

BASH

THE BOURNE-AGAIN SHELL

101 Comandos y tips para principiantes y expertos

Introducción a la
Ingeniería en Computación
UNRN Sede Andina
2021



Universidad Nacional
de **Río Negro**

101 Comandos y tips para principiantes y expertos

Comandos Bash en Linux

<https://dev.to/awwsmm/101-bash-commands-and-tips-for-beginners-to-experts-30je>

[Andreew](#) – 13/01/2019 y actualizado 25/09/2019

Traducido y adaptado por Martín René Vilugrón – para la cátedra Introducción a la Ingeniería en Computación de la Universidad Nacional de Río Negro, Sede Andina.

Introducción

Desde hace muchos años, he utilizado GNU/Linux en mis computadoras personales al punto de comprar mis máquinas sin sistema operativo. Hoy día, salvo aplicaciones muy puntuales, podemos hacer todo lo cotidiano sobre esta fantástica pieza de Software Libre. Y a pesar de que mi computadora en la oficina tiene Windows [r], he tenido que interactuar con equipos a los que no se les puede conectar una pantalla, ya sean servidores o equipos embebidos.

Lo que vamos a ver en este apunte, es solo una mínima parte de lo que podemos hacer en terminales UNIX/Bash, y de hecho, como todo lo que es de este tipo de sistemas operativos, existen otras consolas con lenguajes similares y si hay algo para destacar, es lo infinitamente personalizable que es, simplemente no estamos limitados de ninguna forma.

Si crees que sabes todo lo que hay que saber sobre `bash`, dale una mirada de todas formas: he incluido algunos consejos y recordatorios de configuraciones que puede haber olvidado, lo que podría facilitar su trabajo.

Los comandos a continuación se presentan en un estilo más o menos narrativo, por lo que si recién está comenzando con `bash`, puede trabajar desde el principio hasta el final. Las cosas generalmente se vuelven menos comunes y más difíciles hacia el final.

Índice

Introducción.....	1
Los básicos.....	5
Primeros comandos, navegar por el sistema de archivos.....	5
pwd / ls / cd.....	5
Apilando instrucciones ; / &&.....	6
(Doble caño).....	8
Impaciencia &.....	8
Obteniendo ayuda.....	9
-h / --help.....	9
man.....	9
Ver y modificar archivos: head / tail / cat / less.....	10
nano / nedit.....	11
Creando y borrando archivos y directorios.....	11
touch.....	11
mkdir / rm / rmdir.....	13
Moviendo y copiando archivos, creando enlaces y la historia de comandos.....	14
mv / cp / ln.....	14
Historia de comandos.....	15
Árboles de directorios, uso de disco y procesos.....	16
mkdir -p / tree.....	16
df / du / ps.....	17
Misceláneos.....	18
passwd / logout / exit.....	18
clear / *.....	19
Intermedios.....	19
Uso de disco, memoria y procesador.....	19
ncdu.....	19
top / htop.....	20
Versiones de software.....	21
-version / --version / -v.....	21
Variables de entorno y alias.....	22
Variables de entorno.....	22
Alias.....	23
Scripts Bash basicos.....	24
Scripts bash.....	24
Hashbang - #!.....	25
Substitución de comandos.....	25
Prompt personalizado y ls.....	25
Archivos de configuración.....	26

Archivos de configuración / .bashrc.....	26
Tipos de shells.....	27
Encontrando cosas.....	28
whereis / which / whatis.....	28
locate / find.....	29
Descargando cosas.....	30
ping / wget / curl.....	30
apt / gunzip / tar / gzip.....	31
Redirigiendo la Entrada y Salida.....	32
/ > / < / echo / printf.....	32
0 / 1 / 2 / tee.....	33
mktemp.....	34
Avanzado.....	35
Superusuario.....	35
sudo / su.....	35
!!.....	36
Permisos de archivos.....	37
Permisos de archivo.....	37
chmod / chown.....	37
Gestión de usuarios y grupos.....	38
Usuarios.....	38
Procesamiento de texto.....	39
uniq / sort / diff / cmp.....	39
cut / sed.....	40
Búsqueda por patrones (pattern matching).....	41
grep.....	41
Copiando archivos via ssh.....	42
ssh / scp.....	42
rsync.....	43
Procesos de larga duración.....	44
yes / nohup / ps / kill.....	44
cron / crontab / >>.....	45
Misceláneos.....	46
pushd / popd.....	46
xdg-open.....	47
xargs.....	47
Oneliners.....	49
¿Que ocupa más espacio?.....	49
Compartir una carpeta.....	49
Bonus: Cosas divertidas pero en su mayoría inútiles.....	50
w / write / wall / lynx.....	50

nautilus / date / cal / bc.....	51
Bonus II: ¿Que significan los colores de ls?.....	52
Glosario.....	53
La lista de los pendientes.....	58

Los básicos

Primeros comandos, navegar por el sistema de archivos

Los sistemas de archivos modernos tienen árboles de directorios [carpetas], donde un directorio es un directorio raíz [sin directorio principal] o es un subdirectorio [contenido dentro de otro directorio único, al que llamamos su "padre"]. Recorrer hacia atrás a través del árbol de archivos [desde el directorio secundario al directorio principal] siempre lo llevará al directorio raíz.

Algunos sistemas de archivos tienen varios directorios raíz [como las unidades de Windows: `C:\`, `A:\`, etc.], pero los sistemas Unix y similares a este solo tienen una única raíz llamada `/`.

`pwd` / `ls` / `cd`

Cuando trabajamos con un sistema de archivos, el usuario está siempre dentro de algún directorio, el cual llamaremos directorio actual o directorio de trabajo [*working directory*]. Para mostrar el directorio de trabajo actual, utilizaremos `pwd`:

```
[ andrew@pc01 ~ ]$ pwd
/home/ andrew
```

Para mostrar el contenido de este directorio, incluyendo sus archivos y/o sus sub-directorios, utilizaremos `ls`:

```
[ andrew@pc01 ~ ]$ ls
Git  TEST  jdoc  test  test.file
```

Extra:

- Mostrar archivos "ocultos" (inician con punto), utilizaremos `ls -a`
- Para mostrar los detalles, `ls -l`
- Estas opciones se pueden combinar y podemos indicar ambas opciones juntas como `ls -la` o `ls -l -a`

Para cambiar a otro directorio, utilizamos `cd` [change directory]:

```
[ andrew@pc01 ~ ]$ cd TEST/
```

```
[ andrew@pc01 TEST ]$ pwd
/home/andrew/TEST

[ andrew@pc01 TEST ]$ cd A

[ andrew@pc01 A ]$ pwd
/home/andrew/TEST/A
```

`cd ..` es la abreviatura para “`cd` al directorio padre”:

```
[ andrew@pc01 A ]$ cd ..

[ andrew@pc01 TEST ]$ pwd
/home/andrew/TEST
```

`cd ~` o solo `cd` es un atajo para “`cd` al directorio de usuario (home)”, el cual usualmente es `/home/username`

```
[ andrew@pc01 TEST ]$ cd

[ andrew@pc01 ~ ]$ pwd
/home/andrew
```

Extra

- `cd ~user` significa “`cd` al directorio de `user`”
- `cd ~` significa “`cd` al directorio de usuario”
- Es posible saltar múltiples niveles con `cd ../..` y etc.
- Para volver al directorio anterior, `cd -`
- `.` es un atajo para “este directorio”, por lo que `cd .` No tiene ningún efecto.
- No es algo que siempre esté disponible, pero `ls` puede mostrar con colores el tipo de archivos, [bonus II: ¿Qué significan los colores de ls?](#)

Apilando instrucciones ; / &&

Las cosas que escribimos en la línea de comandos se llaman comandos y siempre ejecutan algún código de máquina almacenado en algún lugar de su computadora. A veces, este código de máquina es un comando integrado de Linux, a veces es una aplicación, a veces es un código que usted mismo escribió. De vez en cuando, queremos ejecutar un comando inmediatamente después de otro. Para hacer eso, podemos usar el `;` [punto y coma]:

```
[ andrew@pc01 ~ ]$ ls; pwd
Git TEST jdoc test test.file
/home/andrew
```

Arriba, el punto y coma significa que primero (`ls`) enumero el contenido del directorio de trabajo y luego (`pwd`) imprimo su ubicación. Otra herramienta útil para encadenar comandos es `&&`. Con `&&`, el comando de la derecha no se ejecutará si falla el comando de la izquierda. `;` y `&&` se pueden usar varias veces en la misma línea:

```
# whoops! Hay un error en el primer comando de la cadena!
[ andrew@pc01 ~ ]$ cd /Giit/Parser && pwd && ls && cd
-bash: cd: /Giit/Parser: No such file or directory

# Luego de corregirlo, el resto de la cadena puede ejecutarse
[ andrew@pc01 ~ ]$ cd Git/Parser/ && pwd && ls && cd
/home/andrew/Git/Parser
README.md doc.sh pom.xml resource run.sh shell.sh source src target
```

Sin embargo, con `;`, la cadena de instrucciones seguirá ejecutándose incluso si hay un error.

```
# pwd y ls se ejecutan igual, incluso luego del error en el cd
[ andrew@pc01 ~ ]$ cd /Giit/Parser ; pwd ; ls
-bash: cd: /Giit/Parser: No such file or directory
/home/andrew
Git TEST jdoc test test.file
```

! Por más de que sea posible, eviten la tentación de apilar líneas sin una buena razón. La legibilidad del script baja rápidamente.

¿En qué momento usar `&&`?

Usar `&&` puede caer en dos situaciones muy diferentes, primero, queremos hacer un script rápido que podamos cargar en una sola pasada:

```
[ andrew@pc01 ~ ]$ cd Parser && pwd && ls && cd
```

Esto cambia el directorio a `Parser`, muestra donde estamos, muestra el contenido del directorio y vuelve al directorio `$HOME`.

Ahorrar en líneas dentro de un script no ayuda a la legibilidad, complicándolas instrucciones que son ejecutadas y haciendo mas fácil que perdamos de vista los comandos de la derecha.

|| (Doble caño)

De forma similar a `&&`, agregar `||` luego de un programa, hace que el que esta a la derecha se ejecute solo cuando el de la izquierda falla. Es una excelente manera de resaltar los problemas que se puedan ocasionar en scripts.

```
[ rodrigo@pc01 ~ ]$ cd NoExistente || echo "FAIL"
bash: cd: NoExistente: No existe el archivo o el directorio
FAIL
```

Aporte por Rodrigo Locatti [2021]

Impaciencia &

`&` se parece a `&&` pero en realidad cumple una función completamente diferente. Normalmente, cuando ejecuta un comando de ejecución prolongada, la línea de comando esperará a que ese comando finalice antes de que le permita ingresar otro. Poner `&` después de un comando permite ejecutar un nuevo comando mientras uno más anterior sigue activo:

```
[ andrew@pc01 ~ ]$ cd Git/Parser
[ andrew@pc01 ~ ]$ mvn package & cd
[1] 9263
```

Extra:

Cuando usamos `&` después de un comando para "ocultarlo", decimos que el trabajo [o el "proceso"; estos términos son más o menos intercambiables] está "en segundo plano". Para ver qué trabajos en segundo plano se están ejecutando actualmente, use el comando `jobs`:

```
[ andrew@pc01 ~ ]$ jobs
[1]+  Running cd Git/Parser/ && mvn package &
```

