Capítulo 1

Unix / Linux

Una gran parte de la comunidad científica lleva a cabo su desarrollo e investigación en un ambiente `UNIX`, sea a través de un computador con sistema operativo `UNIX`, `Linux`, `Mac OS X`, o incluso `Windows`. `UNIX` es uno de los sistemas operativos más comunes en estaciones de trabajo (workstations) dentro de redes de computadores, y que hoy predomina en la computación científica. En muchos casos, las redes computacionales de las distintas universidades del mundo tienen toda la información de los usuarios en un servidor, cuyo sistema operativo esta basado en `UNIX`.

Hoy, sistemas `UNIX` incluyen Solaris, HP-UX, etc., o `tipo-Unix` (UNIX-like) como por ejemplo `Linux` y `OpenBSD`. En nuestro trabajo científico es probable que trabajemos en una máquina con sistema operativo `Linux`, una versión libre de `UNIX`, que puede ser utilizada en computadores personales y de escritorio y que ha tenido muy buena aceptación de la comunidad. Muchas personas tiene un computador con sistema operativo `Mac OS X`, el cual también se puede manejar como un `UNIX` a través de la Terminal. Y claro, no hay razón para no poder utilizar un computador con sistema operativo `Windows`, para lo cual es necesario utilizar programas gratuitos como `CygWin`.

Todos los sistemas `tipo-UNIX` tienen una estructura básica de archivos y carpetas. La carpeta principal o `root` (no confundir con el usuario root, que tiene todos los privilegios de editar archivos) se encuentra en `/`, y dentro de esa carpeta hay directorios como `usuarios` o `users`, donde se encuentra ubicada la cuenta personal de los usuarios. Por ejemplo, el usuario que vamos a usar en este texto se llama Pedro Perez, y su cuenta de usuario es `pperez`, y su carpeta, datos, archivos, programas, etc., estn ubicados en `/users/pperez`. En un computador `Mac`, sería `/Users/pperez`.

Otros directorios que se encuentran en el `root` como `/bin /var /sbin` y muchos mas, no deben ser editados, borrados ni cambiados de ubicación, a menos que realmente el usuario sepa lo que está haciendo y tiene permisos de `root` o de `sudo` (super-user). Como se vera mas adelante es posible que el usuario tenga que llamar archivos que estn dentro de estas carpetas para correr programas básicos de `Unix`, utilizar librerías, etc.

Para usar Unix en un Linux es necesario abrir una ventana de texto, como por ejemplo una ventana del programa Terminal o una ventana Xterm. Esto se puede hacer oprimiendo el botón derecho del mouse y seleccionando Terminal. En un Mac, el programa Terminal o X11 se encuentran en /Applications/Utilities/. Para hacer uso de Unix en un Windows se recomienda descargar e instalar CygWin (http://www.cygwin.com/).

De acá en adelante se asume que la Terminal está abierta y que los comandos Unix se están escribiendo en dicha ventana. También se asume que el X-environment se encuentra disponible para poder abrir programas basados en X. Para el Mac, es necesario tener abierto el programa X11, que generalmente viene pre-instalado.

Unix viene de varias formas y colores, y esto se ve reflejado en las Shell de Unix. Una shell es un *interpretador de comandos de texto* con una interfaz del usuario de los sistemas operativos tipo-Unix. El usuario a través de las shell puede dar instrucciones al computador escribiendo comandos, que el interpretador lee, y ejecuta. También se pueden generar scripts con uno o más comandos.

Programa 1.1 Ejemplo de un comando Unix sencillo (pwd) en la Terminal.

```
pperez 1> pwd
/Users/pperez
pperez 2>
```

Las dos shells mas utilizadas son la Bourne Shell sh y la C shell csh o una versión similar tcsh. En un ambiente Unix los usuarios deciden que shell usar para su trabajo. Cuando el usuario comienza su sesión, el sistema automáticamente ejecuta el programa shell. Y aunque hoy en día hay shells con interfaz gráfica, en general se usa el termino shell para comandos de texto. En sistemas que usan ventanas (como Windows) el usuario puede nunca tener la necesidad de usar un shell directamente, aunque este funcionando detrás de los clicks.

Aunque existen muchas shell distintas, en este texto vamos a utilizar la tcsh o csh. Una de las mas populares hoy en día es bash, que en varios sistemas operativos es el *default*. Todo lo que se encuentra en este capítulo puede ser llevado a cabo también en bash, aunque no está explicado en esta edición. Hay mucha discusión sobre cual es la mejor opción de shell y de ninguna manera creemos ser expertos en shell ni afirmamos que tcsh es mejor, simplemente es la que mejor manejamos.

