El shell de GNU/Linux

1. Aspectos generales

El *shell* es el programa que permite al usuario interactuar con el sistema en una terminal de texto (consola). Cada vez que un usuario inicia sesión en una terminal, el *shell* es ejecutado y toma el control de la entrada y salida del sistema.

Existen varios *shells*. Algunos ejemplos son:

- csh
- zsh
- ksh
- dash
- bash

El *shell* predeterminado en sistemas basados en Debian es *bash* (*Bourne-Again Shell*, basado en el *Bourne Shell* del UNIX original).

La herramienta principal de un *shell* es la línea de comandos, que le permite al usuario ejecutar comandos de forma interactiva. Además la mayoría de los *shells* tienen otras herramientas que los convierten en lenguajes de programación completos con variables, condicionales, lazos de control, funciones, etc. Es posible escribir pequeños programas o *scripts* para automatizar la ejecución de secuencias de comandos.

2. La línea de comandos

Nota: Las herramientas y características descritas en este documento aplican únicamente para *bash*. Si bien es cierto algunas características son comunes a todos los *shells*, es necesario investigar cuales son las herramientas equivalentes si se utiliza otro *shell*.

Cuando la línea de comandos está disponible para ejecutar nuevos programas, el *shell* despliega el indicador de comandos (*command prompt*): `$` para el usuario normal y `#` para el superusuario. Además, el indicador de comandos puede configurarse para mostrar información útil para el usuario. La información mostrada por defecto en sistemas Debian/Ubuntu es:

```
xxxx@yyyy:zzzz$
```

xxxx - Nombre de usuario

yyyy - Nombre del sistema

zzzz - Directorio actual

Bash posee herramientas que facilitan al usuario el trabajo en la línea de comandos. Algunas son:

- Autocompletado de programas, nombres de archivo y argumentos de programas (tecla `TAB`)
- Historial (teclas `↑` y `↓`, comando `history`)
- Búsqueda en el historial (`Ctrl-r`)

3. Nombres de archivo y directorio

En el *shell,* los nombres de archivo y directorio pueden ingresarse usando rutas absolutas (respecto a la *raíz*) o relativas (respecto al directorio actual).

El carácter '∼' (*tilde* o virgulilla) corresponde al directorio personal del usuario. Si el usuario es *juan,* la ruta `∼/Documents/doc1.odt` corresponderá a `/home/juan/Documents/doc1.odt`.

Además, es posible usar comodines para referirse a nombres de archivo que contengan un determinado patrón:

`*` - Cero o más caracteres cualesquiera.

`?` - Un único carácter cualquiera.

Ejemplos:

`/home/pedro/Documents/*.pdf` corresponde a todos los archivos que terminan con `.pdf` del directorio `/home/pedro/Documents`.

`/home/maria/??.txt` corresponde a archivos en el directorio `/home/maria` cuyo nombre tiene dos caracteres cualesquiera seguidos por `.txt` (`01.txt`, `aa.txt`, etc.)

4. Variables de entorno

Como se mencionó anteriormente, el *shell* tiene características equivalentes a los lenguajes de programación. Las variables de entorno permiten almacenar valores de igual manera que las variables de un lenguaje de programación.

```
$ A=Linux
$ echo $A
Linux
$ DEST=/home/adiaz/Documents
$ cp doc1.pdf $DEST
```

Al usar las variables de entorno deben tenerse algunas consideraciones especiales:

- Cuando se desea almacenar un valor en una variable, debe usarse la sintaxis `VARIABLE=valor`, sin espacios alrededor del signo '='.
- Aunque no es obligatorio, se acostumbra usar mayúsculas para los nombres de las variables de entorno. Cuando se desea obtener el valor almacenado en una variable de entorno se debe anteponer el carácter '$' al nombre de la variable.

Las variables de entorno tienen un alcance limitado al *shell* que se está ejecutando. Si se ejecuta un nuevo *shell* a partir del *shell* actual la variable no estará definida, a menos que se use el comando `export`:

```
$ export A=Linux
```

Las variables de entorno pueden borrarse usando el comando `unset`:

```
$ unset A
```

5. El *PATH*

PATH es una variable de entorno que almacena una lista (separada por ':') de los directorios que contienen ejecutables en el sistema:

```
PATH=/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin:/usr/local/sbin
```

Los programas que se encuentran en un directorio del *PATH* pueden ejecutarse escribiendo sólo el nombre de archivo en la línea de comandos. Si un ejecutable no está en el *PATH,* debe escribirse la ruta completa al archivo.

6. Archivos de configuración del *shell*

`/etc/profile` - Archivo de configuración para todos los *shells* compatibles con *Bourne Shell* (aplica para todos los usuarios).

`/etc/bash.bashrc` - Archivo de configuración para bash (aplica para todos los usuarios).

`~/.profile`, **`~/.bash_profile`**, **`~/.bashrc`** - Archivos de configuración para el usuario. Se recomienda agregar todas las configuraciones en `~/.bashrc`.

