra el número de comandos almacenados en la historia de comandos. Podemos hacer uso de la historia para editar y correr un comando anterior mediante alguna de las opciones siguientes. Un comando se puede:

(i) Recobrar usando las flechas. Hacia arriba para regresar a un comando anterior, hacia abajo para retornar a un comando posterior y a la izquierda/derecha para moverse dentro del comando y editarlo. Trate esto en su máquina.

(ii) Escirbir `!num` donde `num` es el número del comando. Esto ejecuta el comando tal y como se ejecuto anteriormente

(iii) Si se escribe `!` inmediatamente seguido de la primera o primeras letras de un comando en la historia, el comando más reciente, que inicie con la misma cadena de caracteres, se ejecuta. Por ejemplo, si en la historia está el comando
`cd /home/herman/Herramientas_Computacionales`
como comando más reciente que comienza con la letra `c` , podemos simplemente escribir luego del prompt `!c` y se ejectua de nuevo este comando.

(iv) Modificar la primera palabra del comando (el comando mismo, o sea el argumento 0) por otro usando la sintaxis (en modo comando)
`vieja \verb` nueva donde `vieja` es la palabra del comando original y `nueva` es

la palabra del comando editado. El comando original es el último comando que comienza con la palabra `vieja` .

Por ejemplo, los últimos comandos en mi ambiente son , luego de ejecutar el comando `history` son

```
2252  cd
2253  cd Herramientas_Computacionales/
2254  cd animales/
2255  ls
2256  cat felinos
2257  cdun
2258  make
```

Si quisiera ejecutar el comando 2253, (usando la sintaxis `!2253`) no lo podría hacer desde donde estoy (directorio `animales`) puesto que el comando `cd Herramientas_Computacionales` es un cambio relativo y no absoluto y debo estar parado en el directorio `home` para tal evento. Podemos tratar y ver que no encuentra el directorio. Es decir,

```
:Unix>!2253
cd Herramientas_Computacionales/
bash: cd: Herramientas_Computacionales/: No such file or directory
```

Debemos entonces pararnos en el home con el comando `cd` . Es decir,

```
:animales>cd
:herman>!2253
cd Herramientas_Computacionales/
:Herramientas_Computacionales>
```

Este tipo de situaciones es útil solo cuando el comando es largo y dispensioso de escribir.

Podemos cambiarnos al directorio animales mediante el comando `!2254`. Una vez en el directorio `animales` podemos ver el archivo `felinos` . Asumamos que ejecutamos el comando

```
>cat felinos
```

y vamos a modificar `cat` por `stat` . Es decir, con el comando

```
> ^cat^stat
```

produciendo como resultado

```
> stat felinos
```

Note que estamos cambiando la cadena de caracteres "`cat`" por "`stat`". Asımismo podríamos haber cambiado simplemente un caracter o dos. Es importante notar que solo se cambian los caracteres al inicio del comando.

Si queremos correr de nuevo el comando 2253 pero sin escribir `!2253` , podemos simplemente escribir `!c` , asumiendo que otro comando que comienza con la letra `c` no es más reciente (por ejemplo el comando `cat`), de otra forma necesitamos escribir sufientes caracteres para definir únicamente el comando. Es decir, si el comando más reciente comienza con `cat` y queremos correr el comando que comienza con `cd` , debemos escribir `!cd` . Esto es algo así como la completación usando la tecla `TAB` pero sin usarla (aunque suene a contradiccíon).

Dejamos como ejercicio al estudiante que trate todas las opciones indicadas arriba.

Actividad 26: Trate todas las formas de correr un comando anterior, o modificarlo usando TODOS los métodos explicados arriba.

4 Variables Ambiente, Alias y Configuración del Sistema con el archivo `.bashrc`

4.1 Variables Ambiente

Las variables ambiente (del inglés Environment Variables) son variables a nivel de la shell. Es decir, que se pueden usar en modo comando. El primer comando que usamos para ver las variables ambientales es

```
> env
```

Es probable que la salida del comando `env` sea grande y se necesite una conexión con `more` (es decir `> env | more`) para ver todas las variables por páginas. También se puede correr el comando

```
> printenv
```

El comando `printenv` solo muestra las variables ambiente, mientras que el comando `env` es más general y permite otras operaciones cuando se usan argumentos, sin embargo, en general los dos comandos se usan para el mismo propósito de capturar información acerca de variables ambiente. El usuario puede consultar el manual con los comandos `>man env` y `>man printenv` para mayor información. Veamos algunas variables ambiente importantes y en el camino mostraremos como listar, definir, editar y usar estas variables.

(i) `HOME` : Almacena el directorio de usuario home. Para ver el contenido de cualquier variable ambiente se usa el comando `echo` . Por ejemplo:

```
:Unix>echo $HOME
/home/herman
```

(ii) `HISTSIZE` : Esta variable se describió en la sección 3.6.

(iii) `PATH` : Esta variable muestra todos los directorios con archivos ejecutables que se pueden ejecutar desde cualquier parte del sistema. Si un archivo es ejecutable y no está en el contenido de `PATH` entonces se debe correr con el comando que incluye la dirección absoluta. Vamos a ver un ejemplo.

