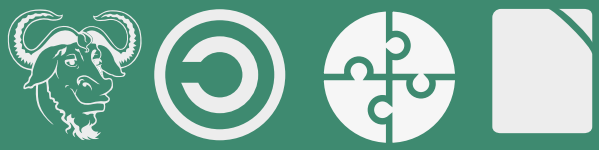
UNL VIRTUAL

Tecnicatura Universitaria en Software Libre



Administración de GNU/Linux I

Unidad 5
Línea de comando y Scripting

Autor
Leonardo Martinez

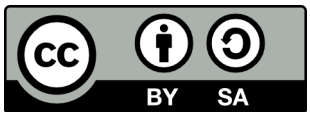
FICH UNL

Contenido

Edición 2022.

Autor: Leonardo Martinez

¡Copia este texto!



1 Introducción a la consola

A simple vista un *shell* es un intérprete de comandos. Normalmente es ejecutado en un modo interactivo donde el usuario escribe comandos, y el *shell* provee salidas de texto a esos comandos. La mayoría de los sistemas operativos poseen *shells*; hasta el *MS-DOS* provee un *shell* simple de nombre **command.com**. En ocasiones el *shell* también es llamado el *Intérprete de la Línea de Comandos* (CLI - Command Line Interpreter).

El intérprete de comandos, el *shell*, tiene una importancia fundamental en las tareas del administrador de sistemas brindando la posibilidad de ejecutar archivos de comandos secuenciales escritos en texto plano y que éste los interpretará. Estos archivos se denominan *shell scripts* y son una herramienta imprescindible en el día a día de los administradores de sistemas.

La práctica de *shell scripting* es necesaria ya que las distrbuciones *GNU/Linux* se basan en este tipo de recurso para iniciar el sistema ejecutando los archivos de `/etc/rcS.d`. Los administradores deben tener los conocimientos necesarios para trabajar con ellos, entender el funcionamiento del sistema, modificarlos y adecuarlos a las necesidades específicas del entorno en el cual trabajen.

Se considera el *scripting* casi un arte, sin embargo no es difícil de aprender, se pueden aplicar técnicas de trabajo por etapas que se ejecutará en forma secuencial y el conjunto de operadores u opciones no es tan extenso como para que genere dudas sobre qué recurso utilizar. La sintaxis es dependiente del *shell* utilizado, pero al ser un lenguaje interpretado con encadenamiento de secuencias de comandos, es muy fácil de depurar y poner en funcionamiento.

Entre los lenguajes interpretados que han popularizado la técnica del *scripting* están **Perl**, **AWK**, **sed**, **Lisp**, **PHP**, **Python** y **Ruby**. Algunos incluyen características como el soporte para orientación a objetos, programación imperativa o funcional.

La mayoría de las distribuciones *GNU/Linux* utilizan como intérprete de comandos en la consola el *GNU Bourne-Again SHell* conocido como `bash`, sin embargo se pueden utilizar otros intérprete que incorporan nuevas características o mejoras como el caso de `zsh` [1] o `fish` [2].

1.1 El *shell* de *GNU/Linux*

El objetivo principal del shell es invocar o *lanzar* otro programa, sin embargo, suelen tener capacidades adicionales, tales como ver el contenido de los directorios, interpretar órdenes condicionales, trabajar con variables internas, gestionar interrupciones, redirigir entrada/salida, etc.

> El shell es una pieza de software que proporciona una interfaz para los usuarios en un sistema operativo y que provee acceso a los servicios del núcleo. Su nombre proviene de la envoltura externa de algunos moluscos, ya que es la "parte externa que protege al núcleo".

Entre los shells más populares (o históricos) en los sistemas ***nix** tenemos:

- **Bourne shell** (sh)
- **Almquist shell** (ash) o su versión en *Debian* (dash)
- **Bourne-Again shell** (bash)
- **Korn shell** (ksh)
- **Z shell** (zsh)
- **C shell** (csh) o la versión **Tenex C shell** (tcsh)

El *Bourne shell* ha sido el estándar *de facto* ya que fue distribuido en *Unix Version 7* publicado en 1977 y *Bourne-Again Shell (Bash)* es una versión mejorada del primero escrita por el proyecto **GNU** bajo *licencia GPL*, la cual se ha transformado en el estándar de los sistemas *GNU/Linux*.

