Bash y Línea de Comando en Linux

1. Bash: Bourne again shell

Es una **aplicación de consola** que "*escucha*" y ejecuta comandos (escritos en su lenguaje propio). Es un programa ejecutable (un **binario**) que permite interactuar con el sistema (y otros programas) a través de una **interfaz de texto** dentro de una **terminal**. Puede ser operado en 2 modos distintos:

- **Interactivo:** la consola espera que el usuario ejecute comandos y bloquea una nueva ejecución hasta no completar la anterior.
- **No-interactivo:** permite ejecutar scripts (una secuencia de comandos dentro de un archivo).

2. Unix Shell

Un **shell de Unix** es un intérprete de línea de comandos o shell que proporciona una interfaz de usuario de línea de comandos para sistemas operativos similares a Unix. El shell es tanto un lenguaje de comandos interactivo como un lenguaje de scripting, y el sistema operativo lo utiliza para controlar la ejecución del sistema mediante scripts de shell.

Los usuarios suelen interactuar con un shell de Unix utilizando un emulador de terminal; sin embargo, la operación directa a través de conexiones de hardware en serie o Secure Shell es común para los sistemas de servidor. Todos los shells de Unix proporcionan comodines de nombre de archivo, *piping*, documentos, sustitución de comandos, variables y estructuras de control para pruebas de condición e iteración.

3. Categorías de comandos UNIX

- Sistema de archivos
- Administración de procesos
- Procesamiento de texto
- Programación shell
- Misceláneos

Fuente: https://en.wikipedia.org/wiki/List_of_Unix_commands

4. Procesos y Descriptores de Archivos (File Descriptors)

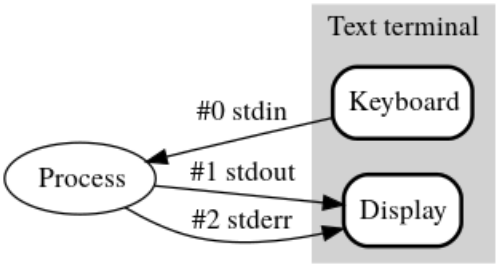
¿Qué es y cómo funciona un programa?

Un **programa** es una colección de instrucciones ejecutadas por el kernel del sistema operativo (código que comunica con el hardware).

Un **proceso** es la ejecución del programa.

Los procesos se comunican entre sí (y con el *filesystem* y otros dispositivos) a través de *file descriptors* (específicos a cada proceso):

- **entrada estándar** (o *stdin* o fd0): procesos en la terminal tienen conectado el stdin al teclado o input de la terminal.
- **salida estándar** (*stdout* o fd1): procesos en la terminal tienen conectados el stdout al display o ventana de la terminal.
- **error estándar** (o *stderr* o fd2): también conectado al display.



5. Primeros comandos en bash

Para navegar el *filesystem*

1. **`cd`** nos permite cambiar de directorio y **`pwd`** imprime el directorio actual (*working directory*).

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

cd ~
pwd
```

   - **`cd~`** cambia el *working directory* a *home*.
   - **`cd`** puede recibir como argumentos *paths* relativos: **`cd ..`** mueve el *working directory* al directorio padre del actual.

2. **`ls`** lista los contenidos de un directorio.

   - Usar **`ls -h`** o `man ls` para ver la ayuda o el manual del comando **`ls`**.
   - Por ejemplo **`ls -la`** lista todo el contenido (incluido archivos ocultos) de un directorio en modo lista.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

man ls
```

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

ls -l
```

Podemos encandenar comandos con `;` o `&&`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

cd ~ ; pwd
```

Con `;` el comando a la derecha se ejecuta independientemente de si el comando anterior falló; con `&&` sólo si el anterior no falló.

Notar que los comandos siempre siguen la siguiente estructura:

```
$> comando -f -flag  argumento$
```

- Las opciones (flags) suelen poder escribirse de manera abreviada, al utilizar un solo guión `-`.
- Podemos acomodar muchas opciones abreviadas detrás de un solo guión.
- Por eso se usa dos guiones – con las opciones no abreviadas: si no una palabra se interpretaría como muchas opciones.

6. Manipulación de directorios y archivos

1. `mkdir` permite crear directorios y `touch`, archivos.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

mkdir test1 test2
touch foo.txt bar.txt
ls -l
```

Con el flag `-p` podemos crear directorios intermedios: `mkdir -p dir1/dir2`. De lo contrario `dir1` debe preexistir o el comando devolverá error.

2. `rmdir` permite remover directorios y `rm`, archivos.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

rmdir test1
rm foo.txt
ls -l
```

- Podemos eliminar directorios usando `rm` con la opción (o flag) `-rf`.
- Podemos usar wildcards para realizar una operación sobre múltiples archivos: `rm *.txt`.

3. `mv` y `cp`: Mover y copiar archivos con `mv` y `cp`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

cp bar.txt baz.txt
mv bar.txt test2
ls -l
```

- Para renombrar un archivo también usamos **mv**.
- Podemos copiar directorios con el flag `-r`.
- Además podemos hacer copias simbólicas con **ln -s <file> <link>** (son parecidas a los accesos directos).

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%%% bash

cp bar.txt baz.txt
mv bar.txt test2
ls -l
```

---

7. Redireccionamiento de `stdin/stdout` y `stderr`

1. **echo** escribe al `stdout` (que luego imprime a la terminal) y **printf** hace lo propio pero formateando primero.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%%% bash

echo 'hello world'
```

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%%% bash

printf '%s\n' 'foo' 'bar' 'baz'
```

El operador *pipe*, `|`, redirecciona el `stdout` de un proceso en otro.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%%% bash

echo 'hello world' | wc -w
```

El operador > redirecciona el `stdout` a una ruta particular.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%%% bash

printf '%s\n' 'foo' 'bar' 'baz' > foo.txt

cat foo.txt
```

Podemos appeandear con » y redreccionar a `/dev/null` para descartar el output.