Busquemos un programa en C de `hello world` en **G*o*Og*le*** . Copiemos y peguemos el programa con algún editor y salvémoslo en un directorio. Por ejemplo, en nuestro directorio `HOME`. Llamamos a este programa `hello_world.c` . Por ejemplo tenemos

```
:herman>cat hello_world.c
#include <stdio.h>
int main()
{
  // printf() displays the string inside quotation
  printf("Hello, World!\n ");
  return 0;
}
```

Vamos a compilar este archivo (programa) con el comando `gcc` (Gnu C compiler)

```
:tmp>gcc hello_world.c -o hello_world
:tmp>ls -lt hello_world
-rwxr-xr-x 1 herman herman 8312 mar  1 15:48 hello_world
```

También listamos el archivo `hello_world` para verificar que el ejecutable fue generado y que en verdad es ejecutable (marcado con la letra `x` en los privilegios) . Este archivo se corre solo con escribir luego del "prompt"

`hello_world` . Veamos esto

```
:herman>hello_world
Hello, World!
```

Ahora nos movemos al directorio `/tmp` con el comando `cd /tmp` y tratamos de correr el programa `hello_world` .

```
:herman >cd /tmp
:tmp>hello_world
hello_world: command not found
```

Una forma de correr el programa correctamente es usando el camino absoluto. Es decir,

```
:herman>/tmp/hello_world
Hello world
```

Una forma de evitar el esfuerzo de memorizar el camino absoluto es agregando, a la variable ambiental `PATH` el direcctorio donde se aloja el programa.

Listamos el contenido de la variable ambiente `PATH` con el comando `echo` como sigue.

```
:herman>echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
/bin:/usr/games: /usr/local/games:
/snap/bin:.:/home/herman/SU/bin:
/usr/share/sage-6.8-x86_64-Linux
```

Vemos entonces que el directorio `/tmp` no está en la cadena de directorios indicado por la variable `PATH` . Mostramos como agregar el directorio `/tmp` a la variable `PATH` , verificamos que se halla agregado y tratamos ahora de correr el comando `hello_world` desde nuestro directorio `home` .

```
:herman>export PATH=$PATH:/tmp
:herman>echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin:/usr/games:/usr/local/games:
/snap/bin:.:/home/herman/SU/bin:
/usr/share/sage-6.8-x86_64-Linux:/tmp
:herman>hello_world
Hello, World!
```

Observamos que el directorio `/tmp` se le agregó al final de la lista de directorios accesibles para ejecución y que el comando `hello_world` corre bien desde el directorio `HOME` de usuario.

Claramente no queremos desarrollar código en `/tmp` . Esto se hizo solo con el fin de ilustrar el uso de la variable `PATH` . En la práctica desarrollamos código en nuestro directorio `HOME` y allí creamos subdirectorios `/src` para los códigos fuente y `/bin` donde alojamos los programas ejecutables. Este último directorio debe ser incluido en la cadena de directorios de acceso a ejecutables `PATH` con el fin de que el sistema lo encuentre desde cualquier lugar donde el usuario esté parado.

Por supuesto hay muchas más variables ambiente que se pueden ver mediante el comando `env` o el comando `printenv` . Ya mostramos como modificar el contenido de la variable ambiente `PATH` . Vamos a mostrar como definir una variable ambiente. Por ejemplo pensemos en el archivo directorio `Herramientas_Computacionales` . La dirección absoluta de este archivo es
`/home/herman/Herramientas_Computacionales`.
Por alguna razón necesitamos accesar este nombre muchas veces en el sistema (para programar o para simplemente desplazarnos a él). Podemos asignarle una variable ambiente a este archivo. Por ejemplo `HC`. Mostramos como definir esa variable y como verificar que estuvo bien definida.

```
:Unix>export HC=/home/herman/Herramientas_Computacionales
:Unix>echo $HC
```

```
/home/herman/Herramientas_Computacionales
:Unix>env | grep HC
HC=/home/herman/Herramientas_Computacionales
```

Note que para usar el contenido de la variable ambiente necesitamos el signo `$` . En este momento podemos usar la variable `HC` para cualquier efecto. Podemos escribir programas donde se use esa variable o la podemos usar para otros propósitos. Por ejemplo vamos al directorio raíz (root) y desde allí podemos ir al directorio `Herramientas_Computacionales` fácilmente usando la variable `HC` . Veamos

```
:herman>cd /
:>cd $HC
:Herramientas_Computacionales>pwd
/home/herman/Herramientas_Computacionales
```

Si la variable solo se usa localmente no hay necesidad de exportarla con el uso de `export` . Por ejemplo, variables que se usan para ciclos. Por ejemplo la asignación `a=$PWD` almacena el directorio local en la variable `a`. Esta variable no pasa a formar parte del ambiente. Es decir, no sale en el listado arrojado por el comando `env` . Para que salga en el listado arrojado por `env` debemos decir `export a=$PWD`. Subprocesos generados por la shell donde estamos heredan todas las variables del ambiente. Por ejemplo, si corremos el comando `xterm` se abre una terminal nueva. Esta terminal arrastra todas las variables ambiente de la shell madre.

4.2 Alias

Un alias es un nombre que se le da a un comando con el fin de simplificar trabajo. Vamos a definir algunos alias y de esta forma entender mejor el significado de la palabra *alias.* Por ejemplo, si queremos listar los archivos en el directoir local de forma larga y en tiempo reverso ejecutamos el comando `ls -lt`. Es común encontrar directorios con muchos archivos y este comando pasa todos los archivos hasta el final sin tiempo de ver cual fue el más reciente modificado o creado (que en este caso es el primero en la lista). Se hace necesario usar la conexión `|` . Es decir, el comando sería mejor `ls -lt | more` . De forma que listamos los archivos en orden reverso y el primero que vemos es el último que se creó. Por ejemplo use el comando

```
> ls -lt /usr/include/
```

genera un listado largo y no hay forma de ver el último archivo que se grabó.

Actividad # 27 : Cree un archivo (use `touch`) y verifique con el comando anterior que fue el último que se creó.