Obteniendo ayuda

-h / --help

Escriba `-h` o `--help` después de casi cualquier comando para abrir un menú de ayuda para ese comando:

```
[ andrew@pc01 ~ ]$ du --help
Modo de empleo: du [OPCIÓN]... [FICHERO]...
    o bien: du [OPCIÓN]... --files0-from=F
Summarize disk usage of the set of FILES, recursively for directories.

Los argumentos obligatorios para las opciones largas son también obligatorios
para las opciones cortas.
    -0, --null          termina cada línea con NUL, no con nueva línea
    -a, --all           muestra resultados para todos los ficheros, no sólo
                        para los directorios
    --apparent-size     muestra los tamaños aparentes, en lugar del uso de
                        disco; el tamaño aparente es normalmente más
pequeño,
                        puede ser más grande debido a agujeros en ficheros
                        dispersos, fragmentación interna, bloques
indirectos,
                        etc.
    -B, --block-size=TAM escala los tamaños por TAM antes de mostrarlos;
                        p. ej., '-BM' muestra los tamaños en unidades de
                        1.048.576 bytes; vea el formato de TAMAÑO más abajo
...
```

man

Escriba `man` antes de casi cualquier comando para que aparezca un el manual para ese comando (salga de `man` con `q`):

```
LS(1)                                User Commands
LS(1)

NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
```

```
List information about the FILES (the current directory by
default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is
speci-
fied.

Mandatory arguments to long options are mandatory for short
options
too.

...
```

Ayuda adicional en español sobre la herramienta, en el [Wiki de archlinux](#)

Ver y modificar archivos: head / tail / cat / less

head muestra las primeras líneas de un archivo. La bandera **-n** especifica el número de líneas que se mostrarán (el valor predeterminado es 10):

```
# Muestra las primeras tres lineas del archivo
[ andrew@pc01 ~ ]$ head -n 3 c
this
file
has
```

tail muestra las últimas líneas de un archivo. Puede obtener las últimas **n** líneas (como arriba), o puede obtener el final del archivo comenzando desde la N-ésima línea con **tail -n +N**:

```
# Muestra el final del archivo, comenzando por la cuarta linea.
[ andrew@pc01 ~ ]$ tail -n +4 c
exactly
six
lines
```

cat concatena una lista de archivos y los envía al flujo de salida estándar (generalmente la terminal). **cat** se puede usar con un solo archivo o con varios archivos, y a menudo se usa para verlos rápidamente. [Tenga cuidado: si usa **cat** de esta manera, puede ser acusado de un uso inútil de **cat** ([useless use of cat, UUOC](#)), pero de todas formas, no es un gran problema, así que no te preocupes mucho por eso].

```
[ andrew@pc01 ~ ]$ cat a
```

```
file a

[ andrew@pc01 ~ ]$ cat a b
file a
file b
```

`less` es otra herramienta para ver rápidamente un archivo: abre una ventana de solo lectura similar a `vim`. (Sí, hay un comando llamado `more`, pero `less`, a pesar de su nombre poco intuitivo, ofrece un superconjunto de la funcionalidad de `more` y se recomienda sobre él). Obtenga más información [¿o menos?] Sobre `less` y `more` en sus páginas de manual.

nano / nedit

`nano` es un editor de texto de línea de comandos minimalista. Es un gran editor para principiantes o personas que no quieren aprender un millón de atajos. Fue más que suficiente para mí durante los primeros años de mi carrera como programador (recién ahora estoy empezando a buscar editores más poderosos, principalmente porque definir su propio resaltado de sintaxis en `nano` puede ser un poco complicado).

`nedit` es un pequeño editor gráfico, abre una ventana X y permite editar con apuntar y hacer clic, arrastrar y soltar, resaltar sintaxis y más. A veces uso `nedit` cuando quiero hacer pequeños cambios en un script y volver a ejecutarlo una y otra vez.

Otros editores comunes de CLI (interfaz de línea de comandos) / GUI (interfaz gráfica de usuario) incluyen `emacs`, `vi`, `vim`, gedit, Notepad++, Atom y muchos más. Algunos geniales con los que he jugado (y puedo respaldar) incluyen Micro, Light Table y VS Code.

Todos los editores modernos ofrecen comodidades básicas como buscar y reemplazar, resaltado de sintaxis, etc. `vi` (*m*) y `emacs` tienen más funciones que `nano` y `nedit`, pero tienen una curva de aprendizaje mucho más pronunciada. ¡Pruebe algunos editores diferentes y encuentre uno que funcione para usted!

Creando y borrando archivos y directorios

touch

`touch` se creó para modificar las marcas de tiempo (timestamps) de los archivos, pero también se puede utilizar para crear rápidamente un archivo vacío. Puede crear un nuevo archivo abriéndolo con un editor de texto, como `nano`:

```
[ andrew@pc01 ex ]$ ls  
[ andrew@pc01 ex ]$ nano a
```

Editando el archivo

```
[ andrew@pc01 ex ]$ ls
```

```
a
```

O simplemente utilizando *touch*:

```
[ andrew@pc01 ex ]$ touch b  
[ andrew@pc01 ex ]$ ls  
a  b
```

Extra

Podemos enviar un programa como los editores a segundo plano con `^z` (`Ctrl` + `z`).

```
[ andrew@pc01 ex ]$ nano a
```

Y mientras editamos el archivo, lo enviamos a segundo plano utilizando `^z` (`Ctrl` + `z`).

Para regresar al editor, utilizamos el comando *fg*.

```
[1]+ Stopped nano a  
[ andrew@pc01 ex ]$ fg
```

Y volvemos al editor.

Extra II

Para matar el proceso en primer plano, podemos utilizar `^c` (`Ctrl` + `c`) mientras está en ejecución, esto es útil para interrumpir un programa que se haya colgado.

Pero para matar un proceso en segundo plano con *kill %N*, necesitamos el número de tarea *N*, que es el número que aparece al presionar `^z` (`Ctrl` + `z`), o que podemos ver usando el comando *jobs*.

mkdir / rm / rmdir

mkdir es utilizado para crear directorios nuevos y vacíos:

```
[ andrew@pc01 ex ]$ ls
a  b
[ andrew@pc01 ex ]$ mkdir c
[ andrew@pc01 ex ]$ ls
a  b  c
```

Pueden borrar cualquier archivo con **rm** – ¡Pero con cuidado que los archivos no son recuperables! (no hay papelera).

```
[ andrew@pc01 ex ]$ rm a
[ andrew@pc01 ex ]$ ls
b  c
```

Pueden agregar un “estás seguro” la bandera **-i**.

```
[ andrew@pc01 ex ]$ rm -i b
rm: remove regular empty file 'b'? y
```

Remover un directorio vacío con **rmdir**, podemos decir que un directorio está vacío cuando con **ls -a**, solo vemos referencias a sí mismo **[.]** y al directorio padre **[..]**:

```
[ andrew@pc01 ex ]$ rmdir c
[ andrew@pc01 ex ]$ ls -a
.  ..
```

Ya que **rmdir** solo puede remover directorios vacíos:

```
[ andrew@pc01 ex ]$ cd ..
[ andrew@pc01 ex ]$ ls test/
*.txt 0.txt 1.txt a a.txt b c
[ andrew@pc01 ~ ]$ rmdir test/
rmdir: failed to remove 'test/': Directory not empty
```

Pero pueden borrar un directorio, y todo su contenido con **rm -r** (**-r** recursivo).

Recursivo es una forma de indicar que la acción afectará a todo el contenido, se utiliza en todos los otros comandos que puedan afectar a múltiples archivos desde un directorio.

Con esta bandera ejerciten **CUÁDRUPLE** cuidado, porque esto potencialmente puede borrar mucho más allá de lo que esperan y de forma **no recuperable**.

```
[ andrew@pc01 ~ ]$ rm -r test
```

Moviendo y copiando archivos, creando enlaces y la historia de comandos

`mv` / `cp` / `ln`

`mv` mueve o renombra un archivo. Puede usar `mv` con un archivo para cambiar su directorio y mantener el mismo nombre de archivo o `mv` un archivo a un "nuevo archivo" (renombrándolo):

```
[ andrew@pc01 ex ]$ ls
a b c d
[ andrew@pc01 ex ]$ mv a e
[ andrew@pc01 ex ]$ ls
b c d e
```

`cp` copia archivos:

```
[ andrew@pc01 ex ]$ cp e e2
[ andrew@pc01 ex ]$ ls
b c d e e2
```

`ln` crea un enlace duro (hard link) a un archivo:


```
# el primer argumento a ln es el archivo a vincular y el segundo el destino.
[ andrew@pc01 ex ]$ ln b f
[ andrew@pc01 ex ]$ ls
b c d e e2 f
```

`ln -s` crea un enlace suave (soft link) a un archivo:

```
[ andrew@pc01 ex ]$ ln -s b g
[ andrew@pc01 ex ]$ ls
b c d e e2 f g
```


Los enlaces duros hacen referencia a los mismos bytes reales en la memoria que contienen un archivo, mientras que los enlaces flexibles se refieren al nombre del archivo original, que a su vez apunta a esos bytes. [Puede leer más sobre enlaces blandos frente a enlaces duros aquí.](#)

Historia de comandos

bash tiene dos características importantes para ayudarlo a completar y volver a ejecutar comandos, la primera es completar con tabulador. Simplemente escriba la primera parte de un comando, presione la tecla tab  y deje que la terminal adivine lo que está tratando de hacer:

```
[ andrew@pc01 dir ]$ ls <ENTER>
anotherlongfilename  thisisalongfilename  anewfilename

[ andrew@pc01 dir ]$ ls t <TAB>
```



... presione la tecla  después de escribir **ls t** y el comando se completa...

```
[ andrew@pc01 dir ]$ ls thisisalongfilename <ENTER>
thisisalongfilename
```



Tal vez tengas que presionar  varias veces si hay alguna ambigüedad:

```
[ andrew@pc01 dir ]$ ls a <TAB>

[ andrew@pc01 dir ]$ ls an <TAB>
anewfilename  anotherlongfilename
```

bash guarda la historia de los comandos previamente enviados y permite su búsqueda con la combinación de teclas **^r** ( + ):



```
[ andrew@pc01 dir ]
```

Presionamos **^r** ( + ) para buscar en el historial de comandos:

```
(reverse-i-search)`':
```

Escribimos 'anew' y vemos el ultimo comando que contenía esta palabra es encontrado.

```
(reverse-i-search)`anew': touch anewfilename
```

Podemos presionar nuevamente **^r** ( + ) para ir hacia la siguiente coincidencia.

Árboles de directorios, uso de disco y procesos

mkdir -p / tree

`mkdir`, por defecto, solo crea un único directorio. Esto significa que si, por ejemplo, el directorio `d/e` no existe, entonces `d/e/f` no se puede crear con `mkdir` por sí mismo:

```
[ andrew@pc01 ex ]$ ls
a  b  c
[ andrew@pc01 ex ]$ mkdir d/e/f
mkdir: no se puede crear el directorio «d/e/f»: No es un directorio
```

Pero si pasamos la bandera `-p` a `mkdir`, creará la ruta completa de directorios aunque no existan:

```
[ andrew@pc01 ex ]$ mkdir -p d/e/f
[ andrew@pc01 ex ]$ ls
a  b  c  d
```

`tree` puede ayudarlo a visualizar mejor la estructura de un directorio imprimiendo un árbol de directorio con un formato agradable. De forma predeterminada, imprime la estructura de árbol completa (comenzando con el directorio especificado), pero puede restringirlo a un cierto número de niveles con la bandera `-L`:

```
[ andrew@pc01 ex ]$ tree -L 2
.
|-- a
|-- b
|-- c
|-- d
    |-- e

3 directories, 2 files
```

Puede ocultar directorios vacíos en la salida de `tree` con `--prune`. Tenga en cuenta que esto también elimina directorios "recursivamente vacíos", o directorios que no están vacíos per se, pero que contienen solo otros directorios vacíos, u otros directorios recursivamente vacíos:

```
[ andrew@pc01 ex ]$ tree --prune
.
|-- a
|-- b
```

df / du / ps

df se usa para mostrar cuánto espacio ocupan los archivos para los discos o su sistema [discos duros, etc.].

```
[ andrew@pc01 ex ]$ df -h
S.ficheros          Tamaño Usados  Disp Uso% Montado en
udev                126G      0    126G   0% /dev
tmpfs               26G    2.0G    24G   8% /run
/dev/mapper/ubuntu--vg-root 1.6T  1.3T   252G  84% /
```

De esta lista, pueden ignorar los sistemas de archivo tmpfs, son sistemas de archivos que residen en memoria y no son permanentes. Son usados como espacio de intercambio con programas tratando una porción de la memoria como un archivo.