< permite obtener input de una ubicación particular (distinta al `stdin`).

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%%% bash

sort < foo.txt
```

Podemos también redireccionar por separado el `stdout (1)`, el `stderr` (2) o ambos con `1>`, `2>` y `&>`.

8. Vista de archivos

Distintas formas de acceder al contenido de un archivo.

1. **`head -n N`** imprime las primeras N lineas de un archivo (por defecto 10).

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%%% bash

head –n 2 foo.txt
```

2. **`tail -n N`** imprime las ultimas N lineas.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%%% bash

tail –n 2 foo.txt
```

3. **`cat`** concatena una lista de archivos y envia el output a `stdout`; se (abu)usa para ver contenidos de un archivo.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%%% bash

cat foo.txt | head –n 2 | tail –n 1
```

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%%% bash

echo 'new' > baz.txt
echo 'bar' >> baz.txt
cat foo.txt baz.txt
```

Podemos también usar programas como **`more`** y **`less`** para ver los contenidos del archivo y movernos (para adelante y para atrás) dentro del mismo.

9. Procesamiento de texto y matcheo de patrones

1. **diff** muestra líneas diferentes entre dos archivos.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

!cat foo.txt
```

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

!cat baz.txt
```

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

!diff foo.txt baz.txt
```

---

2. **cut -d <sep> -f-n** corta una línea en el separador `<sep>` e imprime los elementos hasta la posición n.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

echo 'foo-bar-baz' | cut -d '-' -f-1
```

---

3. **sed** permite hacer operaciones sobre texto y en general se usa para buscar y reemplazar cadenas de caracteres.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

echo 'replace this by this' | sed s/this/that/g
```

---

sed es una herramienta muy poderosa (y compleja), este es un gran tutorial http://www.grymoire.com/Unix/Sed.html para aprenderla en profundidad.

4. **grep** se utiliza para buscar texto dentro de uno o múltiples archivos.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

grep -n 'bar' *.txt
```

---

5. **awk** tiene una funcionalidad "similar"...

---

10. Uso de disco y procesos

1. `df -h` muestra el espacio disponible en el sistema (para todo *filesystem* montado).

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

df -h
```

---

2. `du -shc` muestra el uso de espacio para un directorio particular (y sus subdirectorios).

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

du -shc
```

---

3. Podemos listar todos los procesos (*jobs*) con `ps`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

ps aux | head -n4
```

- Podemos matar el proceso usando el PID: `kill 26860`.
- También podemos obtener el id de un proceso usando `pgrep`.
- `htop` es un visualizador de procesos interactivo que permite realizar operaciones sobre ellos.

```
%% bash

$> sudo apt install htop
$> htop
```

---

11. Comandos y bindings misceláneos

1. `which` devuelve la ubicación de un ejecutable.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

which python
```

---

2. `find` itera sobre un directorio para encontrar archivos (o sub-directorios) cuyo nombre matchee un patrón.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

find . -name '*.png'
```

- Podemos cambiar los permisos de un archivo con **chmod** y el dueño de un archivo con **chown**.
- Podemos abandonar la consola con **exit** (o `Ctrl+d`).
- **clear** limpia la pantalla de la terminal.
- `Ctrl+r` nos permite buscar en la historia de comandos previamente ejecutados.

12. Variables de entorno y configuración de bash

Las variables de entorno son variables persistentes creadas dentro de la consola las cuales asignamos con = y accedemos con `$`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

var='foo'
echo $var
```

Para que una variable de entorno esté disponible en un subproceso (comandos ejecutados dentro de la consola) debemos usar **export** al definirla.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

export VAR='foo'
python -c "import os; print(os.environ['VAR'])"
```

El archivo `.bashrc` es el archivo de configuracion de bash y es leido/ejecutado cada vez que iniciamos una consola de bash.

Podemos forzar que el archivo se ejecute sin reiniciar la consola.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

$> echo 'echo testing!' >> ~/.bashrc
$> source ~/.bashrc
```

Se puede agregar un directorio a la variable de entorno `PATH` que define donde bash busca ejecutables.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

$> echo 'PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
$> source ~/.bashrc
```

13. Scripting en bash

Un script de bash (extension por defecto `.sh`) contiene cualquier número de comandos normal de la consola. Se ejecuta con `sh` o `source` (o **bash** dado que tiene construcciones propias).

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

echo 'ls' > script.sh
bash script.sh
```

Podemos hacerlo ejecutable con **chmod** y ejecutarlo con `./`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
%% bash

chmod +x script.sh
./script.sh
```

Puede contener condicionales, bucles, funciones y mucho más.

Condicionales

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
x='foo'

if [[ $x == 'foo' ]]; then
    echo "x is equal to $x"
else
    echo "x not foo"
fi


case $x in
    foo)
        echo "x is equal to $x" ;;
    *)
        echo "x not foo" ;;
esac
```

Bucles

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
for i in {1..5}; do
  echo "$i"
done
```

```
while [ $x -lt 3 ]; do
  ...
done
```

Al igual que en Python podemos terminar la ejecución con **`break`** y reiniciarla con **`continue`**.

Funciones

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y observe los resultados:*

```
f() {
  x="$1"
  y="$2"
  echo "$((x + y))"
}

echo "$(f 1 4)"
```

Notar el uso de `$(<cmd>)` que evalua el comando `<cmd>`.
Ver muchos otros casos en este cheatsheet.