Todos los ejemplos de esta unidad están basados en el uso de `bash`. Sin embargo es posible instalar cualquiera de los disponibles y utilizarlo. Para saber cuáles son los intérpretes disponibles en el sistemas se debe ejecutar el comando:

```
$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/dash
/bin/bash
/bin/rbash

$
```

El *shell* por defecto para un usuario se indica en el último campo de la línea correspondiente al usuario en el fichero `/etc/passwd`.

```
aswartz:x:1005:1006:Aaron Swartz,,,,Alumno:/home/aswartz:/bin/bash
```

Para conocer el *shell* activo se debe ejecutar el comando:

```
$ echo $SHELL
/bin/bash
$
```

Se puede utilizar cualquiera de los *shells* disponibles en el sistema ejecutando el binario correspondiente. Al salir, se vuelve al *shell* inicial. Para cambiar definitivamente el *shell* de usuario se debe actualizar el archivo `/etc/passwd` con el comando:

```
$ chsh -s /bin/dash
Contraseña:
$
```

Al iniciar una nueva consola se verifica que el nuevo *shell* es efectivo:

```
$ echo $SHELL
/bin/dash
$
```

Cada usuario sólo puede cambiar su propio *shell* y los usuarios administradores pueden cambiar el de todos los usuarios.

1.1.1 Comandos internos del shell

Existen un conjunto de comandos internos [3] al *shell*, es decir, integrados con el código de éste que, en el caso *Bourne* son:

```
:, ., break, cd, continue, eval, exec, exit, export, getopts, hash, pwd, readonly,
return, set, shift, test, [, times, trap, umask, unset.
```

Y además Bash incluye:

```
alias, bind, builtin, command, declare, echo, enable, help, let, local, logout,
printf, read, shopt, type, typeset, ulimit, unalias.
```

Cuando el intérprete *Bash* ejecuta un *script shell*, crea un proceso hijo que ejecuta otro *Bash*, el cual lee las líneas del archivo de a una línea por vez, las interpreta y ejecuta como si los comandos vinieran del teclado uno a uno.

El proceso *Bash* padre espera mientras el proceso *Bash* hijo ejecuta el *shell script* hasta el final, cuando el control vuelve al proceso padre, éste vuelve a poner el *prompt* nuevamente.

Un comando de *shell* será algo tan simple como `touch archivo1 archivo2 archivo3` que consiste en el propio comando seguido de argumentos, separados por espacios considerando `archivo1` el primer argumento y así sucesivamente.

1.1.2 Variables de usuario y de entorno

Existen dos tipos de variables, la que puede crear el usuario y asginarle un valor y las de entorno que ya están creadas al momento de iniciar el sistema y/o el *shell*, cuyo valor puede ser cambiado por el usuario.

1.1.2.1 Variables de usuario

Las variables de usuario son usadas mayormente dentro de los *scripts*, le otorgan una potencia importante al momento de manipular la información de contexto para tomar decisiones durante la ejecución de los *scripts*.

Los nombres de las variables consisten en letras, dígitos y guiones y se crean nombrándolas y asignándoles un valor en una sentencia del tipo `nombre_de_variable=valor`.

La variable existe mientras el proceso de *shell* en el que se creó esté activo, mantiene su espacio en memoria siempre y cuando su *shell* padre exista, no se pueden eliminar asignándole el valor `null`.

El alcance de una variable se refiere a la disponibilidad de la misma para otros procesos. Por defecto, las variables sólo son efectivas en el proceso que son definidas o sus procesos hijos. Así, una variable definida en un *shell script* es visible para los procesos hijos iniciados en el mismo y se denominan variables *privadas* o *locales*.

Para que una variable sea considerada *global* y esté disponibles para todos los procesos del *shell* se debe hacer explícita la definición mediante el comando `export`.

```
$ export directorio=/home/administrador/pruebas
$
```

Para hacer referencia a una variable se debe anteponer al nombre el caracter `$`, por ejemplo par ver el contenido de una variable se utiliza el comando `echo`:

```
$ echo $directorio
/home/administrador/pruebas
$
```

1.1.2.2 Variables de entorno

El funcionamiento y aspecto del *shell* son el resultado de las variables de entorno almacenadas en los archivos de configuración del sistema. Algunos de estos archivos son de configuración general como `/etc/profile` y `/etc/bash.bashrc`, y otros son personales de cada usuario, como `.profile` y `.bashrc`, alojados en el directorio `$HOME`