Ahora bien, no queremos escribir `ls -lt | more` cada vez que necesitemos hacer esto (que puede ser muy a menudo). Generalmente buscamos cosas que hicimos hace poco tiempo, pues son las cosas en las cuales estamos trabajando. También queremos, por seguridad, verificar que creamos o salvamos un archivo. Entonces creamos un *alias* como sigue.

```
:herman>lm
lm: command not found
:herman>alias lm='ls -lt | more'
:herman>lm
total 340
drwxr-xr-x  4 herman herman  16384 mar  1 15:12 Downloads
drwxr-xr-x  2 herman herman   4096 mar  1 13:53 Pictures
-rw-rw-r--  1 herman herman  47087 feb 28 15:33 secante.png
...  etc.
```

El primer comando `lm` verifica que este comando no existe. Ahora vamos a crear un alias de `lm` con la instrucción `alias lm='ls -lt | more'` y luego lo probamos escribiendo `lm` . Vemos que ahora el comando `lm` lista los archivos en orden desde el más reciente hasta el más antiguo. Simplicamos la salida con el texto . etc , pues hay 340 líneas de texto (las cuales se paginan con el uso del comando `more`) . La lista siguiente muestra algunos de los alias que arroja el comando `>alias` en mi sistema:

```
alias ake='make'
alias cdjup='cd /home/herman/Dropbox/Cursos/MetodosNumericos/Jupyter'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias h='history 40'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias lm='ls -lt --color | more'
alias ls='ls --color=auto'
alias luna='~/eclipse/eclipse'
```

```
alias m='more'
alias mak='make'
alias mkae='make'
alias mkdr='make dir'
alias mke='make'
alias mkee='make'
alias mkidr='mkdir'
alias okjlar='okular'
alias okjular='okular'
alias okulr='okular'
alias phones='cat ~/Howto/phones'
alias pusha='a=$PWD'
alias pushb='b=$PWD'
alias pushc='c=$PWD'
alias pushd='d=$PWD'
alias qjup='kill $(pgrep jupyter)'
alias up='cd ..'
alias vi='gvim'
alias vinput='vi input.tex'
```

Note que algunos de los alias en esta lista corrigen mi forma de escribir. Por ejemplo a veces, debido a la velocidad de escritura, por escribir `make` , escribo `mak` o `mkae` o `mke` . Igualmente con el comando `okular` . A esto se le conoce en inglés como *safe typing* . Sin embargo debemos ser cuidadosos pues no queremos definir un alias que sobre-escriba algún comando importante del sistema que pueda afectar el desarrollo normal de los procesos. Por ejemplo `alias ls='rm'` podría ser catastrófico.

Actividad # 28 : Cree un alias para el comando
`cd /home/herman/Herramientas_Computacionales`
con el nombre de `cdhr` . Verifique que el alias quedó bien creado. Observe que acá `herman` soy yo. Usted necesita cambiar `herman` por el nombre de usuario que tiene. Una forma de evitar tener que saber el nombre de usuario es usar el símbolo `~` . Este símbolo significa el directorio home. Por ejemplo, para moverme al directorio
`Herramientas_Computacionales` lo puedo hacer con el comando
`cd ~/Herramientas/Computacionales` .
Redefina el alias definido arriba pero use el caracter `~` para evitar saber el nombre del usuario.

Defina tres alias `pusha, pushb, pushb` como se muestra en la lista arriba. Luego navege a un directorio y corra el alias `pusha` , a otro directorio y corra el alias `pushb` y a un último directorio y corra `pushc` . En este momento se hace fácil navegar por esos directorios, pues dejaron huella. Para irse al directorio marcado por `pusha` simplemente se corre el comando `cd $a` igualmente los comandos `cd $b` y `cd $c` nos transportan al segundo y tercer directorios marcados por `pushb` y `pushc` respectivamente. Se puede copiar o mover archivos desde cualquier parte del sistema a los tres directorios fácilmente . Por ejemplo, el comando `cp archivo $a` copia el archivo con nombre `archivo` al directorio etiquedado con la letra `a` .

4.3 Configuración del Ambiente de Trabajo. Archivo `.bashrc`

Cuando se define una variable ambiente o un alias, este solo existe mientras la sesión de trabajo este activada. Si el usuario cierra la sesión (por ejemplo cierra la terminal) entonces se pierden las variables ambiente y alias definidos. La forma de que, para cada sesión, tanto las variables ambientales como los alias se activen automáticamente es registrando esta información en el archivo `/.bashrc . El archivo \verb` /.bashrc es oculto (con el fin de protergerlo contra borrado accidental). De hecho todos los archivos que empiezan con punto "." se ocultan bajo el comando `ls`. Esto lo explicamos en la sección 2 . El archivo `.bashr` un archivo importante del sistema. El significado (en inglés) original de `bashrc` es Born Again Shell Run Commands .

Actividad # 29: Abra el archivo `~/.bashrc` con el editor de su preferencia. Registre la variable ambiental `HR` y el alias `lm` tal como se definieron arriba dentro de este archivo. Defina un alias nuevo llamado `+cdhr+ el cual cambia de directorio de donde est\'e, al c` . Incluya los alias `pusha`, `pushb` y `pushc` definidos arriba. Salve y cierre el editor. Estas líneas las puede agregar en cualquier parte del documento que no esté contenida en una instrucción del tipo `if` , `else` , `for` , `while` , etc, es decir instrucciones que no queden atrapados en líneas que indican una desviación del flujo normal (hacia adelante) del programa.

El archivo `~/.bashrc` contiene la configuración del sistema para el usuario. Este archivo se ejecuta cada que el usuario hace login o se abra una una terminal nueva. Como en este momento no hicimos login sino que simplemente modificamos el archivo `~/.bashrc` , entonces debemos ejecutarlo manualmente. La forma de ejecutar esos archivos del sistema es mediante el comando

```
> source .bashrc
```

Actividad # 30 : Revise el contenido de la variable ambiente `HC` . Si esta existe entonces elimínela con el comando `unset HC` . Revise que fué eliminada, recuerde el comando `echo` o `env` (junto con `grep`) o `printenv` . Revise que el alias `lm` no estás en su sistema. Esto lo puede hacer, por lo menos, de dos formas:

(i) `> alias lm`

Si obtiene una salida (por ejemplo `alias lm='ls -lt | more'` es por que el alias está definido.

