BASH

THE BOURNE-AGAIN SHELL

Angel Luis Calvo

ESCUELA IT

En esta sesión:

- Qué es Bash
- Comandos y ejecución
- Redirecciones
- Fundamentos de scripts
- Buenas prácticas
- Variables
- Chequeo

Angel Luis Calvo

Angel Luis Calvo

angelonx@gmail.com

ESCUELA IT

Qué es bash

Angel Luis Calvo

ESCUELA IT

Qué es bash

Es un tipo de *shell* que solemos encontrar en sistemas operativos de base Linux.
A un shell podemos llamarlo “línea de comandos” o “intérprete de comandos”
Nos servirá para introducir comandos del sistema operativo de forma textual, para ejecutar cualquier clase de tarea.
Ejemplo:
Para crear un directorio para pruebas de este curso:

```
mkdir /home/usuario/curso
```

Qué es bash, un poco de historia

Los shells tienen su origen en el sistema Unix. De él deriva Linux.
Bash es el intérprete de comandos más común en los sistemas Linux de la actualidad.
Existen otros, como:

- sh → es de los primeros, creado por Steve Bourne. La evolución para software libre sería bash.
- csh → originario de Berkeley, versión libre tcsh.
- ksh → ATT Korn shell.
- dash → muy ligero, pero con menos características.

Qué es bash, un poco de historia

Más actuales:

- zsh → un poco más funcional que bash, más predictivo, compatible con bash.
- fish → con más colores y más ayudas, no compatible.

Hay muchos más shells.

Hay que tener en cuenta el shell que se use para los scripts que se quieran utilizar.

Qué es bash, un poco de historia

POSIX: Portable Operating System Interface uniX. Es un conjunto de reglas y estándares que pretenden garantizar la interoperabilidad entre sistemas Unix. Evolucionó a Single Unix Specification (SUS)
Puede consultarse información sobre estas reglas en: The Open Group

Todos los sistemas Linux cumplen, pero no al 100%, con estas normas. Ojo a esto, puede que haya comandos con opciones no válidas según el entorno de trabajo.

Qué es bash, un poco de historia

Para saber qué intérprete de comandos estamos usando, podemos consultar la variable de entorno SHELL:

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

O mejor, pero solo en línea de comando, no en un script:

```
echo $0
/bin/bash
```

Qué es bash, ventajas

¿Por qué utilizar bash?

- Puede ser una manera rápida de hacer algo en el sistema operativo.
- Hay veces que no podemos usar entorno gráfico:
  - Accediendo por terminales remotas (SSH)
  - En modo de rescate, cuando hay problemas.
  - En servidores, por ahorro de recursos.
- Se puede utilizar para crear y ejecutar scripts
  - Para hacer más rápidas ciertas tareas.
  - Para automatizar procesos.

Qué es bash: archivos de config

Hay algunos archivos de sistema que ayudan a configurar bash:

/etc/profile → archivo general del sistema, con variables de entorno.
/etc/bashrc → archivo general, que solo afecta a quien use bash.
En el home de cada usuario:
.bash_profile → indica a bash que lea otros archivos o se configura todo aquí, se ejecuta al inicio de sesión.
.bashrc → se ejecuta al arrancar una consola de bash.
Hay más archivos, para detallar cosas al iniciar o cerrar sesión, por ejemplo.

Comandos y ejecución

Angel Luis Calvo

ESCUELA IT

Comandos y ejecución

El shell lo utilizamos para ejecutar comandos, programas y scripts.
Se le llama intérprete de comandos porque precisamente hace eso; **interpreta** lo que se escribe en la línea de comandos y si puede entenderlo, lo **ejecuta**.
Para poder cumplir con esto debe poder encontrar el comando correspondiente.
Pueden darse dos condiciones:

- Que el comando esté cargado en memoria.
- Que el comando haya que buscarlo en el disco.

Comandos y ejecución

Cuando el comando está en memoria se ejecuta directamente, tiene más prioridad. Se le llama comando **interno**.
Cuando está en disco se carga en memoria y luego se ejecuta. Se le llama comando **externo**. Pruebe el comando type para averiguar de qué tipo es.
A la hora de buscarlo en disco, bash consulta la variable de entorno PATH, donde figuran los directorios para realizar la búsqueda.
Ejemplo:

```
echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/sbin: ...
```

Comandos y ejecución: variables de entorno

Las variables de entorno configuran unos parámetros de sistema comunes, muy usados por las aplicaciones, como el *shell* por defecto y el *path*. Para verlas podemos ejecutar printenv.