El listado de las variables de entorno definidas en el *shell* se pueden obtener indistintamente con dos comandos, `printenv` y `env`.

```
$ env
XDG_SESSION_ID=4
TERM=xterm
SHELL=/bin/bash
SSH_CLIENT=192.168.0.106 54641 22
SSH_TTY=/dev/pts/4
USER=administrador
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd
=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;4
4:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;
31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7
z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=0
1;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz
=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*
.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;3
1:*.cab=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm
=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:
*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=0
1;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*
.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;
35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=
```

```
01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.o
gv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36
:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=0
0;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
MAIL=/var/mail/administrador
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us
r/local/games:/snap/bin
PWD=/home/administrador
LANG=es_AR.UTF-8
SHLVL=1
HOME=/home/administrador
LANGUAGE=es_AR:es
LOGNAME=administrador
SSH_CONNECTION=192.168.0.106 54641 192.168.0.100 22
LC_CTYPE=es_AR.utf8
LESSOPEN=| /usr/bin/lesspipe %s
XDG_RUNTIME_DIR=/run/user/1000
LESSCLOSE=/usr/bin/lesspipe %s %s
_=/usr/bin/env

$
```

La diferencia entre ambos comandos es que en el caso de `printenv` se puede utilizar para conocer el valor de una variable de entorno particular:

```
$ printenv SHELL
/bin/bash
$ printenv USER
administrador
$
```

El comando `env` permite modificar el entorno en el que corren los programas pasándole un grupo de definiciones de variables en una línea de comando, por ejemplo:

```
$ env HOME="/usr/local/programa" comando_a_ejecutar opciones_comando
```

Como se indica antes, los procesos hijos heredan las variables de entorno del proceso padre, al usar `env` de esta manera, otorga la posibilidad de reemplazar o agregar nuevos valores al entorno del proceso hijo.

Para ver las variables del *shell* activo el comando indicado es `set` que, al ejecutarse sin parámetros muestra las variables configuradas y las funciones *bash* definidas. Normalmente la extensión de la salida de este comando es muy grande.

1.1.2.3 Variables comunes

Algunas de las variables de entorno y del *shell* se utilizan con mucha frecuencia. Algunas de ellas son:

SHELL

Describe el *shell* activo que interpreta los comandos, en la mayoría de los casos será `bash`.

TERM

Especifica el tipo de terminal que se emula cuando corre el *shell*.

USER

El usuario en actualmente en sesión.

PWD

El directorio de trabajo actual.

OLDPWD

El directorio de trabajo anterior. El *shell* mantiene la información para poder regrear al directorio anterior con el comando: `cd -`.

MAIL

La ubicación del buzón de correo del usuario actual.

PATH

Un listado de directorios que el sistema utilizará para buscar los comandos.

LANG

El lenguaje y configuración de localización actual, incluyendo la codificación de caracteres.

HOME

El directorio de inicio del usuario actual.

_

El último comando ejecutado.

Además de estas variables de entorno, también están algunas variables del *shell* muy útiles:

BASH_VERSION

La versión de `bash` que se está ejecutando en formato legible por un humano.

BASH_VERSINFO

La versión de `bash` en formato válido para leer con máquina.

HOSTNAME

El nombre de **host** de la computadora.

IFS

Siglas de *Internal Field Separator* para separar la entrada en la línea de comando. Por defecto es un espacio en blanco.

UID

El *UID* del usuario actual.

1.2 Redireccionamiento y tuberías

En los sistemas ***nix** cada proceso puede tener tres descriptores de archivos del estándar *POSIX* que, a modo de canales, conectan la entrada y salida *(I/O)* de un comando o aplicación con la terminal o consola cuando se ejecuta:

- Entrada estándar **stdin** (standard input)
- Salida estándar **stdout** (standard output)
- Error estándar **stderr** (standard error)

Estos canales se utilizan para enviar y recoger los datos que surgen de la ejecución de un *script*, comando o aplicación y se representan con un número entero, el número `0` representa a **stdin**, el `1` a **stdout**, y el `2` a **stderr**.

En la mayoría de los *shells*, incluso en *Bash*, se puede redirigir **stdout** y **stderr** a un archivo.

Por ejemplo, para enviar la salida estándar del comando `ls` a un archivo el comando es:

```
$ ls -l > directorio.txt
```

Este comando creará un archivo llamado `directorio.txt`, cuyo contenido será lo que se vería en la pantalla si se ejecutara el comando `ls -l` solamente.

