Tutorial en Linux

Herman Jaramillo

March 3, 2019

1 Introducción

1.1 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]

1.2 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, 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 Red Hat, CentOS, Fedora, OpenSUSE y Ubuntu entre otros. Este documento se preparó con el sistema operacional

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

de Ubuntu. La página en este enlace [4] muestra con detalle una lista de distribuciones de Linux y componentes dentro de las distribuciones.

1.3 Componentes

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

1.3.1 Kernel

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

1.3.2 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 csh, ksh, bsh, bash, csh, tcsh, etc. Aunque shell es una palabra en inglés que traduce "concha" nos seguimos refiriendo con su nombre en inglés el por ser altamente usado en la comunidad de Linux. En este tutorial solo nos concentraremos en la shell tipo "bash" (Born again shell). 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. Si hay archivos que comienzan con el mismo nombre `TAB` lista todas las opciones para que el usuario escoja la deseada.

1.3.3 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 [5] y el GNOME [6] El escritorio KDE se desarrolla sobre las herramientas para desarrollo de interfaces

[4] https://en.wikipedia.org/wiki/List_of_Linux_distributions
[5] https://en.wikipedia.org/wiki/KDE
[6] https://en.wikipedia.org/wiki/GNOME

gráficas (GUI=Graphical User Interface) Qt [7] el cual es de uso comercial, mientras que Gnome se desarrolla sobre GTK [8] que es de uso libre.

La página Best Desktop Environments for Linux [9] muestra un conjunto de escritorios usados en Linux. El ambiente

2 Estructura de Archivos en Linux

La estructura en que los archivos se crea 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 de grandes que solo duran mientras se tiene una sesión (prendida y apagada del equipo) de trabajo.

(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.

(v) El directorio `etc` ofrece, entre otros, los nombres del 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.

[7] https://www.qt.io/
[8] https://www.gtk.org/
[9] https://itsfoss.com/best-linux-desktop-environments/

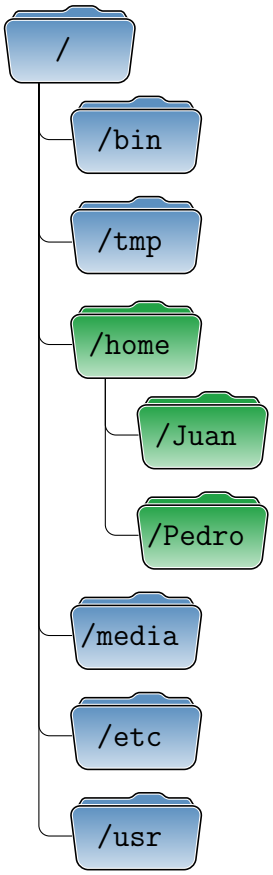

Figure 1: Estructura de los Archivos en Linux en forma de árbol

Existen otros directorios tales como `dev`, `mnt` los cuales no estudiamos en este curso. Sugerimos visitar la página en este enlace [10] para una información mucho más completa acerca de la estructura de los archivos en Linux.

Cuando ingresamos al sistema, automaticamente 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` .

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

2.1 Actividad de creación de archivos

Los archivos tiene dos formas fundamentales. Un archivo es un directorio (es decir contiene otros archivos) o es una rama terminal del árbol (es decir, es 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 en la matriz de puntos (si no se halla en la barra). 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-Sh-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 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. 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` . Si se quieren ver ejemplos se puede navegar oprimiento la tecla "/" y escribiendo la palabra que se quiere buscar. 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 *superuse 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 [11] y Gnome Software [12]. 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

```
> 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

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

```
> 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: 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 vacíoa 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 guion `-` 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 carateres 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.