Si el usuario quiere saber que shell está utilizando, puede escribir el comando (Programa 1.2) o para una lista más corta (Programa 1.3). Si el usuario desea cambiar su shell predeterminada (Programa 1.4).

En lo que sigue abajo solo se describe de manera básica el manejo de Unix a través de un shell, principalmente pensando en aquellos lectores que nunca han sido expuestos a comandos de linea o Unix. Nosotros no somos expertos y no esperamos convertir al lector en experto, solo que se pueda defender en un

Programa 1.2 Comando para recibir información sobre las variables del ambiente del usuario.

```
pperez 1> printenv
```

Programa 1.3 Comando para recibir información sobre la `shell` en uso.

```
pperez 1> printenv SHELL
```

Programa 1.4 Comando para cambiar la `shell` predeterminada. La próxima vez que se abra la `Terminal` se utilizará `tcsh`.

```
pperez 1> chsh -s /bin/tcsh
```

ambiente de `Unix`.

El sistema operativo `Unix` fue originalmente diseñado para computadores centrales (*mainframe*) donde la seguridad es muy importante y varios usuarios pueden conectarse. No se quiere que un usuario pueda borrar los archivos de otro o que pueda alterar los programas, llibrerías, etc. Para esto hay muchas *características* de seguridad.

Lo primero para la seguridad es el `login` o nombre de usuario y la clave o `password`. Tanto en el computador personal como en una cuenta en un servidor, cada usuario tiene su `login` (pperez en nuestro caso) y una clave que solo el usuario debe saber [Para cambiar la clave ver Programa 1.6].

Programa 1.5 Comando para cambiar la clave del usuario. Usualmente se le pide la clave actual y la nueva (con confirmación).

```
pperez 1> passwd
```

Obviamente el usuario debería elegir un clave compleja, pero que siempre recuerde. No hay necesidad de compartir la clave con ningún otro usuario, ya que se le puede dar acceso a los archivos a cualquier usuario de la red, sin necesidad de darle la clave.

Asumiendo que se haya iniciado una sesión (logged in) y esté abierta una `Terminal`, el usuario estará ubicado en su directorio `home`. `Unix` usa una estructura de directorios, similar a *folders* en los PCs, pero sin los íconos. En muchos casos, el cursor de la terminal tiene información de la máquina en la cual se está trabajando, o el nombre de usuario o la ubicación dentro del árbol de directorios.

Programa 1.6 Ejemplo de cursor de la Terminal con `tcsh` para usuario `pperez`, usando el computador llamado `compu1`. Similar ejemplo para `bash` en el computador `compu2` y donde además se muestra la ubicación dentro del árbol de directorios `~/Desktop`. Es posible poner el cursor de la manera que el usuario desee, como se muestra en la tercera linea. En muchos casos es conveniente poner este comando en el archivo de inicio conocido como `.tcshrc` o `.bashrc`.

```
pperez@compu1:~>

[compu2:~/Desktop] pperez%

set prompt="%n@%m %h> "

# n es nombre de usuario
# m es el nombre del computador
# h es el numero de linea.

pperez@compu1 2>
```

1.1. Comandos Básicos

A continuación se explican varios comandos básicos de `Unix`, muchos de los cuales un usuario debe saberse de memoria. Hay varios que no se aprenden comúnmente, pero que siempre se pueden consultar a través de las páginas de manuales (man pages) o a través de la web. Lo que se sugiere al lector es aprender algunos comandos de uso común y conocer la capacidad de otros comandos aún sin saber de memoria el comando o la forma de llamarlo.

El comando `pwd` (*Print Working Directory*) despliega el directorio donde el usuario está ubicado. El comando `ls` despliega la lista de contenidos del directorio actual. Si se adiciona un parametro adicional como `ls -F` se puede diferenciar si los archivos son directorios, ejecutables, etc. [Programa 1.7].

Programa 1.7 Ejemplo de comandos `pwd` y `ls`.

```
pperez@compu1 2> pwd
/users/pperez

pperez@compu1 2> ls
cuentas.txt  Desktop  Documents

pperez@compu1 2> ls -F
cuentas.txt  Desktop/  Documents/
```

Unix posee un conjunto de manuales en línea que puede ser accesado con

el comando `man`. Por ejemplo, si el usuario olvidó que hace `pwd`, usando este comando [ver Programa 1.8] se obtiene la descripción de `pwd`. Una de las características de Unix es que el comando `man` despliega el manuel en una forma del editor `VI` en vez de desplegarlo en una ventana diferente, donde el usuario pueda subir y bajar con el mouse. El comando sencillo hace que se despliegue en la Terminal, pero no es posible subir y bajar el manual con las flechas del teclado. Lo que se puede hacer es guardar el resultado del comando en un archivo de texto sencillo (ascii) usando `man pwd >man.pwd`, donde el archivo tiene el nombre `man.pwd`.