1.2.1 Entrada estándar (stdin)

El canal **stdin** o *standard input* dirige los datos que son enviados al programa, puede ser el usuario de terminal o la salida de datos de un programa.

En un script en **perl**, se solicita la interacción del usuario mediante *STDIN*, por ejemplo en este caso, se recoge mediante el uso de *STDIN* el valor de dos variables que el usuario escribe:

```
#!/usr/bin/perl
print "Introduce tu nombre:\n";
my $nombre = <STDIN>;
print "Introduce tu apellido:\n";
my $apellido = <STDIN>;
```

Es posible redireccionar la entrada estándar de un comando para que su procedencia sea de un archivo y no desde el teclado. En la ausencia de argumentos de nombres de archivos, muchos comandos y utilitarios de *GNU/Linux* leen su entrada desde el **STDIN**; que en la mayoría de los casos es el teclado asignado a la terminal.

La forma para redireccionar el **STDIN** es la siguiente:

```
$ wc -l < usuarios_nuevos.txt
2
$
```

En el ejemplo, el comando `wc -l` cuenta las líneas del **STDIN** que en este caso es el archivo `usuarios_nuevos.txt`.

Redireccionar la entrada estándar desde un archivo permite escribir todos los argumentos de entrada previamente, revisar y corregir errores, así como poder reusar el archivo repetidamente.

1.2.2 Salida estándar (stdout)

A través de la salida estándar, **stdout** se reciben los datos que vuelca el comando o programa durante su ejecución, por ejemplo, el resultado de los comandos `ls`, `cat` o cualquier otro comando de terminal.

En cambio, hay otros comandos o programas que no muestran salida, a no ser que se le especifique, como por ejemplo mover o copiar ficheros.

En el ejemplo del *script* de **perl**, si bien cada comando `print` opera sobre la salida estándar, se puede agregar un comando que muestre por la **stdout** la información ingresada por el usuario:

```
#!/usr/bin/perl
print "Introduce tu nombre:\n";
my $nombre = <STDIN>;
print "Introduce tu apellido:\n";
my $apellido = <STDIN>;
chop $nombre;
print "Hola $nombre $apellido!";
```

Al ejecutar el comando `ls` en un directorio, la salida de datos (stdout) se muestra en la terminal:

```
$ ls
binarios   Documentos  Imágenes  Plantillas  Respaldo  ttyrecord  Vídeos
Descargas  Escritorio  Música    Público     test.pl   usuarios_nuevos.txt
$
```

Para manipular la salida estándar, se utiliza el símbolo de redirección `>`. Siguiendo el ejemplo, para redirigir la **stdout** a un archivo de texto, la expresión del comando es:

```
$ ls > listado.txt
```

Al redirigir la salida estándar, no aparece el resultado del comando en pantalla, sino que se ha almacenado en el archivo indicado:

```
$ cat listado.txt
binarios
Descargas
Documentos
Escritorio
Imágenes
listado.txt
```

```
Música
Plantillas
Público
Respaldo
test.pl
ttyrecord
usuarios_nuevos.txt
Vídeos
$
```

1.2.3 Error estándar (stderr)

A través del canal **stderr** los programas o comandos envían el informe de error de la ejecución de un comando. En los *shell scripts* o comandos se puede combinar el uso de **stdout** y **stderr** para separar la salida estándar de los errores, almacenarlos en registros independientes o manipularlos por separado. Por defecto la salida de errores se vuelca en la terminal.

Ejemplo con parámetros incorrectos:

```
$ ls --este_parametro_no_existe
ls: opción no reconocida «--este_parametro_no_existe»
Pruebe 'ls --help' para más información.

$
```

Para guardar la salida del canal **stderr** se vuelcan en un archivo redireccionándo la salida. Para realizar la redirección se debe utilizar el número `2`, asociado a la **stderr**, seguido del símbolo de redirección `>`.