Ejercicio cero:
Borrar el PATH y probar qué pasa

```
PATH=
echo $PATH
```

Angel Luis Calvo

Comandos y ejecución: permisos

En Linux, sobre los archivos, hay **tres** tipos de permisos básicos.

- Lectura (r)
- Escritura (w)
- Ejecución (x)

Esos permisos se aplican de tres en tres, sobre el propietario, el grupo y el resto de usuarios, respectivamente.
Para verlos:

```
ls -l
– rwx r-x r-x 1 root root may 10 pruebas.sh
```

Comandos y ejecución: permisos

El permiso de ejecución es **imprescindible** para poder ejecutar cualquier comando, programa o script.

Los directorios necesitan ese mismo permiso para poder recorrerlos.

```
ls -l
drwx r-x r-x 1 root root oct 12 Descargas
```

Comandos y ejecución: permisos

Por defecto, normalmente y dependiendo de la distribución, los permisos de un archivo al crearlo son:

rw- r-- r--

Y de un directorio:

rwx rwx rwx

Para modificar permisos usamos **chmod**
El propietario y root siempre pueden cambiar los permisos.

Comandos y ejecución: permisos

Ejemplos con **chmod**
Damos control total al propietario: chmod u+rwx prueba.sh
Damos control total al grupo: chmod g+rwx prueba.sh
Le quitamos permisos al resto: chmod o-rwx prueba.sh
Lo mismo usando binario: chmod 770 prueba.sh

Para más info: https://blog.desdelinux.net/permisos-y-derechos-en-linux/

Comandos y ejecución: útiles

Algunos comandos muy útiles que debemos conocer son:
cp, mv, rm, touch, cat, grep, find, tail, head, entre otros muchos.

Una breve descripción de ellos:

https://masqueweb.es/comandos-linux/

Comandos y ejecución: ejecución múltiple

Se pueden ejecutar varias cosas en la misma línea de comando, solo hay que separar por punto y coma:

```
cd /home; ls -l; cat /etc/hosts
```

La siguiente opción efectúa un AND lógico, sólo se ejecuta el segundo si el primero va bien:

```
cd /home/usuario && pwd
```

Y la siguiente una OR lógica, si el primero se ejecuta, el siguiente ya no:

```
touch mi_archivo || ls -l
```

Comandos y ejecución: ejecución múltiple

Se pueden ejecutar cosas en segundo plano, finalizando el comando con el símbolo &:

```
tar czvf copiavar.tar.gz /var &
```

Con fg podemos volver a ponerlo en primer plano. Comandos relacionados: bg, ps, jobs

Nota: En este ejemplo, el comando tar comprime en formato gzip la carpeta /var en un archivo en el directorio actual.

Comandos y ejecución: ejecución múltiple

Al cerrar sesión se mata el proceso, podemos evitarlo con nohup. Si se necesita recuperar se puede utilizar el comando screen.

```
nohup tar czvf copiavar.tar.gz /var &
```

Los mensajes siguen saliendo en primer plano, lo que hace recomendable redirigirlos.

Redirecciones

Angel Luis Calvo

ESCUELA IT

Redirecciones

La salida por defecto en bash es la pantalla, la entrada es el teclado. Esto quiere decir que normalmente, cuando se ejecuta un comando, éste mostrará su salida por pantalla, y esperará recibir órdenes desde el teclado.

Hay dos salidas en bash, una es la normal que llamamos *stdout* (1), otra es la de errores, *stderr* (2). La entrada se llama *stdin* (0).

Se puede cambiar este flujo, a eso lo llamamos **redirección**.

Redirecciones

Ejemplos redireccionando la salida:
ls -l > prueba.txt → la salida del comando aparecerá en el archivo prueba.txt, si no existe lo crea, si ya existe lo sobreescribe.
ls -l 2> /dev/null → la salida de errores se envía al dispositivo nulo.
ls -l >> prueba.txt → la salida del comando aparecerá en el archivo prueba.txt, si no existe lo crea, si ya existe añade la salida al contenido que ya hubiese.
ls -l &> prueba.txt → la salida normal y la de errores aparecen en el archivo prueba.txt

Redirecciones

Ejemplos redireccionando la salida:
Para que quede registrado lo que haya hecho un comando en segundo plano, a la vez de que sus mensajes no molesten en primer plano:

```
tar czvf copiavar.tar.gz /var > logtar 2>logtarerr &
```

Redirecciones

Ejemplos redireccionando la entrada:
sort < prueba.txt → la entrada del comando la coge a través del archivo prueba.txt.
cat << FIN > prueba.txt → la entrada del comando la coge a través del teclado hasta teclear lo que indicamos (FIN en este caso) y lo guarda todo en el archivo prueba.txt. (*HereDoc*)
apt upgrade < tecla → en un archivo con la letra s redireccionamos la entrada para que no pida confirmación al actualizar.