(ii) Otra forma es corriendo el comando `alias` que lista todos los alias. Podría necesitar una conexión con `grep` . Es decir

```
> alias  | grep lm
```

(iii) Por último, otra forma, es simplemente corriendo el comando `lm` y si funciona es por que el alias `lm` está definido en el sistema.

Elimine el alias con el comando `unalias lm`. Verifique que el comando `lm` no existe mediante la ejecución `> lm`. Ejecute el comando
`source ~/.bashrc` , y verfique que tanto la variable ambiental `HC` como el alias `lm` y el alias `cdhc` están definidos en el sistema.

Las variables ambiente del sistema se pueden definir en el archivo `/etc/environment` (para editar este archivo se necesita privilegio de super-usuario). No se necesita la palabra `export` cuando se definen variables en el archivo `etc`environment . Estas se propagan a todos los niveles automaticamente. Cuando se hace login al sistema todas las variables definidas en `etc`environment se pasan al usuario primero, luego se pasan aquellas definidas en el archivo `bashrc` Revise el archivo `/etc/environment`.

Actividad # 31 : Configure su "prompt" para que aparezca su nombre de usuario antes del signo `>`. Es decir, si mi nombre de usuario es `herman` entonces mi prompt se debe ver como

```
>herman
```

5 Suspender Trabajos: top, Ctrl+c, Ctrl+z, jobs, bg, kill

A menudo es necesario saber qué programas están corriendo en el sistema y qué recursos se están utilizando. El comando `top` muestra los procesos que están corriendo en tiempo real. El encabezado de la salida muestra la hora, el número de días que lleva la máquina prendida, la carga en el sistema en los últimos 1, 5 y 15 minutos (1% hasta 100%), el número total de tareas y las que están corriendo, el porcentaje de uso del CPU , la memoria principal usada, libre y disponible así como de `Swap` (la memoria `Swap` usa el disco para apoyarse y tener más disponibilidad en la memoria principal), luego lista los procesos en el orden en que el sistema los está corriendo. En este listado se muestra (toda la memoria es en K) :

(1) `PID` : el número del proceso (process ID)

(2) `USER` : el nombre del usuario dueño del proceso

(3) `PR` : prioridad

(4) `NI` : Valor que el usuario le pone a la prioridad (nice value). Valores negativos tiene mayor prioridad. Ver el comando `nice` . Las prioridades de corrida las debe asignar un super-usuario.

(5) `VIRT` : memoria virtual

(6) `RES` : memoria residente (RES) .La parte de la memoria virtual que no esta "swapped"

(7) `SHR` : Memoria compartida (shared memory) que se comparte con otros procesos

(8) `%CPU`: porcentaje de CPU usado por el proceso

(9) `%MEM`: porcentaje de memoria usado por el proceso

(10) `TIME` : tiempo que lleva el proceso corriendo y

(11) `COMMAND` : el nombre del comando que está corriendo el proceso.

Se puede observar que el comando `top` deja la terminal comprometida. Es decir luego de esto no se pueden ejectuar más comandos, pues el comando `top` está mostrando en tiempo real los procesos que están siendo ejecutados por el sistema. Para poder retornar al modo comando debemos cancelar la tarea. Esta tarea se puede cancelar con la combinación de teclas `Ctrl+c` (se oprimen simultáneamente las teclas `Ctrl` y `c`). Si queremos dejar

esta terminal abierta para supervisar los procesos del sistema podemos abrir una terminal nueva de por lo menos 3 formas:

(i) Oprima la combinación de teclas `Ctrl` +Alt+t . La terminal nueva no conserva las variables ambientales y alias definidos en aquella donde se esta corriendo el comando `top` .

(ii) Enfoque el ratón en la terminal donde se corre el comando `top` . Oprima la combiación de teclas `Ctrl` +Sh+n (`n` significa "nueva").

(iii) Use la aplicación del escritorio.

Note que la combinación `Ctrl+Alt+t` nos coloca en el directorio home (`~`) mientras que `Ctrl` +Sh+n nos coloca en el mismo directorio donde venimos trabajando (justo antes de ejectuar el comando `top`). Podría ocurrir que `Ctrl+Sh+t` abre un `TAB` en la misma ventanda y en el mismo directorio. Esto depende del sistema que se esté usando.

De cualquier forma terminemos fulimnantemente el proceso `top` mediante el uso de la combinacón `Ctrl` +c cuando nos enfocamos en la ventana desde donde se corrió el comando.

Un trabajo se puede suspender fulminantemente (definitivamente) o temporalmente. Miremos como se puede hacer esto. Para esto vamos a escribir un programa en Python que nos muestre los números entre 1 y 10. Comencemos por abrir el editor de su preferencia y escribir estas 4 líneas

```
i=1
while i <= 10 :
  print(i)
# i=i+1
```

guardamos estas líneas en el archivo `delunoal10` .py . Ahora corremos el programa con el comando

```
:herman>python3 delunoal10.py
```

Observamos que el programa imprime 1 y sigue sin parar [20]. Lo podemos parar con `Ctrl` +c como hicimos antes pero vamos a hacerlo esta vez con `Ctrl` +z. La combinación de teclas `Ctrl` +z suspende el trabajo pero solo temporalmente. Esto general el mensaje `[1]+ Stopped python delunoal10.py`. El comando

[20] Al remover el comentario # ,del programa en Python, vemos que éste corre como debe ser imprimiendo los números del 1 al 10

```
> jobs
```

muestra los programas supendidos en la terminal que estamos usando. En este caso muestra el mismo mensaje `[1]+ Stopped python delunoal10.py`. Si hubieran más trabajos suspendidos el comando `jobs` mostraría la lista de los programas suspendidos. Por ejemplo, el comando `ls -R` muestra todos los directorios en forma recursiva (parecido al comando `tree` pero la salida la muestra sin las conexiones entre las ramas, sino como directorios y sus contenidos). Vamos a correr los comandos `cd / ; ls -R` y luego lo suspendemos con la combinación `Ctrl` +z . Observe que se puede usar el punto y coma `para correr varios comandos simult\'aneamente` a estos comandos los llamamos comandos compuestos (en contraste con los comandos simples, donde no hay punto y coma ";"). En este caso nos vamos a la raíz (de donde cuelgan un gran número de archivos) y luego listamos todo el árbol del sistema con el comando `ls -R` .