Si quisieramos almacenar los errores de la ejecución del comando en caso de suceder, podemos volcarlos usando en lugar del símbolo `>` el número `2` seguido de `>`:

```
$ ls --este_parametro_no_existe 2> error.log
```

Los errores ya no aparecen en pantalla, son guardados en el archivo `error.log`:

```
$ cat error.log
ls: opción no reconocida «--este_parametro_no_existe»
Pruebe 'ls --help' para más información.

$
```

1.2.4 Combinación de salidas y opciones especiales

Se puede redirigir tanto la **stderr** como la **stdout** a archivos separados para tener un registro de los resultados del comando:

```
$ cd /home
$ ls -R > $HOME/salidals.txt 2> $HOME/erroresls.txt
$ cd
```

Entonces el contenido de la salida estándar **stdout** se guarda en el archivo `salidals.txt`:

```
$ cat salidals.txt
.:
administrador
alovelace
aswartz
aturing
lmartinez
lost+found
ltorvalds

./administrador:
binarios
Descargas
Documentos
erroresls.txt
error.log
Escritorio
Imágenes
listado.txt
Música
Plantillas
Público
Respaldo
salidals.txt
test.pl
ttyrecord
usuarios_nuevos.txt
Vídeos

./administrador/Descargas:
google-chrome-stable_current_amd64.deb
vivaldi-stable_1.4.589.11-1_amd64.deb

./administrador/Documentos:
trabajo_práctico_2

./administrador/Documentos/trabajo_práctico_2:
Administración
DeptoTI
Personal
Ventas

./administrador/Documentos/trabajo_práctico_2/Administración:

./administrador/Documentos/trabajo_práctico_2/DeptoTI:

./administrador/Documentos/trabajo_práctico_2/Personal:

./administrador/Documentos/trabajo_práctico_2/Ventas:

./administrador/Escritorio:

./administrador/Imágenes:

./administrador/Música:
```

```
./administrador/Plantillas:

./administrador/Público:

./administrador/Respaldo:
201607

./administrador/Respaldo/201607:
Documentos

./administrador/Respaldo/201607/Documentos:
classified.pdf
confidential.pdf
default.pdf
default-testpage.pdf
form_english.pdf
form_russian.pdf
secret.pdf
standard.pdf
topsecret_copia_respaldo.pdf
topsecret.pdf
unclassified.pdf

./administrador/Vídeos:

./alovelace:

./aswartz:

./aturing:

./lmartinez:

./ltorvalds:

$
```

Y el contenido del canal error estándar **stderr** se guarda en el archivo `erroresls.txt`:

```
$ cat erroresls.txt
ls: no se puede abrir el directorio './lost+found': Permiso denegado

$
```

Una opción interesante es la posibilidad de redirigir tanto la **stderr** como la **stdout** al mismo sitio, en lugar de especificar dos veces el mismo destino se utiliza el modo `2>&1`.

```
$ cd /home
$ ls -R > $HOME/salidalsfull.txt 2>&1
$ cd
```

En este caso el archivo incluye la información de ambas salidas.

```
$ cat salidalsfull.txt
.:
```

```
administrador
alovelace
aswartz
aturing
lmartinez
lost+found
ltorvalds

./administrador:
binarios
Descargas
Documentos
erroresls.txt
error.log
Escritorio
Imágenes
listado.txt
Música
Plantillas
Público
Respaldo
salidalsfull.txt
salidals.txt
test.pl
ttyrecord
usuarios_nuevos.txt
Vídeos

./administrador/Descargas:
google-chrome-stable_current_amd64.deb
vivaldi-stable_1.4.589.11-1_amd64.deb

./administrador/Documentos:
trabajo_práctico_2

./administrador/Documentos/trabajo_práctico_2:
Administración
DeptoTI
Personal
Ventas

./administrador/Documentos/trabajo_práctico_2/Administración:

./administrador/Documentos/trabajo_práctico_2/DeptoTI:

./administrador/Documentos/trabajo_práctico_2/Personal:

./administrador/Documentos/trabajo_práctico_2/Ventas:

./administrador/Escritorio:

./administrador/Imágenes:

./administrador/Música:

./administrador/Plantillas:

./administrador/Público:
```

```
./administrador/Respaldo:
201607

./administrador/Respaldo/201607:
Documentos

./administrador/Respaldo/201607/Documentos:
classified.pdf
confidential.pdf
default.pdf
default-testpage.pdf
form_english.pdf
form_russian.pdf
secret.pdf
standard.pdf
topsecret_copia_respaldo.pdf
topsecret.pdf
unclassified.pdf

./administrador/Vídeos:

./alovelace:

./aswartz:

./aturing:

./lmartinez:
ls: no se puede abrir el directorio './lost+found': Permiso denegado

./ltorvalds:

$
```

