Primera vez con un SO en serio

Ignacio Vissani

23 de marzo de 2011

Resumen

Esta clase es una introducción al menjo básico de la consola de Linux para un usuario.

1. Intro

Hoy vamos a ver algunas cosas básicas para manejarnos en el *shell*. Empecemos por ver qué es un *shell*.

"Un *shell* de `Unix` es un intérprete de comandos y un ambiente de *scripting* que provee una interfaz tradicional para el sistema operativo `Unix` y para los sistemas operativos del estilo de `Unix`. El usuario comanda la opeartoria de la computadora ingresando comandos en forma de texto para que el intérprete de línea de comandos lo ejecute o mediante la creación de *scripts* de texto formados por uno o más comandos de este tipo."

Es decir que es una interfaz provista por algunos sistemas operativos (la gran mayoría) en modo texto que permite la ejecución de comandos o secuencias de comandos.

Un *shell* es también un programa. Un mismo sistema operativo puede proveer varios *shell*s distintos. Existen por ejemplo el Bourne-*shell*(sh) , el C-*shell*(csh), el Korn-*shell*(ksh), etc., todos *shell*s del sistema `Unix`. Hoy en día uno de los más difundidos es el Born again *shell*(bash) que es el *shell* por defecto de `Linux`.

2. Primeros pasos

Para poder movernos sin problemas tenemos que saber algunos comandos básicos. El primer comando que vamos a aprender es el `man`.

Ejercicio 1. *En la consola tipee el comando* `man man`*.*
¿Qué ocurre?

Ahora que ya sabemos cómo pedir ayuda, la idea es probar algunos comandos básicos. En cada ejercicio lo que deberían hacer es primero consultar las páginas del manual del comando para ver cómo se usa, qué opciones tiene, etc., y después hacer lo que pide el ejercicio.

Ejercicio 2. *1.* `pwd` *Indique qué directorio pasa a ser su directorio actual si ejecuta:*

a) `cd /usr/bin`

b) `cd`

c) *¿Cómo explica el punto anterior?*

2. `cat` *¿Cuál es el contenido del archivo* `/home/<usuario>/.profile`*?*

3. `find` *Liste* ***todos*** *los archivos que comienzan con* `vmlinuz`*.*

   Estos archivos son imágenes del kernel Linux.

4. `locate` *Liste* ***todos*** *los archivos cuyo nombre contiene* `vmlinuz`*. ¿Qué diferencia hay entre este comando y el* `find`*?*

5. `mkdir` *Genere un directorio* `/home/<usuario>/tp`*.*

6. `cp` *Copie el archivo* `/etc/passwd` *al directorio* `/home/<usuario>/tp`*.*

7. `grep`

   Muestre las líneas que tienen el texto "localhost" en el archivo `/etc/hosts`*.*

   Muestre todas las líneas que tengan el texto "POSIX" de todos los archivos (incluyendo subdirectorios) en `/etc`*. Evite los archivos binarios y aquellos archivos y directorios que no tienen permiso de lectura para su usuario.*

8. `passwd` *Cambie su password.*

9. `rm` *Borre el archivo* `/home/<usuario>/tp/passwd`

10. `ln`

    Enlazar el archivo `/etc/passwd` *a los archivos* `/tmp/contra1` *y* `/tmp/contra2`*.*

    Hacer un `ls -l` *para ver cuantos enlaces tiene* `/etc/passwd`*.*

    Estos enlaces se llaman "hardlinks". Cada nuevo enlace referencia el mismo espacio ocupado del disco rígido, y por lo tanto cada hardlink es igual de representativo de esos bytes ocupados del disco rígido. El espacio ocupado solamente se liberará cuando todos los enlaces hayan sido borrados.

    Ahora enlace el archivo `/etc/passwd` *de manera "soft" al archivo* `contra3`*.*

    Verifique con `ls -l` *que no aumentó la cantidad de enlaces de* `/etc/passwd`*.*

    Estos enlaces se llaman "softlinks" y apuntan no a los bytes del disco rígido sino a la ruta del archivo a ser enlazado. Operar sobre el softlink es igual que operar sobre el archivo, sin embargo los softlinks no cuentan en la cantidad de enlaces (ya que no apuntan a los bytes ocupados del disco rígido) y pueden ser borrados sin afectar al archivo original, aunque si se borra el archivo original el softlink quedará huérfano y no apuntará a nada.