Luego ejecutamos el comando `jobs` y veremos en la lista

```
:>jobs
[1]-  Stopped                 python delunoal10.py (wd: /tmp)
[2]+  Stopped                 ls  -R
```

Se muestra el número del trabajo, el signo + o - indicando con + que este trabajo está de primero en la cola. La palabra `Stopped` indicando que el trabajo está detenido y al final, el comando asociado con el proceso detenido. Si queremos terminar el trabajo `[2]` definitivamente podemos correr el comando `kill %2`. Observemos

```
:>kill %2
[2]+  Stopped                 ls -R
:>jobs
[1]-  Stopped                 python delunoal10.py (wd: /tmp)
```

Luego del comando `kill %2` obtenemos el mensaje de que el trabajo `[2]` fue parado, y si listamos los trabajos en la pantalla, con el comando `jobs` vemos que solo aparece el trabajo `[1]` correspondiente al comando `python` delunoal10.py .

Reiniciemos el trabajo suspendido con el comando

```
> bg
```

que significa “background”. Este comando trae a la pantalla el programa que está en cola. Es decir, marcado con el signo + . Esta vez, probablemente, no podamos suspenderlo con `Ctrl` +c ni con `Ctrl` +z . Abramos otra terminal (con `Ctrl+alt+t` , desde cualquier parte del sistema o `Ctrl` +Sh+n , desde otra terminal). Para suspender este trabajo podemos usar el comando `ps` . Esta herramienta lista los procesos que están corriendo en el sistema. Podemos inicialmente correr el comando

```
> ps -aux | more
```

para pasar páginas en todos los procesos que está corriendo el sistema (las banderas `-aux` se pueden consultar con el comando `man ps` y dejamos esto al estudiante). Tenemos los siguientes campos en los procesos:

(1) `USER` : nombre del usuario

(2) `PID` : Número del procesos (del inglés Process ID).

(3) `%MEM` : Porcentaje de memorio que usa el proceso.

(4) `VSZ` : Tamaño de la memoria virtual

(5) `RSS` : Tamaño de la memoria residente (separada para el proceso solamente)

(6) `STAT` : Estatus. Algunos estatus son `S` interrumplible. Esperando por señales , `X` muerto, `Z` zombie (muerto pero no eliminado) , I inactivo (en inglés es idle) , `R` corriendo, etc. Cada una de estas banderas puede estar acompañada de los símbolos:

- `<` : Prioridad alta (malo para los otros usuarios)
- `N` : Prioridad baja (bueno “nice” para los otros usuarios)
- `L` : bloqueo (“lock”) . Maneja candado de seguridad.
- `s` : Lider de una sesión
- `l` : Multi-threaded (multi-hilo)
- + : Está en el “foreground” .

(7) `START` : Hora de comienzo

(8) `TIME` :Tiempo que lleva corriendo

(9) `COMMAND` : comando que corre el proceso.

Ejecutemos el comando `ps -aux | grep python`. Pueden aparecer varias líneas. Por ejemplo:

```
:>ps -aux | grep python
root    754  0.0  0.1 170516 17232 ?    Ssl  07:05   0:00 /usr/bin/python3
/usr/bin/networkd-dispatcher --run-startup-triggers
root    907  0.0  0.1 187204 19680 ?    Ssl  07:05   0:00
/usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown
--wait-for-signal
herman  8648  0.2  0.0  25544  6316 pts/2 T    10:43   0:02 python delunoal10.py
herman  8794  0.0  0.0  14432  1048 pts/2 S+   10:57   0:00 grep python
```

Para poder entender el significado de estas líneas necesitamos el encabezado. Podemos usar un grep compuesto como se mostró en la sección 3.2 agregando el campo `USER` (que está en el encabezado) . Para filtrar el número de líneas, y sabiendo ya que la palabra `delunoal10` es única en la lista de líneas vamos a usar el siguiente comando:

```
:tmp>ps -aux | grep delunoal10\|USER  .
USER     PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
herman 11546 96.1  0.0  25544  6320 pts/3    R+   12:23   7:08 python delunoal10.py
herman 11674  0.0  0.0  14432  1004 pts/2    S+   12:30   0:00 grep delunoal10+
\|USER .
```

La segunda línea lista precisamente el comando que estamos corriendo para obtener la primera línea. La primera línea muestra que el programa ha estado corriendo por 7 minutos y 8 segundos. Podemos parar definitivamente el programa con el comando `kill` ("matar"). Para finalizar un proceso definitivamente se usa el número de proceso (PID), que acá es 11546. El comando

```
:tmp>kill -9 11546
```

termina el trabajo y aparece en la pantalla que estaba ocupada, escribiendo una secuencia infinta de unos, "1" la palabra `Killed` . La sintaxis de kill es `kill -signal PID` (signal=señal) . La señal 9 en este caso se llama `SIGKILL` y es más fuerte que otras señales (por defecto la señal usada es `SIGTERM`) . Otras señales podrían no terminar el proceso por proteger al usuario sobre reacciones en cadena que se podrían presentar a raíz de la finalización del proceso. Use el comando `man kill` para más información sobre `kill`.