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`,

```
animales>chmod a-r leon
:animales>ls -l leon
--w------- 1 hjaramillo hjaramillo 931 feb 25 16:39 leon
:animales>chmod g+r  leon
:animales>ls -l leon
--w-r----- 1 hjaramillo hjaramillo 931 feb 25 16:39 leon
:animales>chmod a+r leon  leon
:animales>ls -l leon
-rw-r--r-- 1 hjaramillo hjaramillo 931 feb 25 16:39 leon
:animales>chmod o-r leon
:animales>ls -l leon
-rw-r----- 1 hjaramillo hjaramillo 931 feb 25 16:39 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 el binario `XYZ` donde `X, Y, Z` son números en base 2 entre 0 y 7. Por ejemplo el comando

```
:animales>chmod 751 leon
:animales>ls -l leon
-rwxr-x--x 1 hjaramillo hjaramillo 931 feb 25 16:39 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 $1 = (1)_2$. Ahora vamos a ver como cambiar el dueño de un archivo. Observe los siguientes comandos y sus salidas

```
:animales>sudo touch basura
[sudo] password for hjaramillo:
:animales>ls -l basura
-rw-r--r-- 1 root root 0 feb 28 07:34 basura
:animales>sudo chown hjaramillo basura
:animales>ls -l basura
total 20
-rw-r--r-- 1 hjaramillo root          0 feb 28 07:34 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 hjaramillo 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 hjaramillo basura` (`chown` significa "change owner" o cambio de dueño) modifica el archivo basura de forma que el nuevo dueño es `hjaramillo`.

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 [13] y el editor emacs Estos editores 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 google escribimos `leon africano wiki` y copiamos el primer párrafo, línea por línea. 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 central. 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

[13] https://www.ccsf.edu/Pub/Fac/vi.html

```
> cat leon
```

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, línea por línea. 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/hjaramillo/Herramientas_Computacionales
```

Actividad 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.

2.2 Actividad de navegación por el sistema

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 [14] . 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:

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

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

```
> pwd
```

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

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

De forma que la dirección absoluta es
`/home/hjaramillo/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 vegetales. Hacemos `ls` para verificar los comandos. Es decir, tenemos el proceso

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

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

```
:animales>cd
:hjaramillo>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
:hjaramillo>cd /home/hjaramillo/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
:hjaramillo>cd Herramientas_Computacionales/animales/
:animales>pwd
/home/hjaramillo/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 Navege por los siguientes directorios en el orden indicado: `root, home, vegetales, minerales, animales` . Use direcciones:

(i) absolutas

(ii) relativas

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

Se puede usar el comando `cat` para concatenar 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 hjaramillo hjaramillo 1326 feb 26 14:16 felinos
-rw-r--r-- 1 hjaramillo hjaramillo  931 feb 25 16:39 leon
-rw-r--r-- 1 hjaramillo hjaramillo    0 feb 25 14:54 nada
-rw-r--r-- 1 hjaramillo hjaramillo  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 más atributos del archivo `felinos` con los siguientes comandos.

```
> stat felinos
```

Este comando arroja el "status" del archivo felinos. La informaci'on 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: Experimente el comando `file`con varios archivos en el sistema. Busque archivos en Phython, C, PDD, texto simple, 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 quisieramos 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.

2.3.1 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 commando `diff` que muestra la diferencia entre dos archivos. La comparación se hace línea por línea.

Este es el reultado 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 nńumero (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. S 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
```

2.4 Renombrar, remover archivos (y directorios)

Un archivo se puede renombrar con el comando `mv` (move del inglés mover). Por ejemplo, creamos un archivo vacíco con el comando `touch` lo observamos con el comando `ls` y luego le cambianos el nombre como sigue. Estando parados en el directorio `animales` :