Programa 1.8 Ejemplo de comando `man`. También se puede guardar en un archivo de texto `man.pwd` que luego se puede desplegar con el comando `cat`.

```
pperez@compu1 2> pwd
PWD(1)          BSD General Commands Manual         PWD(1)

NAME
     pwd -- return working directory name

SYNOPSIS
     pwd [-L | -P]

DESCRIPTION
     The pwd utility writes the absolute pathname of the
     current  working directory to the standard output.
...

pperez@compu1 3> man pwd > man.pwd

pperez@compu1 3> cat man.pwd
...
STANDARDS
     The pwd utility conforms to IEEE Std
     1003.1-2001 (‘‘POSIX.1’’).

BUGS
     In csh(1) the command dirs is always faster because
...
     The -L option does not work unless the PWD environment
     variable is exported by the shell.

BSD                  April 12, 2003                  BSD
```

El usuario puede abrir con cualquier editor de texto regular (nedit, vi, emacs, etc., que más tarde se discutirán). También es posible ver el contenido del archivo

con el comando `cat`, el cual despliega el contenido completo de un archivo de texto en la Terminal y en este caso sí se puede utilizar la barra de desplazamiento o el mouse y/o teclado.

Los comandos en Unix muchas veces tienen opciones adicionales que pueden hacerlos más útiles para el usuario. Por ejemplo `ls` despliega la lista de contenidos del directorio. Si se quiere tener la lista más completa con fechas de creación, permisos, etc., se puede utilizar `ls -l` donde la `l` significa *long output*. En el directorio `home` del usuario existen varios archivos *escondidos*, muchos de gran importancia como el archivo `.tcshrc`. Con el comando simple, el archivo no aparece en la lista, pero si aparece con el comando `ls -a`, donde la `a` se refiere a *all files*.

Programa 1.9 Ejemplo de movimiento, creación y remover directorios.

```
cd dirname
# Para moverse al directorio dirname
cd /users/pperez
# Para moverse a un directorio, con el path exacto (directorio home)
cd
cd ~/
# Igual al anterior, para ir al directorio home
cd ~/dir1/dir2
# Para ir a un directorio dentro de otro. El dir2,
# ubicado dentro de dir1, a su vez ubicado
# en el directorio home.

cd
cd dir1
cd dir2
# O se puede hacer paso a paso.

cd ..
# Para devolverse un directorio.

cd ../dir2
# Devolverse un directorio e ir a dir2.

mkdir dir1
# Esto crea un directorio llamado dir1
rmdir dir1
# Para remover este directorio
```

Comandos para cambiar de directorios, crear directorios, e incluso remover directorios se encuentra en Programa 1.9. Muchas personas prefieren utilizar una convención para diferenciar entre archivos y directorios, por ejemplo utilizando

`nombre.dir` para failmente saber que este es un directorio y no un archivo de texto o un ejecutable. Otras personas prefieren utilizar mayúsculas para directorios `NOMBRE`, pero lo hace un poco más complicado ya que Unix diferencia entre mayúsculas y minúsculas. Si el usuario prefiere, se puede utilizar el comando `ls -F` donde automáticamente Unix despliega diferencias entre directorios, archivos, ejecutables, etc. Para directorios se tiene un sufijo `/`, para ejecutables un `*`. Es además posible hacer un alias para que cada vez que el usuario digite `ls`, el comando sea `ls -F`. Sobre alias ver más adelante.

Programa 1.10 Ejemplo de `mv` y `cp`, mover y copiar archivos y directorios.

```
rm filename
# remover archivo filename
mv fname1 fname2
# mover arcivo fname1 a fname2
# Si fname2 existe, es sobreescrito.
mv -i filename1 filename2
# Para evitar sobreescribir.

# mv can also be used to move files between directories:

mv fname dirname
# Mover fnane a directorio (con el mismo nombre).
# Asume que directorio existe!!
mv fname dirname/fname
# es equivalente

The copy command works in a similar way:

cp fname1 fname2
# Similar a comando mv, pero no desaparece fname1
cp -i filename1 filename2
# pregunta si fname2 existe, si quiere sobreescribir.

rmdir dirname
# Elimina el directorio dirname
# Asume que directorio esta vacio
rm -r dirname
# remueve el directorio y su contenido
# Usar con mucha precaucion
```

Para borrar archivos, se utiliza el comando `rm` que significa *remove*. Si lo que se quiere es cambiar el nombre de un archivo o moverlo a otro ubicación se usa el comando `mv` o *move*. Tener cuidado ya que el archivo original desaparece, para lo cual el comando `mv -i` pregunta al usuario si quiere sobreescribir el archivo

existente. Esta opción muchas personas prefieren ponerla como el *default*, en el archivo `.tcshrc` (que veremos más adelante).