11. `df` *¿Qué espacio libre tiene cada uno de los filesystems montados?*

12. `ps` *¿Cuántos procesos de usuario tiene ejecutando? Indique cuántos son del sistema.*

13. `uptime` *¿Cuanto tiempo lleva ejecutando su máquina virtual?*

14. `uname` *¿Qué versión del kernel de Linux está utilizando?*

3. Entrada y Salida standard

Ahora que ya sabemos movermos por el *filesystem* vamos a ver algunas cosas más avanzadas.

Muchas veces queremos que la salida de un comando sea la entrada de otro. Para eso usamos el caracter pipe (| ó ¦) que hace justamente eso. Es decir que `comando1 ¦ comnado2` ejecuta `comando1` y luego ejecuta `comando2` pasándole por la entrada standard la salida de `comando1`.

Ejercicio 3. `cat+wc` *Diga cuántos usuarios hay definidos en el sistema (Pista: todos los usuarios están definidos en /etc/passwd)*

También podríamos querer guardar la salida de un comando en un archivo para poder verla más tarde. Para eso existe > que *redirecciona* la salida standard. Es decir que `comando1 >/home/<usuario>/pepe` guarda la salida de `comando1` en el archivo */home/<usuario>/pepe*.

Investigue la diferencia entre usar > y usar >> para redireccionar la salida standard.

Ejercicio 4. *1.* `seq` *Genere el archivo /home/<usuario>/1000-lineas que debe contener* ***500*** *líneas y en cada una de ellas debe estar el número de línea. Es decir que debe ser de la forma*

1
2
⋮
500

2. `seq` *Agrege al archivo /home/<usuario>/1000-lineas las 500 líneas que faltan para completar las 1000.*

Ahora unos ejercicios un poco más interesantes:

Ejercicio 5. *1. STDOUT*

a) *Conserve en el archivo* `/home/<usuario>/tp/config` *la salida del comando* `ls` *que muestra todos los archivos del directorio* `/etc` *y de los subdirectorios bajo* `/etc`.

b) *Presente cuantas líneas, palabras y caracteres tiene* `/home/<usuario>/tp/config`.

c) *Agregue el contenido, ordenado alfabéticamente, del archivo* `/etc/passwd` *al final del archivo* `/home/<usuario>/tp/config`.

d) *Presente cuantas líneas, palabras y caracteres tiene* `/home/<usuario>/tp/config`.

2. Pipes

a) *Liste en forma amplia los archivos del directorio* `/usr/bin` *que comiencen con la letra "a". Del resultado obtenido, seleccione las líneas que contienen el texto* `apt` *e informe la cantidad de caracteres, palabras y líneas.*

Está prohibido, en este ítem, usar archivos temporales de trabajo.

4. Editores (nano, vi)

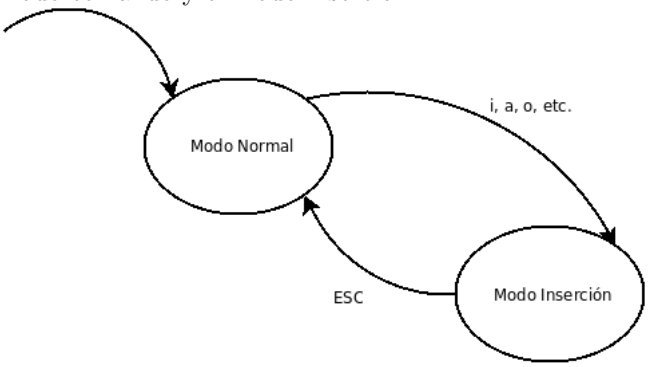
4.1. vi

Si bien podemos crear un archivo de texto usando el comando `cat` en general resulta vital para el uso de cualquier sistema operativo contar con un editor de textos. Vamos a ver someramente los comandos básicos de dos de los editores más populares de los entornos del tipo `Unix`.