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

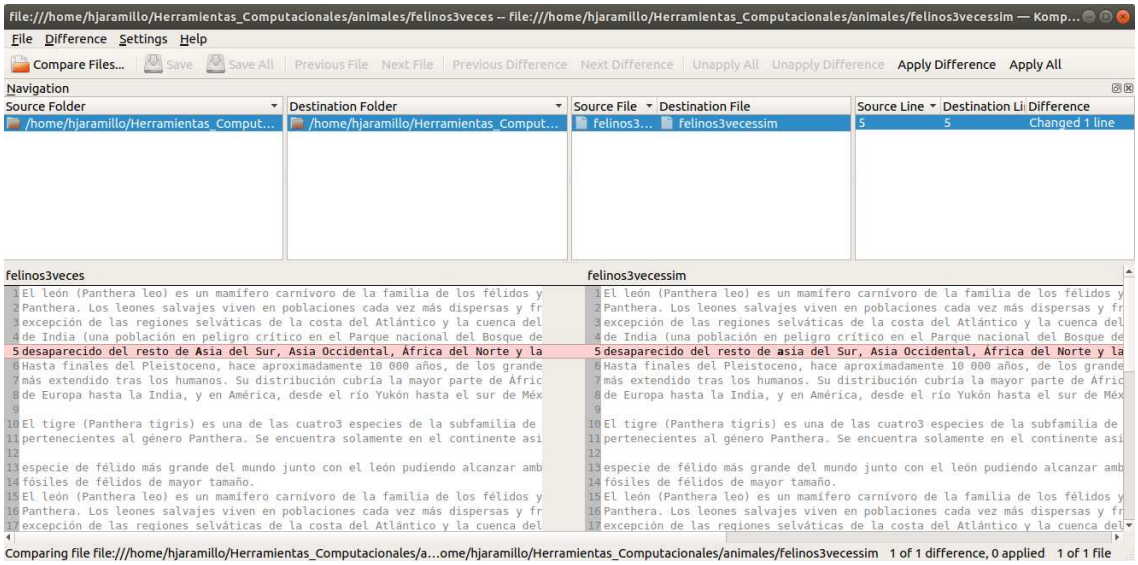

Figure 2: Comparación gráfica de dos archivos con el comando `kompare` .

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: 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.

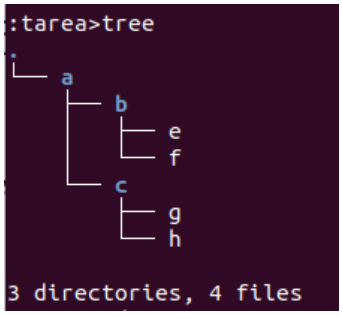

Figure 3: Árbol para actividad en la sección 2.4

3 Herramientas de Búsqueda, filtros e Historia

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

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.

```
:hjaramillo>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` . La segunda línea tiene la información `find: ‘./.dbus’: Permission denied`. Esto significa, aunque la búsquerda fue exitosa, se produjo 1 línea donde el usuario no tiene permiso para buscar (por falta de privilegios). Si queremos dirigir las líneas de error a un archivo lo podemos hacer mediante el comando

```
:hjaramillo>find . -name animales 2> errores
./Herramientas_Computacionales/animales
```

Una mirada al archivo `errores` produce

```
:hjaramillo>cat errores
find: ‘./.dbus’: Permission denied
```

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

```
:hjaramillo>find . -name animales 2> /dev/null
./Herramientas_Computacionales/animales
```

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

Actividad: 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* .

```
:animales>grep Asia leon
desaparecido del resto de Asia del Sur, Asia Occidental,
África del Norte y la península balcánica en tiempos históricos
```

Vemos que aparece la línea donde la palabra *Asia* está localizada. Para más información quisieramos saber la línea donde esta palabra está localizada. Para esto corremos el comando de forma que encontramos

```
5:desaparecido del resto de Asia del Sur, Asia Occidental,
África del Norte y la península balcánica en tiempos históricos
```

que la palabra *Asia* está localizada en la línea 5.

Podemos buscar dos cadenas de caracteres distintas que pueden estar en líneas diferentes. Por ejemplo