1.2. Archivos y edición de texto

Los archivos en Unix pueden ser de texto simple (ascii) o pueden ser archivos binarios (usualmente versión ejecutable de programas de computador). Un editor estándar en Unix es conocido como `Vi` o `Vim` y sigue siendo utilizado por algunos científicos que piensan que es el mejor editor de texto. Si el lector se sabe todos los trucos de `Vi`, éste puede llegar a ser un editor muy poderoso, con capacidades inexistentes en otros editores. Además tiene la gran ventaja de poder ser utilizado en cualquier terminal, así que es posible iniciar una sesión en un servidor al otro lado del mundo y aún así seguir editando archivos (en un Mac, Windows, Linux, etc.). Si el lector sabe o quiere aprender `Vi`, perfecto, pero de antemano no es el editor más sencillo de aprender.

Sin embargo, `Vi` no es un editor amigable para Windows, ni usa el mouse. Otro poderoso editor de texto es `Emacs`, que según muchos conocedores es más moderno y mucho más poderoso que `Vi`. Sin embargo, tiene las mismas desventajas en el sentido de no ser amigable ni usa el mouse.

Existen otros editores amigable en Mac, Windows, Linux, donde se puede hacer *copy-paste* con el mouse, con el teclado, se puede hacer *scroll*, etc. Muchos de ellos lastimosamente son propias de cada plataforma (Mac por ejemplo tiene XCode). Se recomienda utilizar editores multi-plataforma, ya que estos se aprenden una vez y se pueden utilizar en cualquier sistema operativo. Algunos editores son:

- vi – line editor. Muy poderoso, pero de la vieja escuela.
- emacs – muy poderoso, uno de los más usados por profesionales de la computación. Puede ser difícil de aprender. Si el lector es nuevo en métodos computacionales, recomiendo aprender este editor.
- nedit – Editor X-Window system. Sencillo.
- gedit – Editor of the GNOME desktop environment. Sencillo.

El usuario puede utilizar cualquiera de estos editores para crear un nuevo archivo o editar uno ya ya existente. Tenga en cuenta que los archivos en Unix son *case-sensitive* (sensible a mayúsculas/minúsculas). Por convención, el tipo de archivo generalmente indica el tipo de archivo que es; por ejemplo:

- testprog.f – Código fuente para Fortran 77
- testprog.f90 – Código fuente para Fortran 90/95
- testprog.c – Código fuente para C
- testprog.cc – Código fuente para C++

- testprog.m – Código fuente de script para Matlab.
- testprog.o – Archivo tipo objeto (para .f, .f90, .c, .cc)
- figure1.ps – Archivo postscipt (tipo de archivo utilizado por impresoras por ejemplo).
- figure1.gif – Archivo tipo GIF (figuras por ejemplo).

Es cuestión personal pero una buena idea tener un criterio claro de la convención que se quiera utilizar. Puede ser muy útil para manejar archivos de un mismo tipo con *Wildcards* (ver abajo).

PRECAUCIÓN: No es recomendable utilizar el caracter `-` en nombres de archivos; esto puede llevar a varios problemas. Use `_` o `.` para separar palabras. Tampoco utilice espacios en blanco en nombres de archivos o directorios (esto probablemente no es problema en un PC, pero si es complicado cuando intente leer los archivos usando fortran o matlab).

1.3. Wildcard (Comodín)

Los comandos Unix se vuelven muy poderosos cuando son utilizados con el caracter *comodín* o *Wildcard*, que se usa con el signo`*`, que reemplaza cualquier caracter o grupo de caracteres. Por ejemplo, puede ser muy útil si el usuario quiere ver la lista de archivos que terminan en `.f` (programas de fortran 77), se digita:

```
ls *.f
```

Todos esos archivos se pueden mover al mismo tiempo al subdirectorio `src` así:

```
mv *.f  source
```

O por ejemplo el usuario tiene miles de archivos con figuras llamados `mypost1`, `mypost2`, etc. Para eliminar a todos:

```
rm mypost*
```

Por obvias razones el uso de este caracter es de cuidado cuando se combina con `rm`. Por ejemplo, supongamos que queremos eliminar todos los archivos terminados en `%`, que algunos editores de texto usan para guardar copias temporales de edición. Para hacer esto:

```
rm *%
```

pero si por error se pone

```
rm * %
```

TODOS los archivos en el directorio actual serán elliminados!! Tener siempre cuidado con el uso del wildcard y comando como `rm` o `mv`.