En el comando anterior, **-h** no significa "ayuda", sino "legible por humanos". Algunos comandos usan esta convención para mostrar tamaños de archivos / discos con **K** para kilobytes, **G** para gigabytes, etc., en lugar de escribir un número entero gigantesco de bytes.

du muestra el uso del espacio de archivos para un directorio en particular y sus subdirectorios. Si desea saber cuánto espacio hay libre en un disco duro determinado, use **df**; si desea saber cuánto espacio ocupa un directorio, use **du**:

```
[ andrew@pc01 ex ]$ du
4      ./d/e/f
8      ./d/e
12     ./d
4      ./c
20     .
```

du toma un indicador **--max-depth = N**, que solo muestra directorios **N** niveles hacia abajo [o menos] desde el directorio especificado:

```
[ andrew@pc01 ex ]$ du -h --max-depth=1
12K    ./d
4.0K   ./c
20K    .
```

ps muestra todos los procesos actualmente en ejecución del usuario [también conocidos como trabajos o jobs]:

```
[ andrew@pc01 ex ]$ ps
```

PID	TTY	TIME	CMD
16642	pts/15	00:00:00	ps
25409	pts/15	00:00:00	bash

Misceláneos

passwd / logout / exit

Cambie la contraseña de su cuenta con *passwd*. Le pedirá su contraseña actual para la verificación, luego le pedirá que ingrese la nueva contraseña dos veces, para que no cometa ningún error tipográfico:

```
[ andrew@pc01 dir ]$ passwd
Changing password for andrew.
(current) UNIX password:    <type current password>
Enter new UNIX password:    <type new password>
Retype new UNIX password:   <type new password again>
passwd: password updated successfully
```

logout sale de un shell en el que ha iniciado sesión [donde tiene una cuenta de usuario]:

```
[ andrew@pc01 dir ]$ logout

-
Session stopped
- Press <return> to exit tab
- Press R to restart session
- Press S to save terminal output to file
```

exit sale de cualquier tipo de shell:

```
[ andrew@pc01 ~ ]$ exit
logout

-
Session stopped
- Press <return> to exit tab
- Press R to restart session
- Press S to save terminal output to file
```

`clear` / *

Ejecute `clear` para mover la línea terminal actual a la parte superior de la pantalla. Este comando simplemente agrega líneas en blanco debajo de la línea de solicitud actual. Es bueno para limpiar su espacio de trabajo.

Utilice el glob `*`, también conocido como Kleene Star, también conocido como comodín] cuando busque archivos. Note la diferencia entre los siguientes dos comandos:

```
[ andrew@pc01 ~ ]$ ls Git/Parser/source/  
PArrayUtils.java      PFile.java            PSQLFile.java         PWatchman.java  
PDateTimeUtils.java   PFixedWidthFile.java PStringUtils.java     PXSVMFile.java  
PDelimitedFile.java   PNode.java            PTextFile.java        Parser.java  
  
[ andrew@pc01 ~ ]$ ls Git/Parser/source/PD*  
Git/Parser/source/PDateTimeUtils.java  Git/Parser/source/PDelimitedFile.java
```

El glob se puede usar varias veces en un comando y coincide con cero o más caracteres:

```
[ andrew@pc01 ~ ]$ ls Git/Parser/source/P*D*m*  
Git/Parser/source/PDateTimeUtils.java  Git/Parser/source/PDelimitedFile.java
```

Intermedios

Uso de disco, memoria y procesador

Acá empezamos por algunos comandos que tal vez no estén instalados y tal vez requieran que los agreguemos. Esto es simple en Linux pero si están usando algún terminal emulado u otro sistema operativo tal vez no esté disponible.

ncdu

`ncdu` [NCurses Disk Usage] proporciona una descripción general navegable del uso del espacio de archivos y directorios, como un `du` pero más detallado. Abre una ventana similar a `vim` de solo lectura (presione `q` para salir):

```
[ andrew@pc01 ~ ]$ ncdu  
  
ncdu 1.11 ~ Use the arrow keys to navigate, press ? for help
```

```
----- /home/andrew
```

```
-----
148.2 MiB [#####] /.m2
 91.5 MiB [#####] /.sbt
 79.8 MiB [#####] /.cache
 64.9 MiB [#####] /.ivy2
 40.6 MiB [##] /.sdkman
 30.2 MiB [##] /.local
 27.4 MiB [#] /.mozilla
 24.4 MiB [#] /.nanobackups
 10.2 MiB [ ] .confout3.txt
  8.4 MiB [ ] /.config
  5.9 MiB [ ] /.nbi
  5.8 MiB [ ] /.oh-my-zsh
  4.3 MiB [ ] /Git
  3.7 MiB [ ] /.myshell
  1.7 MiB [ ] /jdoc
  1.5 MiB [ ] .confout2.txt
  1.5 MiB [ ] /.netbeans
  1.1 MiB [ ] /.jenv
564.0 KiB [ ] /.rstudio-desktop
Total disk usage: 552.7 MiB Apparent size: 523.6 MiB Items: 14618
```

top / htop

top muestra todos los procesos actualmente en ejecución y sus propietarios, uso de memoria y más. **htop** es un **top** con interactividad mejorada. [Nota: puede pasar la bandera **-u usuario** para mostrar los procesos de un determinado usuario.].

```
[ andrew@pc01 ~ ]$ htop
```

```

 1 [ 0.0%]  9 [ 0.0%] 17 [ 0.0%] 25 [ 0.0%]
 2 [ 0.0%] 10 [ 0.0%] 18 [ 0.0%] 26 [ 0.0%]
 3 [ 0.0%] 11 [ 0.0%] 19 [ 0.0%] 27 [ 0.0%]
 4 [ 0.0%] 12 [ 0.0%] 20 [ 0.0%] 28 [ 0.0%]
 5 [ 0.0%] 13 [ 0.0%] 21 [ 1.3%] 29 [ 0.0%]
 6 [ 0.0%] 14 [ 0.0%] 22 [ 0.0%] 30 [ 0.6%]
 7 [ 0.0%] 15 [ 0.0%] 23 [ 0.0%] 31 [ 0.0%]
 8 [ 0.0%] 16 [ 0.0%] 24 [ 0.0%] 32 [ 0.0%]
Mem[|||||] 1.42G/252G Tasks: 188, 366 thr; 1 running
Swp[|] 2.47G/256G Load average: 0.00 0.00 0.00
Uptime: 432 days(!), 00:03:55
```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
9389	andrew	20	0	23344	3848	2848	R	1.3	0.0	0:00.10	htop
10103	root	20	0	3216M	17896	2444	S	0.7	0.0	5h48:56	/usr/bin/dockerd
1	root	20	0	181M	4604	2972	S	0.0	0.0	15:29.66	/lib/systemd/syst
533	root	20	0	44676	6908	6716	S	0.0	0.0	11:19.77	/lib/systemd/syst

```

 546 root      20    0  244M      0      S  0.0  0.0  0:01.39 /sbin/lvmetad -f
1526 root      20    0  329M    2252   1916 S  0.0  0.0  0:00.00
/usr/sbin/ModemMa
1544 root      20    0  329M    2252   1916 S  0.0  0.0  0:00.06
/usr/sbin/ModemMa
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill
F10Quit

```

Versiones de software

-version / --version / -v

La mayoría de los comandos y programas tienen una marca `-version` o `--version` que proporciona la versión de software de ese comando o programa. La mayoría de las aplicaciones hacen que esta información esté fácilmente disponible:

```

[ andrew@pc01 ~ ]$ ls --version
ls (GNU coreutils) 8.25 ...

[ andrew@pc01 ~ ]$ ncdu -version
ncdu 1.11

[ andrew@pc01 ~ ]$ python --version
Python 3.5.2

```

... pero algunas son menos intuitivas:

```

[ andrew@pc01 ~ ]$ sbt scalaVersion
...
[info] 2.12.4

```

tenga en cuenta que algunos programas usan `-v` como indicador de versión, mientras que otros usan `-v` para significar "detallado" (verbose), que ejecutará la aplicación mientras imprime muchos diagnósticos o información de depuración:

```

SCP(1)                                BSD General Commands Manual                                SCP(1)

NAME
    scp -- secure copy (remote file copy program)
...
-v      Verbose mode. Causes scp and ssh(1) to print debugging messages
        about their progress. This is helpful in debugging connection,
        authentication, and configuration problems.
...

```

Variables de entorno y alias

Variables de entorno

Las variables de entorno [a veces abreviadas como "env vars"] son variables persistentes que se pueden crear y usar dentro de su shell `bash`. Se definen con un signo igual (=) y se utilizan con un signo de dólar (\$). Puede ver todas las variables de entorno definidas actualmente con `printenv`:

```
[ andrew@pc01 ~ ]$ printenv
SPARK_HOME=/usr/local/spark
TERM=xterm
...
```

Establezca una nueva variable de entorno con un signo = (¡sin embargo, no ponga ningún espacio antes o después de =!):

```
[ andrew@pc01 ~ ]$ myvar=hello
```

Imprima una variable de entorno específica en la terminal con `echo` y un signo \$ precedente:

```
[ andrew@pc01 ~ ]$ echo $myvar
hello
```

Las variables de entorno que contienen espacios u otros espacios en blanco deben estar entre comillas ["..."]. Tenga en cuenta que reasignar un valor a una var env lo sobrescribe sin previo aviso:

```
[ andrew@pc01 ~ ]$ myvar="hello, world!"
[ andrew@pc01 ~ ]$ echo $myvar
hello, world!
```

Las variables de entorno también se pueden definir mediante el comando `export`. Cuando se definen de esta manera, también estarán disponibles para los subprocesos (comandos llamados desde este shell):

```
[ andrew@pc01 ~ ]$ export myvar="another one"
[ andrew@pc01 ~ ]$ echo $myvar
another one
```

Puede desarmar una variable de entorno dejando el lado derecho de `=` en blanco o usando el comando `unset`:

```
[ andrew@pc01 ~ ]$ unset mynewvar  
[ andrew@pc01 ~ ]$ echo $mynewvar
```

Alias

Los *alias* son similares a las variables de entorno, pero generalmente se usan de una manera diferente, para abreviar comandos largos con comandos versiones más cortas:

```
[ andrew@pc01 apidocs ]$ ls -l -a -h -t  
total 220K  
drwxr-xr-x 5 andrew andrew 4.0K Dec 21 12:37 .  
-rw-r--r-- 1 andrew andrew 9.9K Dec 21 12:37 help-doc.html  
-rw-r--r-- 1 andrew andrew 4.5K Dec 21 12:37 script.js  
...  
  
[ andrew@pc01 apidocs ]$ alias lc="ls -l -a -h -t"  
  
[ andrew@pc01 apidocs ]$ lc  
total 220K  
drwxr-xr-x 5 andrew andrew 4.0K Dec 21 12:37 .  
-rw-r--r-- 1 andrew andrew 9.9K Dec 21 12:37 help-doc.html  
-rw-r--r-- 1 andrew andrew 4.5K Dec 21 12:37 script.js  
...
```

Puede eliminar un alias con `unalias`:

```
[ andrew@pc01 apidocs ]$ unalias lc  
  
[ andrew@pc01 apidocs ]$ lc  
The program 'lc' is currently not installed. ...
```

