# Sistemas Operativos
# 2do Semestre 2010

## Bourne Again Shell

**Wenceslao Palma M.**
**<wenceslao.palma@ucv.cl>**

# Shell - Intro

Shell es el intérprete de comandos.

Mas allá de ser el “intermediario” entre el sistema operativo y el usuario puede ser considerado como un lenguaje de programación.

Comandos, utilidades y herramientas pueden ser invocados por un shell script.

Cuando lo anterior no es suficiente es posible potenciar un script con estructuras de control.

Un script es interpretado.

Script están presentes en el proceso de booteo.

bash es “popular” ya que es el shell GNU.

Fue desarrollado conforme a IEEE POSIX P1003.2/ISO 9945

```
#! /bin/bash
#
# network       Bring up/down networking
#
# chkconfig: 2345 10 90
# description: Activates/Deactivates all network interfaces configured to \
#              start at boot time.
#
### BEGIN INIT INFO
# Provides: $network
### END INIT INFO

# Source function library.
. /etc/init.d/functions

if [ ! -f /etc/sysconfig/network ]; then
    exit 0
fi

. /etc/sysconfig/network
```

**Lineas del script /etc/rc.d/rc5.d/S10network**

## Cuando no usar shell-scripts

Tareas en donde el uso de recursos es intensivo (ordenamiento, hashing, archivos, etc.)

Cuando se necesitan arreglos multidimensionales, listas enlazadas, árboles.

Cálculo intensivo.

En su versión más sencilla un script puede ser solo una lista de comandos:

```
#!/bin/bash
ls -la | grep ^d
```

Lo cual resulta en:

```
drwxr-xr-x   8 wpalma wpalma   4096 Sep  15 13:12 .
drwx------   26 wpalma wpalma   4096 Sep  15 13:12 ..
drwxr-xr-x   2 wpalma wpalma   4096 May 30 15:00 docs
drwxr-xr-x   8 wpalma wpalma   4096 May 30 15:00 icons
drwxr-xr-x   2 wpalma wpalma   4096 May 30 15:01 lang
drwxr-xr-x 12 wpalma wpalma   4096 May 30 15:01 plugins
drwxr-xr-x   4 wpalma wpalma   4096 May 30 15:10 skins
drwxr-xr-x   2 wpalma wpalma   4096 May 30 15:02 utils
```

**#!** es un “número mágico” indica al sistema que el archivo es un shell script
**/bin/bash** indica cual es el programa que interpreta los comandos del script

Para ejecutar el script las alternativas son:

bash nombre_script
chmod 555 nombre_script y luego ./nombre_script

En el último caso los permisos pueden ser más restrictivos.

Para saber cuales son los shells disponibles en el sistema:

$more /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/bin/bash2
/bin/ash
/bin/bsh
/bin/tcsh
/bin/csh

Para los usuarios del sistema el shell por defecto se encuentra en **/etc/passwd**

wpalma:x:500:500:Wenceslao Palma M.:/home/wpalma:**/bin/bash**

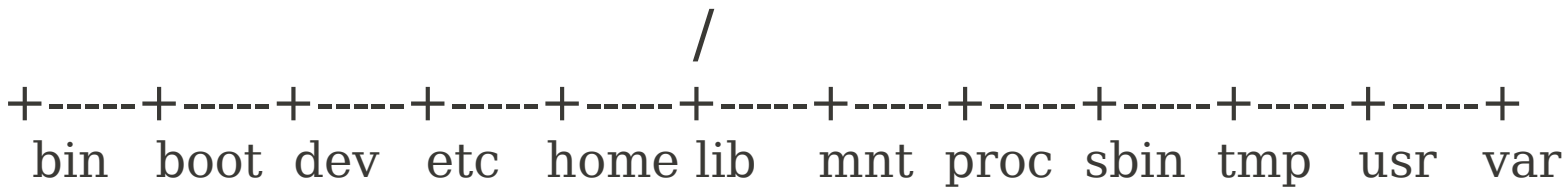
# El ambiente de bash

el ambiente de ejecución de comandos  y scripts es afectado por archivos de configuración y variables de entorno.

/etc/profile
.bash_profile
MAIL, USER, HOSTNAME, PATH, etc.

## Un vistazo al sistema de archivos

Util conocerlo antes de programar algún script.




| | |
|---|---|
| /bin | binarios necesarios para el booteo y adm. en modo monousuario. |
| /boot | contiene la imagen del kernel y archivos de config. para el booteo |
| /dev | archivos utilizados para el acceso a dispositivos |
| /etc | archivos de configuración del sistema |
| /home | directorio de trabajo para los usuarios |
| /lib | bibliotecas del sistema |
| /mnt | puntos de montaje para dispositivos, particiones |
| /proc | pseudo filesystem (reside en RAM), importante nexo con el kernel |
| /sbin | binarios esenciales para administrar el sistema |
| /tmp | almacena archivos temporales |
| /usr | compiladores, documentación, bibliotecas, etc. utilizadas por los usuarios |
| /var | almacena datos con alta volatilidad, básicamente logs y mails, etc. |

## Comandos comunes del sistema de archivos

**ls** : listar nombres de todos los archivos ubicados en el directorio actual.
los modificadores permiten obtener más información de los archivos que se mostrarán.
ls -t, ls -l, ls -u, ls -lt, ls -la.....