7. Entrada y salida estándar

Los caracteres ingresados por el usuario en una terminal se conocen como la entrada estándar. De forma análoga, la salida de los programas que se imprime en una terminal es conocida como la salida estándar. El *shell* es el programa que se encarga de manejar la entrada y la salida estándar.

El sistema operativo representa la entrada y la salida estándar usando tres flujos (*streams*) de texto:

stdin - Caracteres ingresados por el usuario.

stdout - Salida normal de los programas.

stderr - Mensajes de error emitidos por los programas

7.1. Redirección de entrada y salida

El *shell* permite manipular por separado estos tres «archivos» y redirigirlos hacia otro destino:

```
$ ls -l /etc/ > /tmp/lista
```

La salida (*stdout*) del comando `ls` es redirigida al archivo `/tmp/lista`. Si el archivo ya existía, sus contenidos son borrados antes de escribir en él.

```
$ ls -l /etc/ >> /tmp/lista
```

La salida (*stdout*) del comando `ls` es redirigida al archivo `/tmp/lista`. Si el archivo ya existía, el nuevo contenido es agregado al final.

```
$ passwd < ~/contraseña.txt
```

La entrada (*stdin*) para el comando `passwd` es tomada del archivo `~/contraseña.txt`.

```
$ mkdir /srv/www 2> /tmp/error.txt
```

El error estándar (*stderr*) del comando `mkdir` es redirigido a `/tmp/error.txt`.

```
$ tar xvzf archivo.tar.gz &> /tmp/salida-tar
```

stdout y *stderr* del programa `tar` son redirigidos al archivo `/tmp/salida-tar`.

7.2. Tuberías (*pipes*)

Las tuberías permiten conectar la salida y el error estándar de un comando con la entrada estándar de otro:

```
$ comando1 | comando2
```

En este caso, la salida y el error estándar de `comando1` son enviados a la entrada estándar de `comando2`.
Ejemplos:

```
$ apt-cache search text editor | less
$ ls -l /etc/ | grep fstab
$ gzip -dc archivo.gz | wc
```

7.3. El comando **tee**

tee es un comando que escribe en la salida estándar y en un archivo todo lo que es introducido en su entrada estándar. Usando tee es posible almacenar la salida estándar de un programa en un archivo y verla en la terminal al mismo tiempo:

```
$ gcc -o programa programa.c | tee salida_gcc
```

8. *Scripts* básicos en bash

Los *scripts* son archivos de texto con una serie de comandos, que serán ejecutados secuencialmente cuando se ejecute el *script*. De esta forma es posible ejecutar en un solo paso múltiples comandos y almacenar estos comandos para su posterior uso. Cuando se utilizan otras características de bash como las variables de entorno y las estructuras de control (que no serán estudiadas en este curso), es posible crear programas complejos. Un *script* simple de bash tiene la siguiente forma:

```
#!/bin/bash

# Esto es un comentario
```

```
comando-1
comando-2
...
comando-n
```

Para ejecutar el *script* deben cambiarse los permisos del archivo para que este sea ejecutable. Una vez hecho esto puede llamarse por su nombre (o ruta completa si el directorio no está en el *PATH*):

```
$ chmod +x script
$ ./script
```

Otra opción es pasar el *script* como argumento a un nuevo *shell* usando el comando `bash`. El nuevo *shell* ejecutará el *script* y finalizará:

```
$ bash script
```

9. Programación de tareas con cron

cron es un recurso disponible en la mayoría de los sistemas UNIX que permite programar tareas (programas) para que se ejecuten en un momento determinado. Es muy útil para ejecutar utilidades de administración del sistema de manera periódica, sin necesidad de supervisión.

Cada usuario puede crear su propia configuración de *cron* para ejecutar sus propias tareas, limitado obviamente a los recursos sobre los cuales tiene permisos. También existe un *cron* para el sistema, en el cual es posible programar tareas que requieran privilegios de administración.

Para agregar tareas al sistema *cron*, debe editarse un archivo de configuración (`crontab`).

En el caso del *cron* del usuario, el archivo de configuración se edita usando el comando `crontab -e`. En el caso del *cron* del sistema, debe editarse el archivo `/etc/crontab` con cualquier editor de texto.

La sintaxis del `crontab` es la siguiente:

Cron del usuario: `m h dm mes ds comando` *Cron* del sistema: `m d dm mes ds usuario comando`

Donde:

m - minuto (0 - 59)

h - hora (0 - 23)

dm - día del mes (1 - 31)

mes - mes (1 - 12)

ds - día de la semana (0 - 7)

Algunos ejemplos (*cron* del usuario):
Ejecuta `comando1` cada hora:

```
0 * * * * comando1
```

Ejecuta `comando2` cada día a las 11:43 p.m.:

```
43 23 * * * comando2
```

Ejecuta `comando3` todos los martes a las 12:00 m.d.:

```
0 12 * * 2 comando3
```

Ejecuta `comando4` de lunes a viernes a las 12:00 a.m.:

```
0 0 * * 1-5 comando4
```