Extra

[Lea acerca de las sutiles diferencias entre las variables de entorno y los alias aquí.](#)

[Algunos programas, como git, le permiten definir alias específicamente para ese software.](#)

Scripts Bash basicos

Scripts bash

Los scripts `bash` [normalmente terminados en `.sh`] le permiten automatizar procesos complicados, empaquetándolos en funciones reutilizables. Un script `bash` puede contener cualquier número de comandos de shell normales: (Con atención a la segunda y tercera redirección para no reemplazar la primera)

```
[ andrew@pc01 ~ ]$ echo "ls" > ex.sh
[ andrew@pc01 ~ ]$ echo "touch file" >> ex.sh
[ andrew@pc01 ~ ]$ "ls" >> ex.sh
```

Se puede ejecutar un script de shell con el comando `source` o el comando `sh`:

```
[ andrew@pc01 ~ ]$ source ex.sh
Desktop  Git  TEST  c  ex.sh  project  test
Desktop  Git  TEST  c  ex.sh  file  project  test
```

Los scripts de shell se pueden hacer ejecutables con el comando `chmod` (más sobre esto más adelante):

```
[ andrew@pc01 ~ ]$ echo "ls" > ex2.sh
[ andrew@pc01 ~ ]$ echo "touch file2" >> ex2.sh
[ andrew@pc01 ~ ]$ echo "ls" >> ex2.sh

[ andrew@pc01 ~ ]$ chmod +x ex2.sh
```

Se puede ejecutar un script de shell ejecutable precediéndolo con `./`:

```
[ andrew@pc01 ~ ]$ ./ex2.sh
Desktop  Git  TEST  c  ex.sh  ex2.sh  file  project  test
Desktop  Git  TEST  c  ex.sh  ex2.sh  file  file2  project  test
```

Las líneas largas de código se pueden dividir terminando un comando con `\`:

```
[ andrew@pc01 ~ ]$ echo "for i in {1..3}; do echo \
> \"Welcome \"$i times\"; done" > ex3.sh
```

Los scripts de bash pueden contener bucles, funciones y más.

```
[ andrew@pc01 ~ ]$ source ex3.sh
Welcome 1 times
```

```
Welcome 2 times  
Welcome 3 times
```

El mejor uso para empezar, es agrupar secuencias de comandos para ejecutarlos uno detrás de otro. Tengan en cuenta que es un lenguaje de programación completo y pueden gestionar los errores de todos los comandos que ejecuten.

Hashbang - #!

Todos los archivos de texto pueden ser ejecutables, pero es necesario indicar que programa será el encargado de ejecutarlo; esto se logra con la línea hashbang `#!/`, un comentario en la primera línea del script que hace esta indicación.

```
[ andrew@pc01 ~ ]$ cat script.sh  
#!/bin/bash  
ls  
touch file
```

Esta indicación se puede usar para cualquier otro tipo de script, como Python, AWK, Perl y demás.

Es posible indicar cualquier programa, incluso `rm`, haciendo que el script no solo no haga nada, sino que se autodestruya.

Extra

Las secuencias de comandos bash pueden hacer su vida mucho más fácil y colorida. Echa un vistazo a este fantástico [machete sobre scripts de bash](#).

Substitución de comandos

La substitución de comandos permite utilizar la salida de un programa como parte de los argumentos de otro, pudiendo hacer cosas como crear carpetas con la fecha actual.

Prompt personalizado y ls

`$PS1` [Prompt String 1] es la variable de entorno que define su indicador de shell principal:

```
[ andrew@pc01 ~ ]$ printf "%q" $PS1  
$'\n\n\n[\E[1m\n\n]\n\n[\E[30m\n\n]\n\nA'$'\n\n[\E[37m\n\n])\n\n[\E[36m\n\n]\n\nu\n\n[\E[37m\n\n])\n\n@\n\n[\E[34m\n\n])\n\nh'$'\n\n[\E[32m\n\n])\n\nW\n\n[\E[37m\n\n])\n\n|$'$'\n\n[\E(B\n\n[\E[m\n\n])'
```

Puede cambiar su indicador predeterminado con el comando de exportación:

```
[ andrew@pc01 ~ ]$ export PS1="\ninstrucción aquí> "  
  
instrucción aquí> echo $PS1  
\ninstrucción aquí>
```

... ¡[también podés agregar colores](#) !:

También puede cambiar los colores mostrados por `ls` editando la variable de entorno `$LS_COLORS`:

```
# (son colores que no aparecen en el documento)  
CODE: ls  
Desktop  Git    TEST  c    ex.sh  ex2.sh  ex3.sh  file  file2  project  test  
  
CODE: export LS_COLORS='di=31:fi=0:ln=96:or=31:mi=31:ex=92'  
  
CODE: ls  
Desktop  Git    TEST  c    ex.sh  ex2.sh  ex3.sh  file  file2  project  test
```

Extra

Un [constructor gráfico de PS1](#), con ejemplos:

Archivos de configuración

Archivos de configuración / `.bashrc`

Si probó los comandos de la última sección y cerró la sesión y volvió a iniciarla, es posible que haya notado que los cambios desaparecieron. Los archivos de *configuración* (config) le permiten mantener la configuración de su shell o de un programa en particular cada vez que inicia sesión (o ejecuta ese programa). El archivo de configuración principal para un shell `bash` es el archivo `~/.bashrc`. Los alias, las variables de entorno y las funciones agregadas a `~/.bashrc` estarán disponibles cada vez que inicie sesión. Los comandos en `~/.bashrc` se ejecutarán cada vez que inicie sesión.

Si edita su archivo `~/.bashrc`, puede volver a cargarlo sin cerrar sesión usando el comando de origen:

```
[ andrew@pc01 ~ ]$ nano ~/.bashrc
```

... agregue la línea `echo "~ / .bashrc cargado!"` al principio del archivo ...

```
[ andrew@pc01 ~ ]$ source ~/.bashrc
~/.bashrc cargado!
```

... cerrar sesión y volver a iniciar sesión ...

```
Last login: Fri Jan 11 10:29:07 2019 from 111.11.11.111
~/.bashrc cargado!

[ andrew@pc01 ~ ]
```

Tipos de shells

Los shells de inicio de sesión son shells en los que inicia sesión (donde tiene un nombre de usuario). Los shells interactivos son shells que aceptan comandos. Los shells pueden ser de inicio de sesión e interactivos, sin inicio de sesión y no interactivos, o cualquier otra combinación.

Además de `~/.bashrc`, hay algunos otros scripts que el shell referencia automáticamente cuando inicia o cierra sesión. Estos son:

- `/etc/profile`
- `~/`
- `.bash_profile`
- `~/.bash_login`
- `~/.profile`
- `~/.bash_logout`
- `/etc/bash.bash_logout`

Cuáles de estos scripts se obtienen y el orden en el que se obtienen, dependen del tipo de shell abierto. Consulte la [página de manual de bash](#) y [estas](#) publicaciones de [Stack Overflow](#) para obtener más información.

Tenga en cuenta que los scripts de `bash` pueden llamar a otros scripts. Por ejemplo, en su `~/.bashrc`, podría incluir la línea:

```
source ~/.bashrc_addl
```

... que también llamaría a ese script `.bashrc_addl`. Este archivo puede contener sus propios alias, funciones, variables de entorno, etc. A su vez, también podría obtener otros scripts. ¡Tenga cuidado de evitar bucles infinitos de fuente de scripts!

Puede resultar útil dividir los comandos en diferentes scripts de shell según la funcionalidad o el tipo de máquina (Ubuntu frente a Red Hat frente a macOS), por ejemplo:

- `~/.bash_ubuntu`: configuración específica para máquinas basadas en Ubuntu
- `~/.bashrc_styles`: configuraciones estéticas, como `PS1` y `LS_COLORS`
- `~/.bash_java`: configuración específica del lenguaje Java

Intento mantener archivos `bash` separados para configuraciones estéticas y código específico del sistema operativo o de la máquina, y luego tengo un archivo `bash` grande que contiene accesos directos, etc., que uso en cada máquina y en cada sistema operativo.

Tenga en cuenta que también hay diferentes conchas. `bash` es solo un tipo de shell (el "Bourne Again Shell"). Otros comunes incluyen `zsh`, `csh`, `fish` y más. Juegue con diferentes shells y encuentre uno que sea adecuado para usted, pero tenga en cuenta que este tutorial solo contiene comandos de shell de `bash` y que no todo lo que se enumera aquí (tal vez ninguno de ellos) será aplicable a shells que no sean `bash`.

Encontrando cosas

whereis / which / whatis

`whereis` busca archivos "posiblemente útiles" relacionados con un comando en particular. Intentará devolver la ubicación del binario (código de máquina ejecutable), la fuente (archivos de código fuente) y la página de manual de ese comando:

```
[ andrew@pc01 ~ ]$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

`which` que solo devolverá la ubicación del binario (el comando en sí):

```
[ andrew@pc01 ~ ]$ which ls
/bin/ls
```

`whatis` imprime la descripción de una línea de un comando desde su página de manual:

```
[ andrew@pc01 ~ ]$ whatis whereis which whatis
whereis (1)          - locate the binary, source, and manual page files for a
command
which (1)            - locate a command
whatis (1)           - display one-line manual page descriptions
```

which que es útil para encontrar la "versión original" de un comando que puede estar oculto por un alias:

```
[ andrew@pc01 ~ ]$ alias ls="ls -l"

# El ls "original" ha sido escondido por el alias anterior
[ andrew@pc01 ~ ]$ ls
total 36
drwxr-xr-x 2 andrew andrew 4096 Jan  9 14:47 Desktop
drwxr-xr-x 4 andrew andrew 4096 Dec  6 10:43 Git
...

# pero aún podemos llamar al ls "original" usando
# la ubicación devuelta por which
[ andrew@pc01 ~ ]$ /bin/ls
Desktop  Git  TEST  c  ex.sh  ex2.sh  ex3.sh  file  file2  project  test
```

locate / find

locate encuentra un archivo en cualquier lugar del sistema consultando una lista de archivos en caché que se actualiza con regularidad:

```
[ andrew@pc01 ~ ]$ locate README.md
/home/andrew/.config/micro/plugins/gotham-colors/README.md
/home/andrew/.jenv/README.md
/home/andrew/.myshell/README.md
...
```

Debido a que solo se busca en una lista, **locate** suele ser más rápido que la alternativa, buscar. **find** recorre el sistema de archivos para encontrar el archivo que está buscando. Sin embargo, debido a que en realidad está mirando los archivos que existen actualmente en el sistema, siempre devolverá una lista actualizada de archivos, lo que no es necesariamente cierto con **locate**.

```
[ andrew@pc01 ~ ]$ find ~/ -iname "README.md"
/home/andrew/.jenv/README.md
/home/andrew/.config/micro/plugins/gotham-colors/README.md
/home/andrew/.oh-my-zsh/plugins/ant/README.md
...
```

`find` fue escrito para la primera versión de Unix en 1971 y, por lo tanto, está mucho más disponible que `locate`, que se agregó a GNU en 1994.

`find` tiene muchas más funciones que `locate`, y puede buscar por antigüedad del archivo, tamaño, propiedad, tipo, marca de tiempo, permisos, profundidad dentro del sistema de archivos; `find` puede buscar usando expresiones regulares, ejecutar comandos en archivos que encuentra y más.

Cuando necesite una lista rápida (pero posiblemente desactualizada) de archivos, o no esté seguro de en qué directorio se encuentra un archivo en particular, use `locate`. Cuando necesite una lista de archivos precisa, tal vez basada en algo diferente a los nombres de los archivos, y necesite hacer algo con esos archivos, use `find`.

Descargando cosas

`ping` / `wget` / `curl`

`ping` envía un mensaje de control [ICMP] hacia otra dirección de red para medir el tiempo de respuesta, se usa entre otras cosas, para saber si el equipo en la dirección indicada esta funcionando, y para hacer diagnósticos de la conexión.