Otra opción es redirigir tanto **stdout** como **stderr** directamente a la *basura* utlizando el archivo especial `/dev/null`. De esta manera no se genera actividad en la pantalla logrando una ejecución silenciosa, muy útil en la ejecución de comandos dentro de un *shell script*.

```
$ ls -l > /dev/null 2>&1
```

Es normal redireccionar la salida a un archivo existente, cualquier contenido de éste archivo será totalmente eliminado si utilizamos el símbolo `>`. Cuando es necesario preserver el contenido del archivo, registros de *logs* por ejemplo, se debe utilizar el símbolo `>>` que le indica al comando que debe agregar la información al continuación del contenido actual del archivo.

1.2.5 Tuberías (pipes)

Las tuberías son como la redirección, pero, trabajan un poco diferente a ellas. Las tuberías permiten que los flujos de *I/O* (entrada/salida) de una serie de procesos sean conectados, encadenando de esta forma los comandos de una tubería a otra.

El carácter de tubería `|` le indica al *shell* que conecte la salida estándar **stdout** del comando a la izquierda, a la entrada estándar **stdin** del comando en la derecha.

Por ejemplo, para buscar en el fichero `/etc/passwd` todas las líneas que contengan la palabra *Alumno* se debe utilizar el comando `cat` para mostrar el archivo `/etc/passwd` y pasarle el resultado al comando `grep` para que este busque las coincidencias.

```
$ cat /etc/passwd | grep Alumno
ltorvalds:x:1003:1003:Linus Torvalds,,,,Alumno:/home/ltorvalds:/bin/bash
alovelace:x:1002:1002:Ada Lovelace,,,,Alumno:/home/alovelace:/bin/bash
aturing:x:1004:1004:Alan Turing,,,,Alumno:/home/aturing:/bin/bash
aswartz:x:1005:1006:Aaron Swartz,,,,Alumno:/home/aswartz:/bin/bash

$
```

Los comandos se pueden encadenar unos a otros realizando operaciones sobre la **stdout** del comando anterior. Por ejempo, si lo que se necesita es saber es la cantidad de usuarios que son *alumnos* se ejecuta lo siguiente:

```
$ cat /etc/passwd | grep Alumno | wc -l
4

$
```

1.3 Creación de shell scripts

Como todos los antecedentes de *UNiX*, *GNU/Linux* tiene su propia manera de agrupar comandos en un archivo ejecutable, conocido en el mundo *UNiX* como *Scripts del Shell*. Éstos *scripts* juegan un rol esencial, por la característica de multitareas del *shell* de *GNU/Linux*, que puede llevar a cabo muchas cosas simultáneas, y por ende complejas, desde *shell scripts* que sirven en el rol de guión para orquestar, monitorear y coordinar toda esta complejidad.

En esencia y funcionalidad, un *shell script* desempeña el mismo papel que cualquier otro archivo ejecutable. Este puede leer desde la entrada estándar **STDIN**, escribir a la salida estándar **STDOUT** y al error estándar **STDERR**. También es posible abrir y cerrar archivos.

Se puede abrir una base de datos, editarla, y cerrarla. En definitiva, toda tarea que se pueda llevar a cabo desde la línea de comandos, se puede efectúar desde un *shell script*. Los *scripts* son muy usados en las partes más criticas de los sistemas *GNU/Linux*. La inicialización del sistema, inmediatamente después del gestor de arranque, es manejada mayormente por *shell scripts*. El inicio del Sistema **X Window** es normalmente manejado por un *shell script*.

Los *Scripts de Shell* son, simples archivos de texto, normalmente creados por editores de texto como `vi`, `emacs`, `nano` o por cualquier flujo de texto que produzca un archivo. Así, que un *shell script*, puede generar, y luego ejecutar, otro *shell script* en tiempo de ejecución, y luego eliminarlos después que ya no sean necesitados.

Un *shell script* es un archivo de texto que contiene comandos de *shell*. El intérprete **Bash** lee y ejecuta los comando del archivo y sale, este modo operativo se denomina *shell no-interactivo*.

Para hacer ejecutable el archivo del *shell script* se debe asignar el permiso de ejecución al archivo utilizando el comando `chmod`.

```
$ chmod +x mi_script.sh
```

Crear archivos scripts desde cero, involucra cuatro pasos:

1. Identificar qué es lo que desea efectuar con el *shell script*.
2. Determinar con cuál o cuales comandos efectúa la acción deseada.
3. Usar un editor para crear la secuencia de comandos en un archivo de texto.
4. Usando el comando `chmod` asignarle los *bits* de permisos de ejecución al archivo.