Redirecciones: tuberías

Se utilizan para pasar la ejecución de un comando a otro.

Ejemplos:

```
ls -l /etc | grep sudo
-r--r-----  1 root root    755 oct 15  2019 sudoers
drwxr-xr-x  2 root root     4096 abr 29 14:36 sudoers.d

ps aux | grep bash
```

Fundamentos de scripts

Angel Luis Calvo

ESCUELA IT

Fundamentos de scripts

Los scripts son archivos de texto, ejecutables, con instrucciones del sistema operativo, que se ejecutan secuencialmente al ejecutar el script.

¿Cuándo se usarán?
En principio es un procedimiento para hacer cosas sencillas y automatizar tareas cotidianas, para evitar el uso de lenguajes de programación, sin necesidad de compilar nada ni de usar programas añadidos. Como mucho puede ser bueno un editor de texto avanzado. Pueden complicarse todo lo que se quiera.

Fundamentos de scripts: convenciones

En Linux, en principio los archivos pueden llamarse de cualquier manera, sin importar su extensión, el sistema los ejecuta en función de su contenido.
Por convención deberíamos poner siempre extensión .sh, para poder identificarlos fácilmente.
A la hora de ponerles nombre es importante no utilizar nombres ya utilizados, por ejemplo nombres de comandos en uso.

Fundamentos de scripts: sintaxis

La sintaxis a utilizar será la del shell correspondiente, esto es, bash.
Todos los scripts deben tener una primera línea (llamada *shebang*, o *shabang* según diversas fuentes) que indique el shell que se quiere utilizar: #!/bin/bash

Los comentarios dentro del archivo de script se indican con un #
Esos comentarios jamás producen salida alguna, solo ayudan a aclarar el contenido del archivo, o a evitar que se ejecute algo mientras se está desarrollando el script.

Fundamentos de scripts: sintaxis

Caracteres especiales:
La contrabarra \ sirve como carácter de escape, para tomar como literal algo que podría tomarse como una variable, por ejemplo. Al final de una línea indica que la línea continua en la siguiente.

Se usan comillas dobles para definir textos, y se expanden al traducir variables. Eliminan el significado especial de metacaracteres excepto $ y \
Las comillas simples definen textos que no se expanden, no traducen variables. Evitan tener que usar \ continuamente.

Fundamentos de scripts: sintaxis

Metacaracteres.
Una lista de esos caracteres con significados especiales, algunos ya comentados anteriormente:

* ? [ ] ' " \ $; & ( ) | ^ < >

Hay que procurar no utilizarlos en nombres de archivo, variables, funciones etc.

Fundamentos de scripts: ejecución

Debe recordarse que para ejecutar un script debemos proporcionarle permisos de ejecución.
Para ejecutarlo podemos escribir su nombre si está alojado en algún directorio del *path*. Si estamos en su directorio podemos escribir ./ antes del nombre para ejecutarlo.

Si además, lo que se quiere es que las variables definidas en el script, se exporten, también podemos ejecutarlo escribiendo:

. prueba.sh ó bien source prueba.sh

Fundamentos de scripts: ejecución

Primer ejemplo de script
Haremos uno que muestre un mensaje en pantalla.
Usaremos el editor nano, por ejemplo, y tecleamos:
nano primer.sh → y escribimos dentro el código

```
#!/bin/bash
#Nuestro primer script, esto es un comentario.
echo hola
```

Guardamos el archivo, le damos permisos de ejecución. Para ejecutarlo podemos hacer ./primer.sh

Fundamentos de scripts: ejecución

Podemos probar a copiar nuestro script en /bin y ejecutar su nombre sin más.

Para ejecutar el script también podemos invocar al shell:

```
bash primer.sh
```

En este caso, el shell indicado se impone a lo que diga el *shebang*.

Ejercicio

Probemos lo siguiente: hacemos un script, llamado mi_shell.sh que contenga

```
#!/bin/dash
readlink /proc/$$/exe
```

readlink es un comando que nos indica el enlace al que apunta un determinado proceso. Los procesos están en /proc, se identifican con un número, el PID, que se almacena en la variable de sistema $.
Si ejecutamos ./mi_shell.sh nos dira un resultado, pero si hacemos bash mi_shell.sh nos mostrará otro.

Se pueden sacar diversas conclusiones.