1.4. Archivos de Configuración (.xxxrc)

En el directorio`home` de cada usuario se puede tener un archivo llamado `.cshrc` (o dependiendo de la shell en uso, puede ser `.tcshrc` o `.bashrc`), los cuales siempre es leído y ejecutado cada vez que se inicia una sesión. Estos archivos son usados para definir propiedades preferidas por el usuario y personalizar el ambiente Unix. En algunos casos el personal de IT puede proporcionar automáticamente este tipo de archivos al usuario y se recomienda usar este archivo como base y adicionar personalización del ambiente sobre este archivo base.

Una de las personalizaciones que se pueden adicionar, son *aliases*, como por ejemplo

```
alias cp 'cp -i'
alias mv 'mv -i'
```

para evitar borrar accidentalmente archivos importante. Para *quitar* un alias, se digita

```
unalias cp
```

Programa 1.11 Archivo `.tcshrc` ejemplo. Puede también ser llamado `.cshrc`. Se cambio al cursor, se adiciona el directorio actual `.` y otro directorio personal `/users/pperez/programs` y se definen varios *aliases*. Note además que el signo # representa un comentario, lo que le sigue no es tenido en cuenta.

```
#Editar cursor
set prompt="%n@%m %h> "

#Adicionar el path
set path = ($path . /users/pperez/programs/ )

# Aliases
alias ls "ls -F"
alias ghostview   "gv"
alias matlab1 matlab -nodesktop
```

Otro de los parámetros importantes del archivo de configuración es la asignación del `path`. Este `path` tiene la lista de directorios (y su orden) que el computador va a leer cuando se le proporciona un comando. Es así como el usuario al escribir `>ls`, el computador sabe donde buscar el comando apropiado. Si al escribir un comando se obtiene

```
matlab
matlab: Command not found.
```

quiere decir que la lista de directorios en el `path` no es correcta y que no puede encontrar `matlab`. Para adicionar directorios al `path` se utiliza el comando `set path` [ver Programa 1.11 o 1.12].

En algunos casos, el `path` no tiene un su lista el directorio actual, el directorio donde el usuario esta trabajando y esto hace que aún teniendo un ejecutable en ese directorio, el programa no se encuentra. Para arreglar esto, se debe adicionar al `path` un `.` para que siempre busque en el directorio actual.

Cualquier cambio al archivo `.xxxrc` sólo surte efecto al abrir una nueva terminal o ejecutando el comando `source .xxxrc`. Tenga en cuenta que en los dos ejemplo de archivos de configuración, siempre el `path` que está predeterminado es utilizado nuevamente, para evitar eliminar directorios que vienen de manera automática con el sistema operativo y eliminar directorios que el personal de IT ha proporcionado.

Programa 1.12 Archivo `.bashrc` ejemplo. Igual resultado al archivo `.tcshrc` en Programa 1.11. Note la diferencia en notación para el `path`, cursor y alias.

```
# Editar el cursor
export PS1="\u@\H > "

#Adicionar el path
PATH=$PATH:./:~pperez/programs/
export PATH

#Aliases
alias ls="ls -F"
alias gv="ghostview"
alias matlab1="matlab -nodesktop"
```

1.5. Transferencia de archivos (sftp, scp)

En algunos casos es necesario obtener archivos de otro computador (de la oficina a la casa por ejemplo). Una forma de hacer esto es con el comando `sftp` (secure file transfer protocol). En el caso de muchas páginas públicas y centros de bases de datos esto se hace con `anonymous ftp`. Simplemente se digita:

```
sftp otrocomputador
```

donde `otrocomputador` es el nombre (o dirección IP) del otro computador. Cuando el otro computador pida los datos de acceso (login), simplemente se pone `anonymous` y luego se escribe el correo electrónico. Por supuesto que si el usuario tiene una cuenta en ese computador, puede usar su cuenta, y en este caso aún si el otro computador no tiene abierto `anonymous ftp` el usuario puede acceder a los datos.

Una vez en `sftp` en la Terminal se observa el cursor `sftp>` y se puede entonces utilizar los comandos típicos de unix, como `cd` para llegar al directorio de interés o `ls` para ver el contenido de un directorio. Finalmente se utiliza el

comando `get` para descargar el archivo que se desea al computador donde se este trabajando. Para salir, simplemente se digita `quit`.