Por supuesto que una manera de terminar el proceso es cerrando la ventana donde este está corriendo (si esta corriendo en un primer plano –forground–) pero esta forma no es

muy elegante, como tampoco la de apagar o desconectar el computador. Estas formas de apagar el computador podrían tener consecuencias nefastas. Si algún recurso esencial se está usando al momento de apagar el computador se puede perder información importante que impida reiniciar el sistema. Es mejor, para un vehículo (y sus pasajeros), usar los frenos que estrellarlo contra una pared. En ambos casos el vehículo llega a velocidad cero pero los dos métodos pueden tener consecuencias muy distintas.

6 Algunos comandos que pueden ser útiles

6.1 `who` y `date`

Ya habíamos discutido antes sobre el comando `whoami`, el cual anuncia el nombre del usuario en una sesión de `Linux`. Otro comando útil es `date`. Veamos

```
:herman>date jue 28 dic 2023 09:52:12 -05
```

Este comando arroja la fecha y hora local.

6.2 `watch`

Otro comando útil es el comando `watch`. El comando observa un trabajo que está corriendo y reporta el avance sobre el mismo. Es como la barrita de progreso de una tarea en las interfaces gráficas pero muestra datos en vez de la barra.

Por ejemplo, vamos a bajar un archivo de gran tamaño de la internet. Por ejemplo vamos a **G*o*Og*le*** y tratamos de bajar el sistema Ubuntu (en cualquiera de las versiones disponibles). Podemos ver el estado del proceso son `watch` como muestro a continuación.

```
>cd ~/Downloads
>Downloads> watch -n1 ls -lt ubun*
```

En este momento aparece en la pantalla algo como

- Intervalo de tiempo entre reportes
- Estado del comando que se corre.

Por ejemplo el estado arroja el tamaño del archivo que se está descargando de forma dinámica, es decir, a medida que se descarga. La opción `n1` indicca que el comando se refresca cada segundo.

Actividad # 32: Pruebe el comando `watch` con distintos intervalos de reactivación.

6.3 `script`

El comando `script` es muy útil para grabar sesiones de trabajo. En el momento en que se ejecuta el comando `script` se comienza a grabar en el archivo `typescript`, el usuario puede escoger otro nombre para la grabación. Trate el comando `>man script` para más informacón.

Actividad # 33: Pruebe el comando `script`. Luego de comenzar a ejecutar el comando trate varios comandos. Por ejemplo `ls` y luego `more` sobre algunos de los archivos que tiene en el directorio local. Luego suspenda el comando `script` con el comando `exit`. Observe el archivo donde se grabó con el comando `more` o `less`.

El comando `script` es muy útil para compartir actividades que se hicieron con compañeros de trabajo o para reportar tareas como las que se proponen en estas notas.

7 Programación en Bash Básica

La programación en computador consiste en una secuencia de instrucciones que realizan una tarea. Hasta este punto tenemos muchos elementos para esta programación. Cada comando que hemos corrido puede ser una línea en algún programa. A la programación a nivel del sistema (shell) se le llama (en inglés) *shell scripting.* Vamos al directorio `Herramientas_Computacionales` (lo podemos hacer con el alias `cdhc`). Creamos un directorio llamado `Bash` y nos movemos a el con el comando compuesto `mkdir Bash; cd Bash`. Vamos a marcar este directorio con el alias `pusha` , de forma que desde donde estemos nos podamos regresar con el ocmando `cd $a` . Podríamos decirle al computador que nos escriba la frase `Hola Mundo` corriendo el comando

```
:Herramientas_Computacionales>echo "Hola Mundo"
Hola Mundo
```

El comando `echo` simplemente es el "print" de la programación en bash, o en modo comando.

Comencemos con el programa más simple conocido (en inglés) como `hello_world` .sh . Abramos un archivo con un editor y escribamos las siguientes líneas

```
#!/bin/bash
```

```
echo ‘‘Hola Mundo’’
```

Guardamos este programa con el nombre de `hola_mundo` .sh . Si tratamos de correr el programa (escribiendo `hola_mundo` .sh) encontramos:

```
:Bash>hola_mundo.sh
bash: ./hola_mundo.sh: Permission denied
```

Si listamos el archivo como

```
:Bash>ls -l hola_mundo.sh
-rw-r--r-- 1 herman herman 31 Mar  2 17:37 hola_mundo.sh
```

vemos que ni el usuario, ni el grupo, ni los demás tienen la bandera `x` de privilegio de ejecución activada. Debemos entonces modificar el modo de ejecución mediante el comando `chmod u+x`. Luego lo podemos ejecutar. Encontramos

```
:Bash>chmod u+x hola_mundo.sh
:Bash>hola_mundo.sh
Hola Mundo
```

Podemos colocar cualquier número de comandos en un archivo de esta forma y simplemente correr todos los comandos con escribir el nombre del archivo (previamente habilitado para ejecutar). Por ejemplo agregemos dos líneas más al codigo `hola_mundo` .sh . La linea `sleep 4` y la línea `echo Buenos Dias` (olvidamos las tildes a nivel de programación). Luego corremos el programa de nuevo.

```
:Bash>cat hola_mundo.sh
#!/bin/bash

echo "Hola Mundo."
date
sleep 4
date
echo "Buenos Dias"
:Bash>hola_mundo.sh
```

```
Hola Mundo.
jue 28 dic 2023 15:45:35 -05
jue 28 dic 2023 15:45:40 -05
Buenos Dias
```

Vemos que se ejecuta la línea que imprime `Hola Mundo` y luego de 4 segundos la línea con el texto `Buenos Dias`. Ejecutamos los comandos `date` con el fin de verificar la pausa de 4 segundos. Tenga en cuenta que el tiempo que produce `date` es el "wall time", es decir, el tiempo que lee un reloj de muro y dado que Linux tiene muchos programas corriendo a la vez el tiempo en el muro puede ser más lento que en del comando `pause` debido a estos procesos. Esto se ve trivial pero piense que debe hacer 10 tareas y que cada tarea se puede demorar entre 8 y 10 horas. No vamos a querer estar sentados entre 80 y 100 horas para hacer esas tareas o estar visitando el computador cada 8 o 10 horas (o más) para correr la siguiente tarea. Todas las tareas se pueden escribir en un solo archivo y dejar que el computador haga el trabajo corriendo solo un comando. Esta es la magia detrás del scrip shell programming.