Algunas consideraciones de buenas prácticas al momento de desarrollar *shell scripts*:

- Un *shell script* tiene que ejecutarse sin errores.
- Tiene que realizar la tarea para la que fue creado.
- La lógica del programa debe ser clara y bien definida.
- Un *shell script* no realiza tareas innecesarias.
- Un *shell script* tiene que ser reutilizable.
- Responder algunas preguntas guía:
    - ¿Se necesita información del usuario o del entorno de éste?
    - ¿Cómo se va a guardar esa información?
    - ¿Hay que crear archivos? ¿Dónde y con qué permisos?
    - ¿Qué comandos se usarán? Si se va a usar el *shell script* en sistemas diferentes, ¿están disponibles estos comandos?
    - ¿Hay que notificar al usuario? ¿cuándo y por qué?

La estructura de un *shell script* es muy flexible. Sin embargo, se debe asegurar una lógica correcta, control de flujo y eficiencia para que los usuarios puedan utilizarlo más fácil.

1.3.1 Ejecución de un shell script

Cuando **Bash** ejecuta el *shell script* asigna al parámetro especial `0` el nombre del archivo y al resto de los parámetros posicionales les asigna en los argumentos dados, si es que existen. Un *shell script* puede ejecutarse llamando al intérprete y pasando el *shell script* y sus argumentos como argumentos de **Bash** o usando el nombre del *shell script* si tiene el *bit* de ejecución, así las opciones son:

```
$ bash mi_script.sh argumento1 argumento2 argumentoN

$ mi_script.sh argumento1 argumento2 argumentoN
```

Si la primer línea del *shell script* comienza con los caracteres `#!` el resto de la línea indica el intérprete de comandos a utilizar, ya sea **Bash**, **awk**, **Perl** u otro intérprete disponible en el sistema, y continuar el *script* en el lenguaje correspondiente.

Los *shell scripts* de **Bash** comienzan con `#!/bin/bash` asegurando que éste sea el intérprete sin importar si se ejecutan bajo el entorno de otro *shell*.

1.3.2 Aspectos generales

Si la línea no es un comentario, es decir (dejando de lado los espacios en blanco y tabulador) la cadena no comienza por `#`, el *shell script* lee y lo divide en palabras y operadores, que se convierten en comandos, operadores y otras construcciones. A partir de este momento, se realizan las expansiones y sustituciones, las redirecciones, y finalmente, la ejecución de los comandos.

En **Bash** se pueden tener funciones, que son una agrupación de comandos que se pueden invocar posteriormente. Y cuando se invoca el nombre de la *función de shell*, como un nombre de comando simple, la lista de comandos relacionados con el nombre de la función se ejecutará teniendo en cuenta que las funciones se ejecutan en el contexto del *shell script* en curso sin crear ningún proceso nuevo para ello.

Un *parámetro* es una entidad que almacena valores, que puede ser un nombre, un número o un valor especial. Una *variable* es un parámetro que almacena un valor.

Por último, existen un conjunto de expansiones que realiza el *shell* que se lleva a cabo en cada línea y que pueden ser enumeradas como: de tilde/comillas, parámetros y variables, substitución de comandos, de aritmética, de separación de palabras y de nombres de archivos.

1.3.3 Ejemplos de script

Para construir el archivo **PDF** con el texto de los apuntes de la materia se utiliza el *shell script* `crea_pdf.sh` logrando que no sea necesario recordar los parámetros del comando `rst2pdf` necesarios.

```
#!/bin/bash

rst2pdf ${1}.rst ../ADM-GNU-LINUX-I/${1}.pdf -s estilos_propios/tusl2.style --re
peat-table-rows -e preprocess
```

Se observa el uso del parámetro `$1` donde el `1` está entre llaves para separarlo de la cadena de caracteres contigua. Este parámetro toma el valor del primer argumento que acompaña al comando cuando se lo ejecuta:

```
$ crea_pdf.sh Unidad_V
```