Si se quiere descargar todos los archivos en un directorio, usar el comando `mget *` y la Terminal solicitará confirmación para cada archivo. Si no se quiere tener que confirmar cada vez, se puede desactivar el modo interactivo:

```
sftp -i othercomputer
```

al momento de empezar `sftp`. Si los archivos que se quieren descargar son binarios (no de texto simple) se debe digitar al comenzar la sesión `sftp`

```
type binary
```

aunque esto en los sistemas modernos ya no es necesario. Igual es buena práctica siempre poner `type binary`, ya que funciona tanto con archivos binarios como con archivos de texto, mientras que el *default* en Unix es `type ascii`, el cual no funciona con binarios.

Otra forma de transferencia de archivos es por medio del comando `scp`, *secure copy*. Este comando permite copiar archivos individuales, multiples archivos (con un wildcard), directorios, etc., obviamente después de confirmar el nombre de usuario y la clave. La ventaja de `scp` es que es fácil transferir archivos de un computador remoto al local, e incluso del local al remoto. La desventaja es que es necesario saber con precisión el `path` de los archivos que se quieren transferir [ver Programa 1.13].

Programa 1.13 Ejemplos del uso de `scp` para transferencia de archivos.

```
scp fname *.ps pperez@compu2.servidr.com:./folder1
# transfiere archivo fname y todos los terminados en .ps
# del computador actual al computador compu2, y los
# pone en el directorio folder1

scp -r dirname pperez@compu2.servidr.com:./folder1
# copia de directorio completo y sus contenidos al
# mismo lugar del anterior.

scp -r pperez@compu2.servidr.com:fld1/fld2/dirname dirname_local
# transfiere directorio remoto a computador local.
# Note que es necesario el path completo en computador remoto.
```

1.6. Compresión de archivos

En algunos casos los archivos que se descargan o se utilizan están comprimidos. En Unix hay varias formas de compresión de archivos. Uno de ellos es el comando `compress` donde

```
compress fname
```

donde el nuevo archivo tendrá la siguiente forma `fname.Z`, de tal manera que otro usuario sepa que ese archivo está comprimido. Esto es útil para ahorrar espacio o memoria en el computador si el archivo no es utilizado constantemente. Para descomprimir este tipo de archivos

```
uncompress fname
```

Se pueden comprimir multiples archivos a la vez con `wildcards`, por ejemplo:

```
compress *.ps
```

comprimiendo todos los archivos que terminan en `.ps`. Se puede descomprimir multiples archivos también:

```
uncompress *.ps
```

como es de esperarse.

Una alternativa para compresión de archivos es el comando`gzip`, aunque no es estandar Unix, si es muy común. Se utiliza:

```
gzip fname
```

lo cual cambia el nombre del archivo a `fname.gz`. La operación inversa sería

```
gunzip fname.gz
```

El comando `gunzip` también puede descomprimir archivos `.Z` (el caso contrario no funciona, el comando `uncompress` no puede descomprimir archivos `.gz`).

El lector puede utilizar la compresión de archivos para ahorrar espacio de memoria si es necesario, sin la necesidad de borrar archivos, los cuales no esta seguro que quiera borrar.

1.7. Almacenamiento de archivos y directorios con TAR

En algunos casos el usuario quiere guardar o descargar un directorio entero con un gran número de archivos, y se puede sacar provecho del comando `tar`. Los archivos `.tar` son archivos en un formato ampliamente utilizado en unix. El formato y comando `tar` (Tape ARchiver; archivador en cinta) tiene la característica de poder guardar en un solo archivo `.tar` un gran número de archivos, directorios, etc., de cualquier formato. Tenga en cuenta que `tar` no comprime los archivos, para lo cual se puede utilizar `gzip`.

Para utilizar el comando, si el usuario se encuentra en el directorio que contiene los archivos que se quieren archivar, digitar:

```
tar -cvf ../archive.tar .
```

Los argumentos utilizados son:

- -c crear archivo `tat`
- -v verificar, imprimiendo los archivos en la pantalla
- -f nombre del nombre del archivo sigue
- ../archive.tar es el nombre del archivo (note ubicado un directorio atrás).
- . archive (tar) todos los archivos en el directorio actual `.`

De manera alternativa, se puede archivar el directorio completo y todo su contenido digitando

```
tar -cvf archive1.tar dir_name
```

donde `dir_name` es un directorio. Para reabrir el archivo se digita

```
tar -xvf archive.tar
```

donde la `x` representa el comando extraer. Todos los archivos se des-archivarn en el directorio donde está ubicado el usuario.