```
hjaramillo> grep 'Asia\|humanos' leon
Bosque de Gir y alrededores), habiendo desaparecido del resto de Asia del
Asia Occidental, África del Norte y la península balcánica en tiempos histór
mamíferos terrestres, el león era el más extendido tras los humanos.
```

El símbolo `|` (pipe) que para los comandos es una conexión lo vamos a usar en este caso como el símbolo lógico `o`. Pero para que no se confunda el sistema con la conexión le colocamos el backslash `\` para escaparnos al significado de la conexión.

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
```

Sin embargo vemos que este comando puede ser muy lento. Ubiquémonos en el archivo `Herramientas_Computacionales`. Como el árbol que cuelga desde acá es más pequeño el comando corre más rápido. 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:

```
:hjaramillo>find . -name leon 2>/dev/null  | xargs grep -n Asia
5:desaparecido del resto de Asia del Sur, Asia Occidental,
África del Norte y la península balcánica en tiempos históricos
```

La razón por la cual este comando es más rápido es por que primero busca el archivo (`leon`) y luego dentro del archivo la palabra *Asia ,* mientras que el comando `grep -nr Asia` busca al interior de TODOS los archivos. La búsqueda es muy lenta. Vemos como la combinación de `find` y `grep` puede ser muy poderosa.

En el ejemplo anterior nosotros debemos saber que el archivo con nombre `leon` es el que contiene la palabra `Asia` . ¿ Que tal si no sabemos cuál archivo tiene la palabra *Asia* ? 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: 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
:hjaramillo>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: Vaya al directorio `/usr/share` y haga la siguiente actividad. Cuente todos los archivos en `C`. Solo un código fue desarrollado por Michele Cicciotti. Encuentre este código.

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 `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
```

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 y derecha para moverse dentro del comando y editarlo.

(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/hjaramillo/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^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`.

(v) Finalmente tambíen podemos navegar por los comandos de la historia con las flechas del teclado y editando los comandos usando espacios hacia atrás y escribiendo nuevos comandos.

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
:hjaramillo>!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`. El comando dice `cat felinos` y vamos a modificar `cat` por `stat`. Es decir, con el comando

```
> ^cat^stat
```

y el comando

```
> stat felinos
```

se ejecuta. 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.

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

4.1 Variables Ambiente

Las variables ambiente (del ingés 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/hjaramillo
```

(ii) `HISTSIZE`: Esta variable se describió en la sección 3.4.

(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 Google. Copiemos y peguemos el programa con algún editor y salvémolo en un directorio. Por ejemplo, en el directorio `/tmp` . Llamamos a este programa `hello_world.c` . Por ejemplo tenemos

```
:tmp>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 hjaramillo hjaramillo 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

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

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

```
:Unix>cd
:hjaramillo>hello_world
hello_world: command not found
```

La razón por la cual el comando no se encontró este comando es por que el directorio `/tmp` no está en el camino de búsqueda de la variable ambiente `PATH`. Listamos el contenido de la variable ambiente `PATH` con el comando `echo` como sigue.

```
:hjaramillo>echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
/bin:/usr/games: /usr/local/games:
/snap/bin:.:/home/hjaramillo/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`.

```
:hjaramil>export PATH=$PATH:/tmp
:hjaramil>echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin:/usr/games:/usr/local/games:
```

```
/snap/bin:.:/home/hjaramillo/SU/bin:
/usr/share/sage-6.8-x86_64-Linux:/tmp
:hjaramil>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/hjaramillo/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/hjaramillo/Herramientas_Computacionales
:Unix>echo $HC
/home/hjaramillo/Herramientas_Computacionales
:Unix>env | grep HC
HC=/home/hjaramillo/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 `HR`. Veamos

```
:hjaramillo>cd /
:>cd $HR
:Herramientas_Computacionales>pwd
/home/hjaramillo/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.