La programación es mucho más que colocar una línea tras otra y dejar que el programa se ejecute en el orden en que se colocó cada línea. Los elementos más importantes de la programación son la **variables**, **instrucciones condicionales** y los **ciclos** o **bucles**. Estos tres elementos son los que explicamos acá.

7.1 Variables

Para el siguiente ejemplo usamos los comandos `whoami` y `date` que producen el nombre del usuario y el nombre del día. Copie el programa a continuación con un editor y guarde el programa con el nombre `saludo` .sh .

Listado 1: saludo.sh

```
#!/bin/bash

saludo="Buenos Dias"
usuario=$(whoami)
dia=$(date +%A)

  echo "$saludo $usuario! Hoy es $dia".
  echo "estas corriendo la version $BASH_VERSION de bash"
```

No olvide convertir el programa a ejecutable. Por ejemplo con el comando `chmod +x saludo.sh`. En este pequeño programa definimos tres variables: `saludo` , `usuario` y `dia`

. La primera variable es simplemente un texto que nosotros asignamos a `saludo` . La segunda variable toma el nombre del usuario del comando `whoami` y la tercera variable toma su valor del comando `+date+ . El argumento \verb +%A extrae el d\'ia de la s` el manual `man date` para ver esta y más opciones del comando.) Finalmente note que estamos usando la variable ambiente `$BASH_VERSION` . En mi sistema el día se produce en inglés como se muestra enseguida.

```
:Bash>saludo.sh
Buenos Dias herman! Hoy es Saturday.
estas corriendo la version 4.4.19(1)-release de bash
```

Advertencia! No deje espacios entre el signo igual "= y la parte izquierda y derecha del mismo.

7.2 Condicionales

Una de las principales directivas de la programación está en los condicionales. La idea de los condicionales es la de desviar el proceso de acuerdo a la validez o no de alguna condición. El próximo código ilustra un caso simple del uso del condicional `if` en bash.

Listado 2: condicional.sh

```
#!/bin/bash

dir_a=/usr/local
dir_b=/etc

num_a=$(ls $dir_a | wc -l)
num_b=$(ls $dir_b | wc -l)

echo "la carpeta $dir_a tiene $num_a archivos"
echo "la carpeta $dir_b tiene $num_b archivos"

if [ $num_a -lt $num_b ]; then
echo "la carpeta $dir_a tiene menos archivos que la carpeta $dir_b"
else
echo "la carpeta $dir_a tiene mas archivos que la carpeta $dir_b"
fi
```

La salida de este programa se muestra a continuación

```
:Bash>condicional.sh
la carpeta /usr/local tiene 10 archivos
```

```
la carpeta /etc tiene 259 archivos
la carpeta /usr/local tiene menos archivos que la carpeta /etc
```

Note que usamos indentación bajo `if` y bajo `else`. `Bash` no es tan exigente como `Python`. El programa corre igual sin indentación, pero recomiendo que se haga la indentación por claridad. Podemos usar argumentos en el modo comando para proveer flexibilidad a este programa. Copiemos este programa al programa `condicional_argumentos` .sh con el comando `cp condicional.sh condicional_argumentos.sh` y lo editamos cambiando usrlocal por `1  y \verb /etc  por \verb` 2. Es decir el nuevo programa `condicional_argumentos` .sh se lista como

Listado 3: condicional_argumentos.sh

```
#!/bin/bash

dir_a=$1
dir_b=$2

num_a=$(ls $dir_a | wc -l)
num_b=$(ls $dir_b | wc -l)

echo "la carpeta $dir_a tiene $num_a archivos"
echo "la carpeta $dir_b tiene $num_b archivos"

if [ $num_a -lt $num_b ]; then
echo "la carpeta $dir_a tiene menos archivos que la carpeta $dir_b"
else
echo "la carpeta $dir_a tiene mas archivos que la carpeta $dir_b"
fi
```

Lo corremos como sigue

```
:Bash>condicional_argumentos.sh /etc /usr/local
la carpeta /etc tiene 259 archivos
la carpeta /usr/local tiene 10 archivos
la carpeta /etc tiene mas archivos que la carpeta /usr/local
```

El primer argumento `/etc` se asocia a la variable `$1` y el segundo argumento `/usr/local` se asocia a la variable `$2` . Observamos que invertimos el orden de las variables de forma que ahora se ejecuta la segunda rama del `if` y en vez de la palabra `menos` tenemos la palabra `mas` . En general los tipos tipos de condicionales en BASH son:

(i) Condicionales (simples) que se desvían y regresan al tronco principal . La sintaxis es `if/fi` . El código se inserta en el cuerpo que denotamos con el símbolo `/` .

(ii) Condicionales binarios. Se bifurcan en dos ramas. la sintaxis se resume en `if/then/else/fi` . Una rama se programa en el cuerpo de `then` y la otra en el cuerpo de `else` .

(iii) Condicionales multi-rama. Estos se codifican con la palabra `case` . La palabra `case` significa caso. Se listan los casos que pueden ir desde 1 hasta n donde n es un entero mayor que 1.