```
[ andrew@pc01 ~ ]$ ping google.com
PING google.com (74.125.193.100) 56(84) bytes of data.
Pinging 74.125.193.100 with 32 bytes of data:
Reply from 74.125.193.100: bytes=32 time<1ms TTL=64
...
```

`wget` lo pueden usar para descargar archivos de internet usando su URL.

```
[ andrew@pc01 ~ ]$ wget \
> http://releases.ubuntu.com/18.10/ubuntu-18.10-desktop-amd64.iso
```

Cuando el archivo es grande, podemos indicarle a `wget` con la bandera `-c` / `--continue` para que guarde información de recuperación de la descarga, para recuperar la descarga, ejecuten el mismo comando y bandera.

```
[ andrew@pc01 ~ ]$ wget -c \
> http://releases.ubuntu.com/18.10/ubuntu-18.10-desktop-amd64.iso
```

`curl` puede ser usado de la misma manera que `wget`, pero no tenemos que olvidar la bandera `--output`:

```
[ andrew@pc01 ~ ]$ curl \  
> http://releases.ubuntu.com/18.10/ubuntu-18.10-desktop-amd64.iso \  
> --output ubuntu.iso
```

`curl` y `wget` tienen sus propias fortalezas y debilidades. `curl` admite muchos más protocolos y está más disponible que `wget`; `curl` también puede enviar datos, mientras que `wget` solo puede recibir datos. `wget` puede descargar archivos de forma recursiva, mientras que `curl` no.

En general, uso `wget` cuando necesito descargar cosas de Internet. No suelo enviar datos mediante `curl`, pero es bueno tenerlo en cuenta en las raras ocasiones en que lo hace.

apt / gunzip / tar / gzip

Las distribuciones de Linux descendientes de Debian tienen una fantástica herramienta de administración de paquetes llamada `apt`. Puede usarse para instalar, actualizar o eliminar software en su máquina. Para buscar `apt` para una pieza de software en particular, use `apt search` e instálelo con `apt install`:

```
[ andrew@pc01 ~ ]$ apt search bleachbit  
...bleachbit/bionic,bionic 2.0-2 all  
  delete unnecessary files from the system  
  
# Va a ser necesario usar 'sudo' para instalar software  
[ andrew@pc01 ~ ]$ sudo apt install bleachbit
```

El software de Linux a menudo viene empaquetado en archivos `.tar.gz` ("tarball"):

```
[ andrew@pc01 ~ ]$ wget \  
> https://github.com/atom/atom/releases/download/v1.35.0-beta0/atom-  
amd64.tar.gz
```

... estos tipos de archivos se pueden descomprimir con `gunzip`:

```
[ andrew@pc01 ~ ]$ gunzip atom-amd64.tar.gz  
[ andrew@pc01 ~ ]$ ls  
atom-amd64.tar
```

Un archivo `.tar.gz` se convertirá en un archivo `.tar`, que se puede extraer a un directorio de archivos usando `tar -xzf` (`-x` para "extraer", `-f` para especificar el archivo a "untar"):


```
[ andrew@pc01 ~ ]$ tar -xf atom-amd64.tar
[ andrew@pc01 ~ ]$ mv atom-beta-1.35.0-beta0-amd64 atom
[ andrew@pc01 ~ ]$ ls
atom atom-amd64.tar
```

Para ir en la dirección inversa, puede crear `[-c]` un archivo `tar` desde un directorio y comprimirlo (o descomprimirlo, según corresponda) con `-z`:

```
[ andrew@pc01 ~ ]$ tar -zcf compressed.tar.gz atom
[ andrew@pc01 ~ ]$ ls
atom atom-amd64.tar compressed.tar.gz
```

Los archivos `.tar` también se pueden comprimir con `gzip`:

```
[ andrew@pc01 ~ ]$ gzip atom-amd64.tar
[ andrew@pc01 ~ ]$ ls
atom atom-amd64.tar.gz compressed.tar.gz
```

Redirigiendo la Entrada y Salida

`|` `>` `<` `echo` `printf`

De forma predeterminada, los comandos de shell leen su entrada del flujo de entrada estándar (también conocido como `stdin` o `0`) y escriben en el flujo de salida estándar (también conocido como `stdout` o `1`), a menos que haya un error, que se escribe en el flujo de error estándar (también conocido como `.stderr` o `2`).

`echo` escribe texto en `stdout` de forma predeterminada, que en la mayoría de los casos simplemente lo imprimirá en la terminal:

```
[ andrew@pc01 ~ ]$ echo "hello"
hello
```

El operador de tubería/caño/pipe, `|`, redirige la salida del primer comando a la entrada del segundo comando:

```
[ andrew@pc01 ~ ]$ echo "example document" | wc
      1      2     17
```

`>` redirige la salida de `stdout` a un archivo en particular

```
[ andrew@pc01 ~ ]$ echo "test" > file
[ andrew@pc01 ~ ]$ head file
test
```

`printf` es un `echo` mejorado, que permite formatear y escapar secuencias:

```
[ andrew@pc01 ~ ]$ printf "1\n3\n2"
1
3
2
```

`<` obtiene la entrada de un archivo en particular, en lugar de `stdin`:

```
[ andrew@pc01 ~ ]$ sort <(printf "1\n3\n2")
1
2
3
```

En lugar de un UUOC, la forma recomendada de enviar el contenido de un archivo a un comando es usar `<`. Tenga en cuenta que esto hace que los datos "fluyan" de derecha a izquierda en la línea de comando, en lugar de (lo que quizás sea más natural, para los angloparlantes) de izquierda a derecha:

```
[ andrew@pc01 ~ ]$ printf "1\n3\n2" > file
[ andrew@pc01 ~ ]$ sort < file
1
2
3
```

0 / 1 / 2 / tee

0, 1 y 2 son los flujos estándar de entrada, salida y error, respectivamente. Los flujos de entrada y salida se pueden redirigir con los operadores `/`, `>` y `<` mencionados anteriormente, pero `stdin`, `stdout` y `stderr` también se pueden manipular directamente usando sus identificadores numéricos:

Escriba en `stdout` o `stderr` con `>&1` o `>&2`:

```
[ andrew@pc01 ~ ]$ cat test
echo "stdout" >&1
echo "stderr" >&2
```

De forma predeterminada, `stdout` y `stderr` imprimen la salida al terminal:

```
[ andrew@pc01 ~ ]$ ./test  
stderr  
stdout
```

Redirigir stdout a `/dev/null` (solo la salida de impresión enviada a stderr):

```
[ andrew@pc01 ~ ]$ ./test 1>/dev/null  
stderr
```

Redirigir stderr a `/dev/null` (solo la salida de impresión se envía a stdout):

```
[ andrew@pc01 ~ ]$ ./test 2>/dev/null  
stdout
```

Redirigir toda la salida a `/dev/null` (no imprimir nada):

```
[ andrew@pc01 ~ ]$ ./test &>/dev/null
```

Envíe la salida a la salida estándar y a cualquier número de ubicaciones adicionales con `tee`:

```
[ andrew@pc01 ~ ]$ ls  
file0  
[ andrew@pc01 ~ ]$ echo "test" | tee file1 file2 file3  
test  
[ andrew@pc01 ~ ]$ ls  
file0 file1 file2 file3
```

mktemp

Crea un fichero o un directorio temporal, de forma segura, y muestra su nombre. Este comando es ideal para scripts en los que sea necesario descargar algo una sola vez para procesarlo varias veces. Los archivos temporarios se crean dentro del directorio `/tmp/` y son borrados en cada reinicio.

```
[ rodrigo@pc01 ~ ]$ mktemp  
/tmp/tmp.cd7GPV7uGX
```

La mejor forma de utilizar este comando, es utilizando la [substitución de comandos](#) para asignarlo a una variable y utilizarlo de manera repetida.

```
[ rodrigo@pc01 ~ ]$ archivo=$(mktemp)
```

```
[ rodrigo@pc01 ~ ]$ echo $archivo
/tmp/tmp.cd7GPV7uGX
[ rodrigo@pc01 ~ ]$ wget -q
https://raw.githubusercontent.com/martinvilu/zotero-manual.github.io/master/
introduction.md -O $archivo
[ rodrigo@pc01 ~ ]$ less $archivo
```

No olviden borrar el archivo temporario luego de usarlo.

```
[ rodrigo@pc01 ~ ]$ rm $archivo
```

Aunque los archivos creados en `/tmp/` se borran periódicamente, sean prolijos y limpienlo.

Aporte por Rodrigo Locatti

Avanzado

Superusuario

sudo / su

Puedes comprobar cuál es tu nombre de usuario con whoami:

```
[ andrew@pc01 abc ]$ whoami
andrew
```

... y ejecute un comando como otro usuario con `sudo -u username` (necesitará la contraseña de ese usuario):

```
[ andrew@pc01 abc ]$ sudo -u test touch def
[ andrew@pc01 abc ]$ ls -l
total 0
-rw-r--r-- 1 test test 0 Jan 11 20:05 def
```

Si no se proporciona `-u`, el usuario predeterminado es el superusuario (generalmente llamado "root"), con permisos ilimitados:

```
[ andrew@pc01 abc ]$ sudo touch ghi
[ andrew@pc01 abc ]$ ls -l
total 0
-rw-r--r-- 1 test test 0 Jan 11 20:05 def
```

```
-rw-r--r-- 1 root root 0 Jan 11 20:14 ghi
```

Utilice **su** para convertirse en otro usuario temporalmente (y **exit** para volver):

```
[ andrew@pc01 abc ]$ su test
Password:
test@pc01:/home/andrew/abc$ whoami
test
test@pc01:/home/andrew/abc$ exit
exit

[ andrew@pc01 abc ]$ whoami
andrew
```

[Podés aprender más sobre las diferencias entre su y sudo acá](#)

!!

El superusuario (normalmente "root") es la única persona que puede instalar software, crear usuarios, etc. A veces es fácil olvidarlo y puede aparecer un error:

```
[ andrew@pc01 ~ ]$ apt install ruby
```

```
E: Could not open lock file /var/lib/dpkg/lock-frontend - open (13:
Permission denied)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend),
are you root?
```

Puede volver a escribir el comando y agregar **sudo** al frente (ejecutarlo como superusuario):

```
[ andrew@pc01 ~ ]$ sudo apt install ruby
Reading package lists...
```

¡O puede usar el atajo **!!**, que conserva el comando anterior:

```
[ andrew@pc01 ~ ]$ apt install ruby
E: Could not open lock file /var/lib/dpkg/lock-frontend - open (13:
Permission denied)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend),
are you root?