Actividad: 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.

```
:hjaramillo>lm
lm: command not found
:hjaramillo>alias lm='ls -lt | more'
:hjaramillo>lm
total 340
```

```
drwxr-xr-x  4 hjaramillo hjaramillo  16384 mar  1 15:12 Downloads
drwxr-xr-x  2 hjaramillo hjaramillo   4096 mar  1 13:53 Pictures
-rw-rw-r--  1 hjaramillo hjaramillo  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: Cree un alias para el comando
`cd /home/hjaramillo/Herramientas_Computacionales`
con el nombre de `cdhr` . Verifique que el alias quedó bien creado. Observe que acá `hjaramillo` soy yo. Usted necesita cambiar `hjaramillo` 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 ventana) 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 `~/.bashrc` es oculto (con el fin de protergerlo contra borrado accidental. Es un archivo importante del sistema. El significado (en inglés) original de bashr es Born Again Shell Run Commands .

Actividad: 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 esté, al directorio `~/Herramientas_Computacionales`. 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` .

Actividad: Revise el contenido de la variable ambiente `HR`. Si esta existe entonces elimínela con el comando `unset HR` . 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` . 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`.

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 es 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`).

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

```
:hjaramillo>python delunoal10.py
```

Observamos que el programa imprime 1 y sigue sin parar [15]. 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

```
> 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 simultáneamente; 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

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

```
> bg
```

que significa “background”. Este comando trae a la pantalla el progrma 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 o Ctrl-Sh-n). Para suspender este trabajo podemos usar el comando ps . Este herramienta lista los procesos que están corriendo en el sistema. Podemos inicialmente correr el 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: Tamańo de la memoria virtual

(5) RSS: Tamańo 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)
- + : Est’a 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
hjaramillo  8648  0.2  0.0  25544  6316 pts/2 T    10:43   0:02 python delunoal10.py
hjaramillo  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 este comando.

Por supuesto que una manera de terminar el proceso es cerrando la ventana donde este está corriendo 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 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.

Comencemos con el programa más simple conocido como (en inglés) `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.c` . 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"
sleep 4
echo "Buenos Dias"
:Bash>hola_mundo.sh
Hola Mundo
Buenos Dias
```

Vemos que se ejecuta la línea que imprime `Hola Mundo` y luego de 4 segundos la línea con el texto `Buenos Dias`. 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.

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.

6.1 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"
```

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 `+%A` extrae el día de la salida de `date` . 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
```

6.2 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 "el directorio $dir_a tiene $num_a archivos"
echo "el directorio $dir_b tiene $num_b archivos"
```

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

La salida de este programa se muestra a continuación

```
:Bash>condicional.sh
el directorio /usr/local tiene 10 archivos
el directorio /etc tiene 259 archivos
el directorio /usr/local tiene menos archivos que el directorio /etc
```

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 `/usr/local` por `$1` y `/etc` por `$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 "el directorio $dir_a tiene $num_a archivos"
echo "el directorio $dir_b tiene $num_b archivos"

if [ $num_a -lt $num_b ]; then
  echo "el directorio $dir_a tiene menos archivos que el
     directorio $dir_b"
else
```

```
  echo "el directorio $dir_a tiene mas archivos que el
     directorio $dir_b"
fi
```

Lo corremos como sigue

```
:Bash>condicional_argumentos.sh /etc /usr/local
el directorio /etc tiene 259 archivos
el directorio /usr/local tiene 10 archivos
el directorio /etc tiene mas archivos que el directorio /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 [16]

6.3 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.

6.3.1 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.

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

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

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 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 `/tmp/Basura`, 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 `/tmp/Basura` 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.

6.3.2 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.

7 Comandos para conocer el sistema

A continuación hacemos una lista incompleta de comandos útiles para conocer el sistema donde trabajamos. En cada uno de ellos se puede consultar el manual (`man command`) para mayor información.

- `uname` : Produce el nombre del sistema donde estamos trabajando. Si se usa la bandera `-a` muestra toda la información que se puede capturar de este comando. Pruebe otras banderas.

- 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 .