**cp** *file1 file2*: copiar file1 en file2, sustituye el contenido de file2 si existe.

**mv** file1 file2: mover file1 a file2, sustituye file2 si existe.

**rm** file(s): borrar archivos (no hay vuelta atrás).

**cat** file(s): mostrar el contenido de un archivo.

**wc** file: contar lineas, palabras y caracteres de un archivo.

**grep** pattern file: imprime lineas que coinciden con pattern ( -v para el opuesto)

**tail** file: mostrar las últimas 10 líneas del archivo. tail -f útil para mirar logs :-)

Todos los comandos soportan el uso de metacaracteres.

## Redirección de entrada-salida

En general un comando produce una salida dirigida hacia la pantalla. Sin embargo, es posible redireccionar la salida de un comando hacia un archivo (recordar que los dispositivos también son archivos :-) )

$who > salida.txt
redirecciona la salida del comando who hacia el archivo salida.txt. El símbolo > significa redireccionar la salida en un archivo en vez de la pantalla.

$cat texto.txt firma.txt >> carta.txt
También se puede usar >> lo cual significa “anexa al final de”.

$wc -l < temp
El símbolo < significa tomar entrada desde un archivo y no desde el terminal.

Los comandos $wc -l < temp y $ wc -l temp otorgan la misma salida, cual es la diferencia?

# Pipes

Cuando se ocupa < y > para redireccionar, uno de los participantes es un archivo.

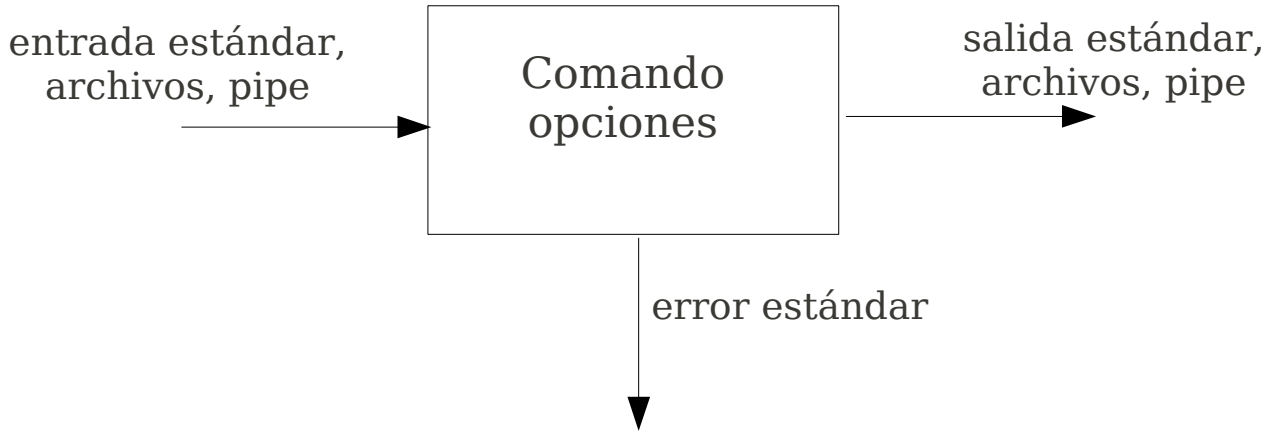
Lo anterior a la hora de combinar comandos no es muy cómodo.

$who > salida.txt
$sort < salida.txt

Cuando se desea relacionar la salida de un comando con la entrada de otro sin la utlización de un archivo se utilizan pipes (|).

$ who | sort

$ps -ef | grep httpd | wc -l



# Comandos Múltiples

## Ejecución secuencial

Para ejecutar varios procesos en forma secuencial se debe utlizar ;

$ls -la carta.txt ; clear ; vi carta.txt

## Ejecución simultánea (background)

Para ejecución simultánea de comandos se utiliza &, la idea es invocar comandos indicando al shell que no espere a que el comando termine.

$lpr tarea.pdf &

Algunos comandos demoran bastante en terminar, por ejemplo una llamada a wget. En este caso es deseable cerrar la sesión e ir a tomar un cafecito :-), pero al cerrar la terminal wget termina abruptamente su ejecución (por qué?).

$nohup wget..... &

Cualquier salida del comando se redirecciona a nohup.out

# Estructuras de Control

## if (ejecución condicional)

```
if [test]; then
    .....
fi
```

Para testear la existencia de un archivo antes de ejecutar algún comando que lo involucre:

```
if [ -a tarea.txt ]; then
    cat tarea.txt;
fi
```

Algunos otros tests relacionados con archivos son:

-d : el archivo existe y es un directorio
-r : el archivo existe y tiene permiso de lectura
-x : el archivo existe y es ejecutable

Testing de strings

==, !=, >, <

Testing numérico

-eq, -lt, -le, -gt, -ge

## if anidados

```
#!/usr/bin
#año bisiesto
$year=`date +%Y`
if  [ $[$year % 4 ] -eq 0 ]; then
      echo “bisiesto”
elif  [ $[$year % 100 ] -eq 0 ]; then
            if [ $[$year % 400 ] -eq 0 ]; then
                  echo “no es bisiesto”
            else
                  echo “es bisiesto \n”
            fi
else
      echo “no es bisiesto\n”
fi
```

## Argumentos desde la línea de comandos

$1,$2,$3,..... variables que contienen los argumentos.
$0 nombre del script
$# cantidad de argumentos

**for**

```
for i in `ls`; do
    if [ $i == pepe.txt ]; then
        cat $i;
    fi
done
```

# Referencias

El Entorno de Programación Unix.
Advanced Bash-Scripting Guide