[ andrew@pc01 ~ ]$ sudo !!
sudo apt install ruby
Reading package lists...
```

De forma predeterminada, ejecutar un comando con **sudo** (e ingresar correctamente la contraseña) permite al usuario ejecutar comandos de superusuario durante los próximos 15

minutos. Una vez transcurridos esos 15 minutos, se le solicitará nuevamente al usuario que ingrese la contraseña de superusuario si intenta ejecutar un comando restringido.

Permisos de archivos

Permisos de archivo

Los archivos pueden ser leídos (*r*), escritos en (*w*) y / o ejecutados (*x*) por diferentes usuarios o grupos de usuarios, o no pueden ser leídos en absoluto. Los permisos de archivo se pueden ver con el comando `ls -l` y están representados por 10 caracteres:

```
[ andrew@pc01 ~ ]$ ls -lh
total 8
drwxr-xr-x 4 andrew andrew 4.0K Jan  4 19:37 tast
-rwxr-xr-x 1 andrew andrew  40 Jan 11 16:16 test
-rw-r--r-- 1 andrew andrew   0 Jan 11 16:34 tist
```

El primer carácter de cada línea representa el tipo de archivo (*d* = directorio, *l* = enlace, *-* = archivo normal, etc.); luego hay tres grupos de tres caracteres que representan los permisos que posee el usuario (*u*) que posee el archivo, los permisos que posee el grupo (*g*) que posee el archivo y los permisos que posee cualquier otro (*o*) usuario. (El número que sigue a esta cadena de caracteres es el número de enlaces en el sistema de archivos a ese archivo [4 o 1 arriba].)

r significa que la persona / esas personas tienen permiso de lectura, *w* es permiso de escritura, *x* es permiso de ejecución. Si un directorio es "ejecutable", eso significa que se puede abrir y su contenido se puede enumerar. Estos tres permisos a menudo se representan con un solo número de tres dígitos, donde, si *x* está habilitado, el número se incrementa en 1, si *w* está habilitado, el número se incrementa en 2, y si *r* está habilitado, el número se incrementa por 4. Tenga en cuenta que son equivalentes a dígitos binarios (*r-x* -> 101 -> 5, por ejemplo). Entonces, los tres archivos anteriores tienen permisos de 755, 755 y 644, respectivamente.

Las siguientes dos cadenas de cada lista son el nombre del propietario (*andrew*, en este caso) y el grupo del propietario (también *andrew*, en este caso). Luego viene el tamaño del archivo, su fecha de modificación más reciente y su nombre. El indicador *-h* hace que la salida sea legible por humanos (es decir, imprime 4.0K en lugar de 4096 bytes).

chmod / chown

Los permisos de archivo se pueden modificar con `chmod` configurando los bits de acceso:

```
[ andrew@pc01 ~ ]$ chmod 777 test
[ andrew@pc01 ~ ]$ chmod 000 tist
[ andrew@pc01 ~ ]$ ls -lh
total 8.0K
drwxr-xr-x 4 andrew andrew 4.0K Jan  4 19:37 tast
-rwxrwxrwx 1 andrew andrew  40 Jan 11 16:16 test
----- 1 andrew andrew    0 Jan 11 16:34 tist
```

... o agregando (+) o eliminando (-) permisos **r**, **w** y **x** con banderas:

```
[ andrew@pc01 ~ ]$ chmod +rwx tist
[ andrew@pc01 ~ ]$ chmod -w test
[ andrew@pc01 ~ ]$ ls -lh
chmod: test: new permissions are r-xrwxrwx, not r-xr-xr-x
total 8.0K
drwxr-xr-x 4 andrew andrew 4.0K Jan  4 19:37 tast
-r-xrwxrwx 1 andrew andrew  40 Jan 11 16:16 test
-rwxr-xr-x 1 andrew andrew   0 Jan 11 16:34 tist
```

El usuario que posee un archivo se puede cambiar con **chown**:

```
[ andrew@pc01 ~ ]$ sudo chown marina test
```

El grupo que posee un archivo se puede cambiar con **chgrp**:

```
[ andrew@pc01 ~ ]$ sudo chgrp hadoop tist
[ andrew@pc01 ~ ]$ ls -lh
total 8.0K
drwxr-xr-x 4 andrew andrew 4.0K Jan  4 19:37 tast
-----w--w- 1 marina andrew  40 Jan 11 16:16 test
-rwxr-xr-x 1 andrew hadoop    0 Jan 11 16:34 tist
```

Gestión de usuarios y grupos

Usuarios

users muestra todos los usuarios que han iniciado sesión actualmente. Tenga en cuenta que un usuario puede iniciar sesión varias veces si, por ejemplo, están conectados a través de varias sesiones **ssh**.

```
[ andrew@pc01 ~ ]$ users
andrew colin colin colin colin colin kolina kolina
```

Para ver todos los usuarios (incluso los que no han iniciado sesión), consulte `/etc/passwd`. **[ADVERTENCIA: ¡no modifique este archivo!** Puede dañar sus cuentas de usuario y hacer que sea imposible iniciar sesión en su sistema). Para eso está `useradd` y `userdel`.

```
[ andrew@pc01 ~ ]$ alias au="cut -d: -f1 /etc/passwd > | sort | uniq"
[ andrew@pc01 ~ ]$ au
_apt
anaid
andrew...
```

Agregue un usuario con `useradd`:

```
[ andrew@pc01 ~ ]$ sudo useradd aardvark
[ andrew@pc01 ~ ]$ au
_apt
aardvark
anaid...
```

Eliminar un usuario con `userdel`:

```
[ andrew@pc01 ~ ]$ sudo userdel aardvark
[ andrew@pc01 ~ ]$ au
_apt
anaid
andrew...
```

[Cambie el shell, el nombre de usuario, la contraseña o la pertenencia a un grupo predeterminados de un usuario con `usermod`.](#)

Procesamiento de texto

`uniq` / `sort` / `diff` / `cmp`

`uniq` muestra o filtra las líneas repetidas en un archivo:

```
[ andrew@pc01 man ]$ cat ejemplo.txt
Mucha música
Mucha música
Mucha música

Más música
Más música

Fin.
[ andrew@pc01 man ]$ uniq ejemplo.txt
```



```
Mucha música
```

```
Más música
```

```
Fin.
```

```
[ andrew@pc01 man ]$ uniq -d ejemplo.txt #Muestra solo lo que se repite
```

```
Mucha música
```

```
Más música
```

sort ordenará las líneas alfabéticamente / numéricamente:

```
[ andrew@pc01 man ]$ sort b
```

```
1
```

```
2
```

```
3
```

diff informará qué líneas difieren entre dos archivos:

```
andrew@pc01 man ]$ diff a b
```

```
2c2
```

```
< 2
```

```
-----
```

```
> 3
```

cmp informa qué bytes difieren entre dos archivos:

```
[ andrew@pc01 man ]$ cmp a b
```

```
a b differ: char 3, line 2cut / sed
```

cut / sed

cut se usa generalmente para cortar una línea en secciones en algún delimitador (bueno para el procesamiento CSV). **-d** especifica el delimitador y **-f** especifica el índice del campo a imprimir (comenzando con 1 para el primer campo):

```
[ andrew@pc01 man ]$ printf "137.99.234.23" > c
```

```
[ andrew@pc01 man ]$ cut -d'.' c -f1
```

```
137
```

sed se usa comúnmente para reemplazar una cadena con otra cadena en un archivo:

```
[ andrew@pc01 man ]$ echo "old" | sed s/old/new/
```

```
new
```

... pero `sed` es una herramienta extremadamente poderosa y no se puede resumir adecuadamente aquí. En realidad, es Turing completo, por lo que puede hacer cualquier cosa que cualquier otro lenguaje de programación pueda hacer. `sed` puede buscar y reemplazar basándose en expresiones regulares, imprimir selectivamente líneas de un archivo que coincidan o contienen un patrón determinado, editar archivos de texto in situ y de forma no interactiva, y mucho más.

Algunos buenos tutoriales sobre sed incluyen:

- <https://www.tutorialspoint.com/sed/>
- <http://www.grymoire.com/Unix/Sed.html>
- <https://www.computerhope.com/unix/used.htm>
- <https://letsfindcourse.com/tutorials/sed-tutorials/sed-tutorial>

Búsqueda por patrones (pattern matching)

grep

El nombre `grep` proviene de `g/re/p` [search globally for a regular expression and print it, o busque globalmente una expresión regular e imprímala]; se utiliza para buscar texto en archivos.

`grep` se usa para encontrar líneas de un archivo que coincidan con algún patrón:

```
[ andrew@pc01 ~ ]$ grep -e ".*fi.*" /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# The file bash.bashrc already sets the default PS1.
fi
fi
...
```

O contienen alguna palabra

```
[ andrew@pc01 ~ ]$ grep "andrew" /etc/passwd
andrew:x:1000:1000:andrew,,,:/home/andrew:/bin/bash
```

`grep` suele ser la opción preferida para simplemente encontrar líneas coincidentes en un archivo, si está planeando permitir que otro programa maneje esas líneas (o si solo desea verlas).

`grep` permite `(-E)` el uso de expresiones regulares extendidas, `(-F)` hacer coincidir cualquiera de las múltiples cadenas a la vez y `(-r)` buscar archivos de forma recursiva dentro de un

directorio. Estos indicadores solían implementarse como comandos separados (*egrep*, *fgrep* y *rgrep*, respectivamente), pero esos comandos ahora están en desuso.

Extra

[Vea los orígenes de los nombres de algunos comandos bash famosos.](#)

Copiando archivos via ssh

ssh / scp

ssh es la forma en que las máquinas basadas en Unix se conectan entre sí a través de una red:

```
[ andrew@pc01 ~ ]$ ssh -p <port> andrew@137.xxx.xxx.89
Last login: Fri Jan 11 12:30:52 2019 from 137.xxx.xxx.199
```

Observe que mi mensaje ha cambiado porque ahora estoy en una máquina diferente:

```
[ andrew@pc02 ~ ]$ exit
logout
Connection to 137.xxx.xxx.89 closed.
```

Cree un archivo en la máquina 1:

```
[ andrew@pc01 ~ ]$ echo "hello" > hello
```

Cópielo en la máquina 2 usando *scp* [copia segura; tenga en cuenta que *scp* usa *-P* para un número de puerto, *ssh* usa *-p*]

```
[ andrew@pc01 ~ ]$ scp -P <port> hello andrew@137.xxx.xxx.89:~
hello                               100%    0    0.0KB/s   00:00
```

ssh en la máquina 2:

```
[ andrew@pc02 ~ ]$ ssh -p <port> andrew@137.xxx.xxx.89
Last login: Fri Jan 11 22:47:37 2019 from 137.xxx.xxx.79
```

¡El archivo está ahí!

```
[ andrew@pc02 ~ ]$ ls
hello  multi  xargs
```

```
[ andrew@pc02 ~ ]$ cat hello
hello
```

rsync

rsync es una herramienta de copia de archivos que minimiza la cantidad de datos copiados buscando deltas (cambios) entre archivos.

Supongamos que tenemos dos directorios: **d**, con un archivo, y **s**, con dos archivos:

```
[ andrew@pc01 d ]$ ls
f0
[ andrew@pc01 d ]$ ls ../s
f0 f1
```

Sincronice los directorios (copiando solo los datos que faltan) con **rsync**:

```
[ andrew@pc01 d ]$ rsync -av ../s/* .
sending incremental file list...
```

d ahora contiene todos los archivos que contiene **s**:

```
[ andrew@pc01 d ]$ ls
f0 f1
```

rsync también se puede realizar sobre **ssh**:

```
[ andrew@pc02 r ]$ ls

[ andrew@pc02 r ]$ rsync -avz -e "ssh -p <port>"
andrew@137.xxx.xxx.79:~/s/* .
receiving incremental file list
f0
f1

sent 62 bytes  received 150 bytes  141.33 bytes/sec
total size is 0  speedup is 0.00

[ andrew@pc02 r ]$ ls
f0 f1
```

Procesos de larga duración

yes / nohup / ps / kill

A veces, las conexiones `ssh` pueden desconectarse debido a problemas de red o hardware. Cualquier proceso inicializado a través de esa conexión se “colgará” y finalizará. Ejecutar un comando con `nohup` asegura que el comando no se colgará si el shell está cerrado o si falla la conexión de red.

Ejecute `yes` [continuamente genera "y" hasta que se mata] con `nohup`:

```
[ andrew@pc01 ~ ]$ nohup yes &  
[1] 13173
```

`ps` muestra una lista de los procesos del usuario actual (tenga en cuenta el número de PID 13173):

```
[ andrew@pc01 ~ ]$ ps | sed -n '/yes/p'  
13173 pts/10    00:00:12 yes
```

... cierre sesión y vuelva a iniciar sesión en esta computadora...

¡El proceso ha desaparecido de `ps`!