No vamos a produndizar sobre el uso de condicionales. Al estudiante que quiera ir más allá en el uso de condicionales le recomendamos visitar el sitio Bash-Beginners [21]

7.3 Ciclos

Los ciclos más comunes en los lenguajes de programación son el ciclo `for` y el ciclo `while` . Vamos a mostrar ejemplos breves de estos dos tipos.

7.3.1 Ciclo `for`

Antes de de mostrar un programa en bash que use el ciclo `for` vamos a ver un ejemplo del uso del ciclo for en una sola línea.

```
:Unix>for i in ‘seq 1 5‘ ; do echo $i; done
1
2
3
4
5
```

Es importante notar que la comilla es de apertura en ambos lados. Es decir ‘, no ’, si se puede ver la diferencia. El comando para `for` consiste en tres partes (separadas con punto y coma) la cabeza `for` , el cuerpo que comienza con `do` y se cierra con la palabra `done` .

Asumamos que necesitamos crear 100 archivos con el texto `1` en el primer archivo, `2` en el segundo , y así hasta que el archivo 100 debe tener el texto `100`. Ejecutemos el alias `pusha` para marcar este directorio el cual podemos necesitar más adelante. Movámonos

[21] http://tldp.org/LDP/Bash-Beginners-Guide/html/chap_07.html

al directorio `tmp` y allí creamos el directorio `basura` y luego nos movemos al directorio `Basura` . Estos pasos los podemos hacer en una sola línea

```
:Bash>pusha; cd /tmp; mkdir Basura; cd Basura
:Basura>
```

El prompt `Basura` ¿ muestra que estamos parados en el directorio `Basura` . Ahora corremos la linea

```
:Basura>for i in ‘seq 1 100‘ ; do touch  numero$i ; echo $i>numero$i; done
```

El comando `ls` verifica que los archivos fueron generados.

```
:Basura>ls
numero1    numero2   numero30  numero41  numero52  numero63  numero74  numero85  numero96
numero10   numero20  numero31  numero42  numero53  numero64  numero75  numero86  numero97
numero100  numero21  numero32  numero43  numero54  numero65  numero76  numero87  numero98
numero11   numero22  numero33  numero44  numero55  numero66  numero77  numero88  numero99
numero12   numero23  numero34  numero45  numero56  numero67  numero78  numero89
numero13   numero24  numero35  numero46  numero57  numero68  numero79  numero9
numero14   numero25  numero36  numero47  numero58  numero69  numero8   numero90
numero15   numero26  numero37  numero48  numero59  numero7   numero80  numero91
numero16   numero27  numero38  numero49  numero6   numero70  numero81  numero92
numero17   numero28  numero39  numero5   numero60  numero71  numero82  numero93
numero18   numero29  numero4   numero50  numero61  numero72  numero83  numero94
numero19   numero3   numero40  numero51  numero62  numero73  numero84  numero95
```

Para verificar el contenido elegimos un número al azar. Asumamos que escogemos el número 57. Observemos el resultado de ver el archivo `numero57` .

```
:Basura>cat numero51
51
```

Se confirma que el archivo `numero51` contiene el número 51. El programa listado a continuación muestra la conversión del comando una sola línea al programa `muchosarchivos` .sh . Note que en vez de `100` pusimos el argumento `$1` para tener la flexibilidad de escoger el número de archivos que queremos general.

Listado 4: muchosarchivos.sh

```
#! /bin/bash

for i in `seq 1 $1`
do
touch  numero$i
echo $i>numero$i
done
```

Recuerde modificar el privilegio de ejecución con el comando

```
chmod u+x muchosarchivos.sh
```

Antes de ejecutar este archivo estemos seguros que estamos en el directorio tmpBasura , donde tenemos una copia de él . Guarde una copia (para sus registros) en el directorio Bash creado arriba con el comando `cp muchosarchivos.sh $a` .

Removamos todo lo que está en el directorio tmpBasura con el comando `rm *` . Ahora corremos el programa

```
:Basura>muchosarchivos.sh 1000
```

Escoja un número al azar entre 1 y 1000 y verifique el archivo correspondiente a ese número tiene como contenido el número.

7.3.2 Ciclo `while`

Para explicar el uso de la instrucción `while` hagamos una copia del archivo `muchosarchivos` .sh en el archivo `muchoswhile` .sh con el comando

```
:Basura>cp muchosarchivos.sh muchoswhile.sh
```

Editamos el archivo cambiándolo para que quede como

Listado 5: muchoswhile.sh

```
#! /bin/bash

i=1
while [ $i -le  $1 ]
do
```

```
touch  numero$i
echo $i>numero$i
i=$(($i+1))
done
```

Vemos que el ciclo `while` necesita de una inicialización de la variable `i` y tener en cuenta que la variable se debe incrementar manualmente. En el caso del ciclo `for` la variable se incrementa automáticamente y esta puede ser una fuente de error en el ciclo `while` que produce ciclos infinitos como el que estudiamos en el programa en Python en la sección 5.

Dejamos al estudiante como ejercicio probar este programa con varios valores del argumento `$1` .

Los programas mostrados acá son programas jugete pero permiten apreciar el poder de la programación en BASH . En la práctica los programas pueden tener cientos de líneas y corren trabajos de gran envergadura en donde se combinan los condicionales, ciclos y líneas de comandos regulares.

Para terminar mostramos a continuación una lista de comandos útiles para concoer el sistema en el cual estamos trabajando.

8 Elimine el usuario

En este punto del curso debemos borrar el usuario que se creó para dejar el sistema tal como se encontró al comienzo del curso. Antes de borrar haga un respaldo de todos los archivos que le puedan ser útiles más adelante. Para borrar el usuario corra los siguientes comandos:

```
>sudo deluser usuario
```

Donde `usuario` es el nombre del usuario que usó en la clase. Note que el usuario todavía aparece en el `home directory`. Lo debe borrar con

```
>sudo rm -r /home/usuario
```