El uso de este comando puede hacer mucho más simple la transferencia de información entre usuarios y computadores, ya que solo requiere transferir un archivo `.tar` en vez de miles de archivos pequeños. Para mayor eficiencia se recomienda adicionalmente comprimir los archivos `.tar` usando el comando `gzip`. El comando `tar` tiene una opción para comprimir inmediatamente después de hacer tar:

```
tar -zcvf archive1.tar.gz dir_name
```

donde el archivo tiene formato `tar` y además está comprimido. Para reabrirlo

```
tar -zxvf archive.tar.gz
```

1.8. Trabajando remotamente (desde casa)

En algunos ocasiones es necesario trabajar en un computador distinta al que está enfrente del usuario. El otro computador puede ser más rápido, tener los programas instalados que se requieren, tener los datos o la información que el usuario necesita, etc. Pero no siempre es posible sentarse al frente de eso otro computador y por lo tanto es necesario un protocolo de comunicación para poder *sentarse* al frente de dicha maquina. El comando `ssh` (secure shell) se utiliza así:

```
ssh -Y milogin@comp_remoto
```

a través del cual el usuario puede hacer `login` y trabajar como si estuviera frente al computador. (a través de una terminal. La opción `-Y` se recomienda ya que permite que comandos y programas que utilicen `X-Windows` pueden abrirse en la pantalla. Esto quiere decir que si quiere ver un archivo PDF en el otro

computador, lo puedo hacer. Claro que la cantidad de información que se utiliza puede ser mucha y hacer que la comunicación y el computador sea más lento.

Vale la pena acá resaltar nuevamente lo que significa `milogin - comp_remoto`. La primera parte es obvia, este es el nombre de usuario de la persona que quiere conectarse remotamente y este usuario debe existir y tener una cuenta en el computador remoto. La segunda parte es el nombre del computador. Esto significa el nombre *único* del computador. En la mayoría de casos esto significa la dirección IP del computador, ya que esto es el número único de dicho computador. En casos especiales, el personal de IT puede asignarle un nombre único al computador (mi computador en la oficina se llama `epicentro.uniandes.edu.co`) y me puedo conectar a él desde cualquier lugar del mundo. Sin embargo, para poder hacer esto se requiere que gente experta asigne ese nombre, lo cual quiere decir que para conectarme al computador de la casa, que yo llamo `compucasa`, no puedo simplemente digitar `ssh pperez@compucasa`, y tengo que saber la dirección IP de ese computador (además probablemente mi proveedor ISP tiene un `firewall` que no me permite hacer esto.

Una nota de precaución: A pesar de que Ud posea una cuenta en un servidor o en un computador ajeno no significa que Ud deba utilizar ese recurso indiscriminadamente. En muchas universidades y en la industria, se requiere permiso para utilizar espacio de disco o tiempo de computador.

1.9. Otros Comandos

Unix guarda en su memoria los últimos comandos utilizados. Para ver los últimos 30 comandos digite `h` o `history`. Para ver el comienzo y final de un archivo de texto:

```
head fname     --- despliega las primeras 10 lineas del archivo
tail fname     --- despliega las 10 finales
```

Para ver un archivo (una página a la vez)

```
more fname
```

y utilice la barra espaciadora para avanzar una página. Si no encuentra un archivo en particular, utilice el comando `find` así:

```
find . -name fname -print
```

Este comando genera una busqueda en el directorio actual (se utilizó `.` y subdirectorios por el archivo llamada `fname` y despliega donde está ubicado. Y si solo se sabe que una parte del nombre del archivo? Se puede utilizar `wildcards`:

```
find . -name 'map*' -print
```

para buscar y desplegar todos los archivos cuyo nombre comience con `map`. Note que se debe utilizar comillas para el uso de `*`.

Unix tiene además otros programas útiles. Por ejemplo el comando `sort` que organiza archivos alfabética y numéricamente.

```
sort +4 -n -b -r file1 -o file2
```

Este comando ordena el archivo `file1` y guarda el archivo ordenado en `file2`. Los otros parámetros

```
+4 no tiene en cuenta los 4 primeros valores

-n  orden numerico (default es alfabetico)

-b  ignore valores en blanco

-r orden inverso
```

Para saber cuanta memoria libre se tiene

```
df -h
```

Esto despliega todos los discos disponibles en el sistema. Los números representan bloques de 512 kbytes, por lo que usar el parámetro `-h` es de ayuda ya que los despliega en unidades de kb, Mb, Gb.

Para saber cuanta memoria esta siendo utilizada por un directorio

```
du -ks *
```

desplegando el uso del directorio y todos sus subdirectorios.