```
[ andrew@pc01 ~ ]$ ps | sed -n '/yes/p'
```

Pero todavía aparece en la salida `top` y `htop`:

```
[ andrew@pc01 ~ ]$ top -bn 1 | sed -n '/yes/p'  
13173 andrew    20    0   4372    704    636 D   25.0   0.0   0:35.99 yes
```

Mate este proceso con `-9` seguido de su número de ID de proceso (PID):

```
[ andrew@pc01 ~ ]$ kill -9 13173
```

Ya no aparece en `top`, porque ha sido eliminado:

```
[ andrew@pc01 ~ ]$ top -bn 1 | sed -n '/yes/p'
```

cron / crontab / >>

cron proporciona una forma sencilla de automatizar las tareas programadas programadas y regulares.

Puede editar sus trabajos **cron** con **crontab -e** (abre un editor de texto). Agregue la línea:

```
* * * * * date >> ~/datefile.txt
```

Esto ejecutará el comando **date** cada minuto, agregando (con el operador **>>**) la salida a un archivo:

```
[ andrew@pc02 ~ ]$ head ~/datefile.txt
Sat Jan 12 14:37:01 GMT 2019
Sat Jan 12 14:38:01 GMT 2019
Sat Jan 12 14:39:01 GMT 2019...
```

Simplemente elimine esa línea del archivo **crontab** para detener la ejecución del trabajo. Los trabajos **cron** se pueden configurar para que se ejecuten en minutos particulares de cada hora [0-59], horas particulares de cada día [0-23], días particulares de cada mes [1-31], meses particulares de cada año [1-12], o días particulares de cada semana [0-6, dom-sáb]. Esto es lo que representan las cinco estrellas al comienzo del comando anterior, respectivamente. Reemplácelos con números específicos para ejecutarlos en días específicos o en momentos específicos.

Si se va a ejecutar un trabajo independientemente de, por ejemplo, el día de la semana, entonces la posición que representa el día de la semana (la quinta posición) debe contener una estrella (*). Es por eso que el comando anterior se ejecuta cada minuto (el intervalo más pequeño disponible). Los trabajos **cron** se pueden configurar para que se ejecuten solo cuando se reinicia el sistema, con **@reboot** reemplazando las estrellas / números. Los trabajos también se pueden ejecutar un número específico de veces por hora o día o en múltiples momentos específicos por hora / día / semana / mes / etc.

[Consulte este tutorial para obtener más información.](#)

Misceláneos

pushd / popd

Utilice `pushd` y `popd` para mantener una pila de directorios, en lugar de hacer `cd` en todas partes.

Comience en el directorio de inicio, `home`: este será el directorio inferior de nuestra "pila":

```
[ andrew@pc01 ~ ]$ pwd  
/home/andrew
```

Vaya a este directorio con un nombre largo, "empújelo" en la pila con `pushd`:

```
[ andrew@pc01 ~ ]$ pushd /etc/java/security/security.d/  
/etc/java/security/security.d ~
```

Muévase a un tercer directorio y agréguelo a la pila:

```
[ andrew@pc01 security.d ]$ pushd ~/test/  
~/test /etc/java/security/security.d ~
```

Cuando se agrega un nuevo directorio a la pila, se agrega al lado izquierdo de la lista impresa por `pushd`. Para "sacar" el directorio superior [volver al directorio más reciente que agregamos], podemos usar el comando `popd`.

"Pop" al directorio superior, muévase al siguiente en la pila con `popd`:

```
[ andrew@pc01 test ]$ popd  
/etc/java/security/security.d ~  
  
[ andrew@pc01 security.d ]$ pwd  
/etc/java/security/security.d
```

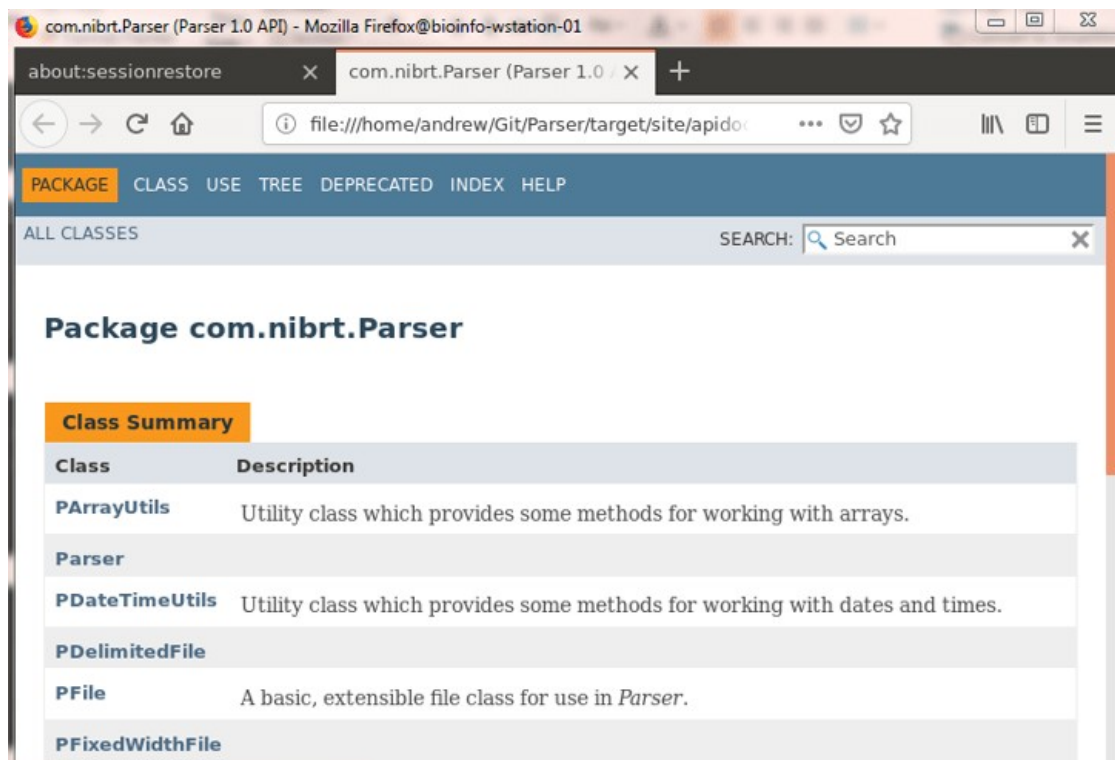
Extrae otro directorio de la pila y volvemos a donde comenzamos:

```
[ andrew@pc01 security.d ]$ popd  
~  
  
[ andrew@pc01 ~ ]$ pwd  
/home/andrew
```

xdg-open

`xdg-open` abre un archivo con la aplicación predeterminada (que podría ser un programa GUI). Es una herramienta realmente útil para abrir documentos HTML desde la línea de comandos. Es el equivalente en Unix del comando `open` de macOS:

```
[ andrew@pc01 security.d ]$ xdg-open index.html
```



xargs

`xargs` vectoriza los comandos, ejecutándolos sobre cualquier número de argumentos en un ciclo.

`ls` en este directorio, su directorio padre y su directorio abuelo:

```
[ andrew@pc01 ~ ]$ export lv=".\\n..\\n../.."
[ andrew@pc01 ~ ]$ printf $lv | xargs ls
.:
multi file
..:
```



```
anaid  andrew  colin...  
  
../...:  
bin    dev    index...
```

Los argumentos se pueden ejecutar a través de una cadena de comandos con la bandera `-I`.
`pwd` este directorio, su directorio padre y su directorio abuelo cambiando con `cd` primero en cada directorio:

```
[ andrew@pc01 ~ ]$ printf $lv | xargs -I % sh -c 'cd %; pwd %'  
/home/andrew  
/home  
/
```

`xargs` es una forma rápida de ejecutar comandos sobre múltiples archivos, pero basado en una lista generada por otro comando.

[Aquí, un gran tutorial sobre `xargs`.](#)

Oneliners

Los “oneliners” son comandos que se pueden mandar como un alias al estar hechos en una sola línea, aquí les dejaré una recopilación de los más interesantes que he visto. ¡Esta sección está abierta a contribuciones! Por lo que si tienen alguno en uso, no duden en compartirlo.

¿Qué ocupa más espacio?

Esto les dará una lista ordenada de

```
[ martin@pc01 ~ ]$ du -s * | sort -nr
```

Pueden canalizarlo a un archivo y monitorear el uso de espacio.

Compartir una carpeta

No tiene ninguna seguridad! Pero para compartir algo rápidamente con alguien [o el celular] funciona. No olviden cerrar después

Estrictamente, no es en script Bash, pero si un oneliner útil.

```
[ martin@pc01 ~ ]$ python -m http.server 8000  
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

La dirección `0.0.0.0`, son todas las direcciones que la computadora tenga, por ejemplo Wi-Fi, Ethernet y localhost.

Para averiguar las direcciones IP de la computadora pueden usar `ip a` o si no funciona, `ifconfig` en particular si el GNU/Linux es más viejuno.

Bonus: Cosas divertidas pero en su mayoría inútiles

w / write / wall / lynx

w es un quién más detallado, que muestra quién inició sesión y qué está haciendo:

```
[ andrew@pc01 ~ ]$ w
17:32:42 up 434 days,  3:11,  8 users,  load average: 2.32, 2.46, 2.57
USER      TTY      FROM          LOGIN@      IDLE   JCPU   PCPU WHAT
colin     pts/9    137.xx.xx.210  03Jan19    5:28m  1:12   0.00s sshd: colin
[priv]
andrew    pts/10   137.xx.xx.199  11:05      1.00s   0.15s  0.04s sshd: andrew
[priv]
colin     pts/12   137.xx.xx.210  03Jan19   34:32    1.59s  1.59s  -bash
...
```

write envía un mensaje a un usuario específico:

```
[ andrew@pc01 ~ ]$ echo "hello" | write andrew pts/10

Message from andrew@pc01 on pts/10 at 17:34 ...
hello
EOF
```

wall es similar a **write**, pero envía el mismo mensaje a todos los usuarios que inician sesión. Los comandos **write** y **wall** solían ser más útiles antes que el correo electrónico, Twitter, WhatsApp y la mensajería instantánea.

lynx es un navegador web completamente funcional y basado en texto:

```
<<< Google
Search Images Maps Play YouTube News Gmail Drive More
Web History | Settings | Sign in

Evelyn Dove's 117th Birthday

-----
Google Search I'm Feeling Lucky Advanced search
Language tools

Google offered in: Gaeilge

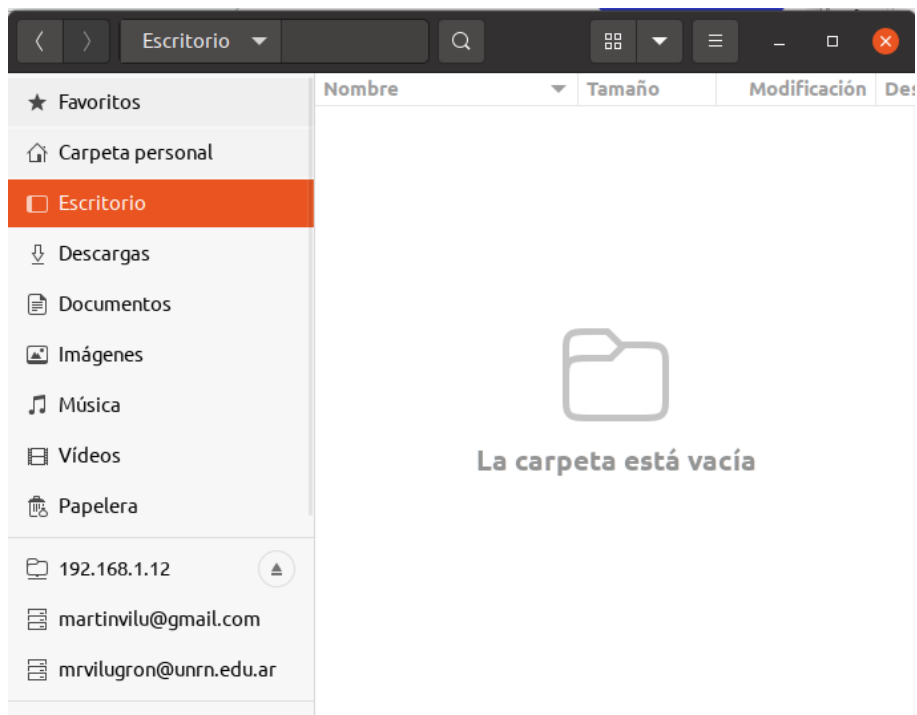
Advertising Programs Business Solutions About Google
Google.ie

2019 - Privacy - Terms

(NORMAL LINK) Use right-arrow or <return> to activate.
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

nautilus / date / cal / bc

nautilus inicializa el explorador de archivos gráfico. (Si no tienen otra ventana abierta, van a necesitar pasarlo a segundo plano con `&` para que la consola no pase a su control.



date muestra la fecha y hora actuales:

```
[ andrew@pc01 ~ ]$ date  
Fri Jan 11 17:40:30 GMT 2019
```

`cal` muestra un calendario ASCII de este mes con la fecha de hoy resaltada:

```
[ andrew@pc01 ~ ]$ cal  
January 2019  
Su Mo Tu We Th Fr Sa  
      1  2  3  4  5  
 6  7  8  9 10 11 12  
13 14 15 16 17 18 19  
20 21 22 23 24 25 26  
27 28 29 30 31
```

`bc` es una calculadora aritmética básica (usa Python en su lugar):

```
[ andrew@pc01 ~ ]$ bc  
bc 1.06.95 ...  
20/4  
5
```

¡Eso es todo por ahora! Aviseme si conoce alguna función adicional o comandos interesantes que deba agregar a esta lista. Además, aviseme si encuentra algún error tipográfico o información errónea. He hecho todo lo posible para comprobarlo todo, ¡pero hay mucho aquí!

Si te gustó esta publicación, ¡considera apoyar mi trabajo [comprándome un café!](#)

Bonus II: ¿Que significan los colores de `ls`?

Cuando está activo, `ls` nos puede indicar con colores de que tipo de archivo está siendo mostrado. Los colores a usar se encuentran almacenados en la variable `LS_COLORS`. Esto puede variar y es posible también agregar resaltes adicionales a más tipos de archivos.

Esta es una lista de los más comunes.

- Azul: directorio
- Verde: Archivo ejecutable
- Celeste: Enlace simbólico
- Rojo con fondo negro: enlace simbólico roto
- Amarillo con fondo negro: Dispositivo
- Rosado: Imagen o foto
- Rojo: Archivo comprimido

Glosario

Tomado del [SW-Carpentry](#)

argumento

Un valor dado a una función o programa cuando se ejecuta. El término a menudo se usa indistintamente (y de manera inconsistente) con [parámetro](#).

bandera o *flag*

Una forma concisa de especificar una opción o configuración a un programa de línea de comandos. Por convención, las aplicaciones Unix usan un guion seguido de una sola letra, como `-v`, o dos guiones seguidos de una palabra, como `--verbose`, mientras que las aplicaciones de DOS usan una barra inclinada, como `/V`. Dependiendo de la aplicación, un indicador puede ir seguido de un único argumento, como en `-o /tmp/output.txt`.

comillas

(en la terminal): Se utilizan comillas de varios tipos para evitar que el intérprete interprete caracteres especiales. Por ejemplo, para pasar la secuencia de caracteres `*.txt` a un programa, generalmente es necesario escribirlo como `'*.txt'` (con comillas simples) para que la terminal no intente expandir el comodín `*`.

comentario

Un comentario en un programa pretende ayudar a los lectores a entender lo que está sucediendo, pero es ignorado por la computadora. Los comentarios en Python, R y la terminal de Unix comienzan con un carácter `#` y se ejecutan hasta el final de la línea; los comentarios en SQL comienzan con `--`, y otros idiomas tienen otras convenciones.

comodín o carácter especial

Un carácter utilizado para coincidir con patrones. En la terminal de Unix, el comodín `*` coincide con cero o más caracteres, para que `.txt` coincida con todos los archivos cuyos nombres terminen en `.txt`.

cuerpo de bucle

El conjunto de instrucciones o comandos que se repiten dentro de un [for bucle](#) o [while bucle](#).

directorio de inicio

El directorio predeterminado asociado con una cuenta en un sistema informático. Por convención, todos los archivos de un usuario se almacenan en o debajo de su directorio de inicio.

directorio de padres

El directorio que “contiene” el que está en cuestión. Cada directorio en un sistema de archivos, excepto el [directorio raíz](#), tiene un padre. Por lo general, se hace referencia al padre de un directorio usando la notación abreviada `..` [pronunciado “dot dot”].

directorio de trabajo actual

El directorio del que se calculan [paths relativos](#); equivalentemente, el lugar donde se buscan los archivos a los que se hace referencia solo por nombre. Cada [proceso](#) tiene un directorio de trabajo actual. El directorio de trabajo actual generalmente se refiere a la notación abreviada `.` [pronunciado “punto”].

directorio raíz

El directorio más alto en un [sistema de archivos](#). Su nombre es “/” en Unix (incluidos Linux y Mac OS X) y “\” en Microsoft Windows.

entrada estándar

Flujo de entrada predeterminado de un proceso. En aplicaciones interactivas de línea de comandos, generalmente está conectado al teclado; en un [pipe](#), recibe datos de [salida estándar](#) del proceso anterior.

expresión regular

Un patrón que especifica un conjunto de secuencia de caracteres. Los RE se usan con mayor frecuencia para encontrar secuencias de caracteres.

extensión de archivo

La parte del nombre de un archivo que aparece después del “.” final. Por convención, esto identifica el tipo de archivo: `.txt` significa “archivo de texto”, `.png` significa “archivo de red portátil de gráficos”, y así. Estas convenciones no son aplicadas por la mayoría de los sistemas operativos: es perfectamente posible (¡pero confuso!) nombrar un archivo de sonido MP3 `homepage.html`. Dado que muchas aplicaciones usan extensiones de nombre de archivo para identificar el [tipo MIME](#) del archivo, los archivos de desincronización pueden hacer que esas aplicaciones fallen.

filtrar

Un programa que transforma una secuencia de datos. Muchas herramientas de línea de comandos de Unix están escritas como filtros: leen datos de [entrada estándar](#), procesarlo y escribir el resultado en [salida estándar](#).

for (bucle)

Un bucle que se ejecuta una vez para cada valor en algún tipo de conjunto, lista o rango. Ver también: [while bucle](#).

guión de terminal - script

Un conjunto de comandos [terminal](#) almacenados en un archivo para su reutilización. Un script de terminal es un programa ejecutado por la terminal; el nombre “script” se usa por razones históricas.

interfaz de línea de comando

Una interfaz de usuario basada en comandos de tipeo, generalmente en un [REPL](#). Ver también: [interfaz gráfica de usuario](#).

interfaz gráfica del usuario

Una interfaz de usuario basada en la selección de elementos y acciones desde una pantalla gráfica, usualmente controlado usando un mouse. Ver también: [interfaz de línea de comandos](#).

lazo

Un conjunto de instrucciones que se ejecutarán varias veces. Consiste en un [cuerpo de bucle](#) y (por lo general) un condición para salir del bucle. Ver también [for bucle](#) y [while bucle](#).

ortogonal

Tener significados o comportamientos que son independientes el uno del otro. Si un conjunto de conceptos o herramientas son ortogonales, se pueden combinar de cualquier manera.

parámetro

Una variable nombrada en la declaración de una función que se usa para mantener un valor pasado a la llamada. El término a menudo se usa indistintamente (y de manera inconsistente) con [argumento](#).

path - ruta

Una descripción que especifica la ubicación de un archivo o directorio dentro de un [sistema de archivos](#). Ver también: [path absoluto](#), [path relativo](#).

path absoluto

Un [path](#) que hace referencia a una ubicación particular en un sistema de archivos. Los paths absolutos generalmente se escriben con respecto al sistema de archivos [directorio raíz](#), y comienzan con “/” (en Unix) o “\” (en Microsoft Windows). Ver también: [path relativo](#).

path relativo

Un [path](#) que especifica la ubicación de un archivo o directorio con respecto al [directorio de trabajo actual](#). Cualquier ruta que no comience con un caracter separador [“/” o “\”] es una ruta relativa. Ver también: [path absoluto](#).

pipe

Una conexión desde la salida de un programa a la entrada de otro. Cuando dos o más programas están conectados de esta manera, se denominan “canalización” o **piping**.

proceso

Una instancia en ejecución de un programa, que contiene código, valores de variables, archivos abiertos y conexiones de red, y así sucesivamente. Los procesos son los “actores” que maneja el [sistema operativo](#); generalmente ejecuta cada proceso durante unos pocos milisegundos a la vez para dar la impresión de que están ejecutándose simultáneamente.

prompt

Un caracter o caracteres se muestran con [REPL](#) para mostrar que está esperando su próximo comando.

read-evaluate-print-loop

[REPL]: Una [interfaz de línea de comandos](#) que lee un comando del usuario, lo ejecuta, imprime el resultado y espera otro comando.

redirigir

Para enviar la salida de un comando a un archivo en lugar de a la pantalla u otro comando, o de manera equivalente, para leer la entrada de un comando desde un archivo.

sistema de archivos

Un conjunto de archivos, directorios y dispositivos de entrada y salida [E/S] (como teclados y pantallas). Un sistema de archivos puede extenderse a través de muchos dispositivos físicos, o muchos sistemas de archivos pueden almacenarse en un solo dispositivo físico; el [sistema operativo](#) administra el acceso.

salida estándar

Flujo de salida predeterminado de un proceso. En aplicaciones interactivas de línea de comandos, los datos enviados a la salida estándar se muestran en la pantalla; con un [pipe](#), se pasa a la [entrada estándar](#) del siguiente proceso.

sistema operativo

Software que gestiona las interacciones entre usuarios y los [procesos](#) de hardware y software. Por ejemplo, Linux, OS X y Windows.

subdirectorio

Un directorio contenido en otro directorio.

tabulación completa

Una función proporcionada por muchos sistemas interactivos en los que presionar la tecla Tab activa la finalización automática de la palabra o comando actual.

terminal

Una [interfaz de línea de comando] como Bash [Bourne-Again Shell] o la terminal de Microsoft Windows DOS que permite a un usuario interactuar con el [sistema operativo](#).

terminal de comandos o terminal de shell

Ver [terminal](#)

tipo MIME

Los tipos MIME (extensiones multipropósito de correo de Internet) describen diferentes tipos de archivos para el intercambio en Internet, por ejemplo, imágenes, audio y documentos.

variable

Un nombre en un programa que está asociado con un valor o una colección de valores.

while (bucle)

Un bucle que se ejecuta siempre que alguna condición sea verdadera. Ver también: [for bucle](#).

La lista de los pendientes

Tomar todas las referencias externas y pasarlas por un gestor bibliográfico.

No estaría mal agregar los primeros pasos de control de versiones.

Agregar información sobre `screen`

Agregar algunos *one-liners* de Python útiles (así como otros scripts cortos interesantes)

Indicar que es un built-in de `bash` y que es un programa separado.

Agregar una sección sobre la “filosofía UNIX” que impulsa a todas estas herramientas

Agregar las otras señales en `kill` para darle oportunidad al programa de que cierre y evitar bajarlo a la fuerza.

Naturalmente, se aceptan contribuciones para resolver estos pendientes.