Empecemos con el `vi` que es el editor por defecto de los sistemas `Unix`.

Para aprender a usar `vi` vamos a crear un peque no programa en `C`. Pero primero veamos algunas cosas básicas para poder manejarnos con el `vi`.

Lo primero que debemos saber es que el `vi` tiene dos modos de operación. El modo comando y el modo inserción.



Para ingresar al `vi` simplemente ejecutamos el comando `vi <nombre_archivo>`. Una vez adentro para salir del editor ejecutamos el comando `:q` (notar que todos los comandos en `vi` se ejecutando estando, naturalmente, en el modo comando).

Al iniciarse `vi` se enecuentra en modo comando. Para pasar del modo comando al modo inserción existen varios comandos. Los más usados son `i`, `a`, `o`.

Para pasar del modo inserción al modo comando se usa la tecla `ESC` ó `CTRL+[`.

El `vi` es un editor muy poderoso con muchos comandos. Sólo voy a mencionar los básicos para poder editar un archivo decentemente.

Pedir ayuda	:help
Desplazamiento	h,j,k,l ó $\leftarrow, \downarrow, \uparrow, \rightarrow$
Más desplazamiento	\$ (fin de línea), $\wedge$ (comienzo de línea), G (fin de archivo), gg (comienzo de archivo) :♯ (GOTO línea número ♯)
Edición	x (borrar caracter bajo el cursor), dd (borrar línea) r (reemplazar caracter), R (reemplazar desde cursor) cw (cambiar palabra)
Búsqueda	/<texto>, n (próxima coincidencia), N (anterior coincidencia)
Archivo	:w (guardar), :e (abrir), :q (salir), :wq (guardar y salir)

4.2. nano

El `nano` es un editor más "visual". Al abrirlo vamos a ver que la pantalla tiene tres secciones.

1. Superior: Se indica la versión de `nano`, el nombre del archivo que está siendo editado y si ha habido cambios o no.

2. Centro: Se muestra el contenido que está siendo editado.

3. Inferior: Se muestra cierta información importante y los atajos más apropiados dependiendo del contexto.

Acá no hay mucho que explicar ya que el editor es bastante autoexplicativo. Vale la pena mencionar que en los atajos que se muestran en la parte inferior el circunflexo ($\wedge$) representa a la tecla `CTRL` y la eme mayúscula representa a la tecla `ALT` o Meta.

Para tener una idea, sumaricemos algunos atajos básicos.

Pedir ayuda	CTRL-G
Desplazamiento	CTRL-B,CTRL-N,CTRL-P,CTRL-F ó $\leftarrow, \downarrow, \uparrow, \rightarrow$
Más desplazamiento	CTRL-E (fin de línea), CTRL-A (comienzo de línea), ALT-/ (fin de archivo), ALT-\ (comienzo de archivo) CTRL-_ (GOTO línea,columna)
Edición	CTRL-D (borrar caracter bajo el cursor), CTRL-K (cortar línea)
Búsqueda	CTRL-W (buscar una cadena), ALT-W (repetir última búsqueda)
Archivo	CTRL-O (guardar), CTRL-R (abrir), CTRL-C (salir), CTRL-X sobre archivo modif. (guardar y salir)

Ejercicio 6. *1. `less` Muestre el contenido del archivo 1000-lineas.txt por pantalla de manera que pueda desplazarse hacia adelante y hacia atrás en el mismo.*

2. `/` Sin salir de `less` busque todas las apariciones de la cadena "50" en el archivo 1000-lineas.txt

3. `v` Sin salir de `less` edite el contenido del archivo. Busque y reemplace todas las apariciones de la cadena "50" por la cadena "11".

4. `q` Salga del editor. ¿A dónde vuelve? ¿Qué pasó con la búsqueda que había realizado anteriormente?

Ejercicio 7. *Usando el editor que prefiera tipee el siguiente programa en lenguaje `C`(lo vamos a usar ma nana).*

loop.c