Para ver un archivo tipo postscirpt `ps, eps` se puede usar

```
ghostview filename.ps
```

Ghostview o `gv` debe estar instalado en el computador para que el comando funcione.

Por ltimo, es importante tener en cuenta la seguridad y la privacidad de los usuarios. Si el computador es de una universidad, del trabajo o público no usarlo para bajar videos o imágenes no relacionadas con el trabajo. Tambin es importante saber que en general el e-mail no es completamente privado, incluso e-mail borrado puede estar guardado en alguna parte o puede ser guardado en servidores de la universidad, del trabajo, etc. Ya se ha sabido de problemas con e-mail que ha salido a la luz pública en temas como el cambio climático.

1.10. Unix avanzado

Lo vista hasta el momento de Unix representa un muy bajo porcentaje de los comandos disponibles. Con los comandos que se discutieron el usuario puede hacer muchas cosas. Sin embargo, se puede hacer mucho más. Si se quiere ser un gurú de Unix, se recomiendo al lector aprender comandos como pipes, grep, awk,sed, etc. Con ellos se pueden generar `scripts` complejos que hacen muchas cosas interesantes. Nosotros no somos expertos en Unix, simplemente mostramos a continuación algunos ejemlpos más avanzados.
Algunos ejemplos

```
grep santos fname
```

Esto despliega cada linea en el archivo `fname` que contenga el los caracteres `santos`.

```
grep -v santos fname
```

Y este despliega todas las lineas en `fname` que NO contengan `santos`.

```
grep santos filename > st.lines
```

Similar al anterior, pero en este caso no se despliega el resultado en la pantalla sino que se guarda el resultado en el archivo `st.lines`.

```
grep santos *
```

Despliega la lista de lineas que contangan los caracteres `santos` en TODOS los archivos en el directorio actual. Sin embargo, es posible que el usuario quiera saber únicamente la lista de archivos que contienen esos caracteres

```
grep -l santos *
```

y solo despliega el nombre de los archivos que cumplen el requerimiento.

```
grep mtspec `find . -name Makefile -print`
```

Un ejemplo más complicado, donde el comando busca la palabra `mtspec` en todos los archivos cuyo nombre sea `Makefile`. Note que los apóstrofes son inversos.

Para ver los procesos que actualmente están activos en el computador

```
ps -eaf
```

despliega la lista de procesos.

```
ps -eaf | grep pperez
```

genera la lista con procesos que tengan la palabra pperez (el usuario en este caso). El signo `|` se conoce como `pipe` que direcciona la salida del primer comando `ps` como entrada al segundo comando `grep`.

```
ps -eaf | grep pperez > junk
```

Igual al anterior pero se guarda el resultado en el archivo `junk`.

Para terminar un trabajo que no se quiere continuar

```
kill PID
```

lo cual termina el trabajo con proceso número `PID` (que se puede obtener con los comandos `ps`o `top`). Esto es útil para procesos que no tienen fin. Para trabajos que no quieren acabar y son tercos

```
kill -9 PID
```

Comparando dos archivos

```
diff file1 file2
```

genera la lista de diferencias entre los archivos `file1` y `file2`. Esto es útil si se han realizado cambios entre versiones de un mismo archivo y no se recuerda cuales fueron estos cambios.

1.10.1. Ejemplos de uso de SED y AWK

```
cat file1 | sed 's/Pedro/Pablo/' > file2
```

Copiar `file1` a `file2`, substituyendo el primer Pedro de cada linea por Pablo.

```
cat file1 | sed 's/Peter/Paul/g' > file2
```

Copiar `file1` a `file2`, substituyendo cualquier Pedro en cualquier linea por Pablo (note el uso de `g` que significa substitución global).

```
cat file1 | sed 's/^/Pablo es /' > file2
```

Inserte el prefijo `Pablo es` al principio de cada linea. Note que `^` significa comienzo de linea.

```
cat file1 | awk '{print $5,$3,$1}' > file2
```

Asumiendo que el archivo `file1` tiene 5 columnas, copia columnas 5, 3, y 1 al archivo `file2`, omitiendo las columnas 2 y 4.

```
cat file1 | awk '{print $2, $1*(-10)}' > file2
```

Cambia el orden de las columnas 2 y 1, multiplicando la columna 1 por -10.

```
cat file1 | awk '{print "Cols 11 to 20 are " substr($0,11,20)}' > file2
```

Comienza cada linea con `Columns 11 to 20 are` y después despliega cada una de las columnas. Las variaciones de los últimos ejemplos son útiles para editar archivos ASCII.