La carátula del apunte, que se realiza a partir de un documento *SVG* editado con el programa *inkskape*, también es un archivo **PDF** que se debe unir al documento del apunte. Para ello se utiliza el comando `gs` con una serie de parámetros que, al igual que los del comando `rst2pdf`

sería engorroso recordar y muy fácil cometer errores al escribirlos cada vez que se genera el documento. La solución es utilizar un *shell script* similar:

```
$ cat armapdf.sh
#!/bin/bash

gs -q -sPAPERSIZE=a4 -dNOPAUSE -dBATCH -sDEVICE=pdfwrite -sOutputFile=Unidad_${1
}_Final.pdf Unidad_${1}_caratula.pdf Unidad_${1}.pdf
```

Y la forma de ejecutarlo es:

```
$ armapdf.sh V
```

En el ejemplo siguiente se muestra un caso sencillo de obtención de información del sistema.

Para ver el contenido se utiliza el comando `cat`, el uso del parámetro `-n` permite visualizar el número de línea de cada comando.

```
$ cat -n info.sh
1 #!/bin/bash
2 clear; echo "Información dada por el shell script info.sh."
3 echo "Hola, $USER"
4 echo
5 echo "La fecha es `date`, y esta semana `date +"%V"`."
6 echo
7 echo "Usuarios conectados:"
8 w | cut -d " " -f 1 - | grep -v USER | sort -u
9 echo
10 echo "El sistema es `uname -s` y el procesador es `uname -m`."
11 echo
12 echo "El sistema está encendido desde hace:"
13 uptime
14 echo
15 echo "¡Esto es todo amigos!"
```

La línea *1* corresponde a la definición del intérprete.

La línea *2* es una combinación de dos comandos separados por el caracter `;`. El comando `clear` limpia la pantalla del terminal y luego el comando `echo` muestra en la *stdout* el texto indicado entre comillas.

En la línea *3* el comando `echo` muestra en pantalla el texto entre comillas que será *expandido* con el valor de la variable **$USER**.

Las líneas *4*, *6*, *9*, *11* y *14* ejecutan el comando `echo` sin argumentos, esto hace que en la terminal se muestre una línea en blanco utilizada para separar el resultado del resto de los comandos del *shell script*.

En la línea *5* se muestra en pantalla un texto que, en este caso, se *expande* con el resultado de las llamadas al sistema que ejecutan el comando `date`. En la línea *10* sucede lo mismo con el comando `uname`.

Las líneas *7*, *12* y *15* hacen un uso simple del comando `echo` mostrando el texto entre comillas.

La línea *8* es la más interesante del *shell script* ya que utiliza *tuberías* para redireccionar la salida de un comando hacia la entrada de otro. El comando `w` lista los usuarios conectados al sistemas, esa información se pasa al comando `cut` que separa en campos delimitados por un espacio y selecciona el primer campo. El resultado se pasa al comando `grep` que selecciona todas las líneas y descarta aquellas que tengan la cadena `USER` para luego pasar la información al comando `sort` que muestra el listado de usuarios eliminando las coincidencias para que aparezcan una sola vez.

En la línea *13* se ejecuta el comando `uptime` directamente.

La ejecución del *shell script* presenta el siguiente resultado:

```
Información dada por el shell script info.sh.
Hola, leonardo

La fecha es mié sep 21 10:32:42 ART 2016, y esta semana 38.

Usuarios conectados:

leonardo

El sistema es Linux y el procesador es x86_64.

El sistema está encendido desde hace:
 10:32:42 up  2:02,  3 users,  load average: 0,23, 0,27, 0,20

¡Esto es todo amigos!
```

2 Bibliografía

- Páginas de Manual (Man Pages) del sistema operativo.

La bibliografía que se indica a continuación corresponde a material que provee una visión interesante sobre los temas propuestos en esta unidad. Es documentación más completa e incluso más extensiva en el desarrollo de algunos temas.

- The Debian Administrator's Handbook, Raphaël Hertzog and Roland Mas
- Administración de sistemas GNU/Linux, Máster universitario en Software Libre, Universitat Oberta de Catalunya
- Básicamente GNU/Linux, Antonio Perpiñan, Fundación Código Libre Dominicano
- Manual de Referencia de Bash (Bash Reference Manual)
- How To Read and Set Environmental and Shell Variables on a Linux VPS, Justin Ellingwood
- Bash Guide for Beginners, Machtelt Garrels

---

1 http://www.zsh.org/
2 https://fishshell.com/
3 Consultar la página de manual para más información y descripción más completa.