ESCUELA IT

Buenas prácticas

Se pueden definir unas reglas de estilo, como las comentadas para los nombres de los scripts:
Nombres de variables de entorno en mayúsculas.
Nombres de variables locales en minúsculas.
Guiones bajos para separar porciones de nombres de las mismas.
Poner comentarios indicando para qué se usan las diferentes secciones de código.
Prevenir el comportamiento de variables, por ejemplo encerrándolas entre comillas.

Buenas prácticas

Hay quien recomienda terminar cada ejecución en ;
Hay una serie de instrucciones específicas que pueden ser buenas al principio de todos los scripts:

```
set -o errexit
set -o pipefail
set -o nounset
```

Con estas opciones el script se detiene si hay un error en ejecución o en una tubería, o si hay variables no declaradas.

Variables

Además de las variables de entorno que proporciona el sistema, podemos crear nuestras propias variables globales dentro de cada script. Si queremos exportarlas para usarlas en otros sitios usamos export:

```
export mi_var=valor
```

Se trata de variables de texto (si se incluyen espacios, los valores deben ir entre comillas)
Además se pueden generar variables locales para las funciones dentro de un script, con el comando *local*.

Variables

Para asignar resultados de comandos a variables, la sintaxis es:

```
fecha=$(date) #también podría utilizarse set
echo Hoy es $fecha #en caso de ambigüedad hay que usar llaves {}
```

Se pueden utilizar las variables de entorno con normalidad.

Ejemplo:

```
mi_var=”una prueba”
echo Soy el usuario $USERNAME ejecutando $0 en el equipo $(hostname)
echo Y esto es $mi_var
```

Ojo a los acentos invertidos, se desaconsejan.

Ejercicio

Hacer un script copia.sh, que haga una copia de un archivo llamado uno.txt (creado para la ocasión) en la carpeta home del usuario que ejecuta el script, indicando en el nombre de archivo de destino la fecha de ese día.

```
#!/bin/bash
#Creamos la variable
fecha=$(date +%d_%m_%Y)

#Hacemos la copia
cp uno.txt “/home/$USERNAME/uno$fecha.txt”
#Hemos usado la variable de sistema USERNAME para que se guarde
#en el perfil del usuario en curso.
echo La variable \$fecha almacena la fecha del sistema #ejemplo de uso de un carácter de escape
```

ESCUELA IT

Variables

Para definir variables numéricas se usa el comando *let*.
Admite operaciones aritméticas.
Ejemplo:

```
let a=10
let b=20
let c=$a+$b
echo la suma de a+b es $c
```

Otra sintaxis es ((expresión)), por ejemplo: `suma=$((10+20))`

Variables

Para incrementar o decrementar valores de las variables se puede emplear la sintaxis:

```
let a++ #incrementa en 1 la variable
let a=a+3 #incrementa 3 en este caso
let a-- #decrementa en 1
```

Para dividir: let c=$a/$b para el cociente, let d=$a%$b para el resto. bash solo admite enteros, para operar con decimales se puede usar el comando auxiliar bc.

Ejercicio

Crear un script que divida el valor de dos variables a y b, de valores 25 y 33 respectivamente, que muestre por pantalla su resultado.

```
#!/bin/bash
let a=25
let b=33
division=$(echo "scale=2; $a/$b" | bc)
echo El resultado es $division
```

ESCUELA IT

Variables

Un comando específico para variables: read
Lee lo que escriba el usuario en línea de comando, almacenándolo en una o más variables.
Se puede utilizar un comando *echo* para solicitar al usuario su colaboración, o ejecutar *read* con el parámetro *-p* para introducir el mensaje.
Tiene un parámetro *-s* para ocultar lo que escribe el usuario.
Por ejemplo:

```
read -sp “indique su contraseña” clave
echo la contraseña es $clave
```

Variables

Otra orden, cut, es idónea para escoger lo necesario de una cadena de caracteres. Corta las columnas o campos seleccionados de uno o más ficheros. La opción *-d* indica lo que hará de delimitador de columnas, la opción *-f* las columnas que queremos escoger. Un comando interesante para usar con *cut* es tr -s " ".

Ejemplos:

cut_prueba.sh

```
fecha=$(date | cut -d " " -f 1,2,3)
hora=$(date | cut -d " " -f 5)
echo "Hoy es $fecha y son las $hora"
```

cut_mal.sh

```
gigas=$(df -h | grep sda5 | cut -d " " -f 1,4)
echo "El tamaño de sda5 es $gigas"
echo ¿Seguro que todo está correcto?
```

Variables

Manejo de cadenas de texto:
Con las variables se pueden realizar muchas operaciones de cambio o sustitución de caracteres y otras cosas, que puede ser muy útil.
Ejemplos:
cadena=“esto es un texto”
echo ${#cadena} #mostrará el número de caracteres de la variable
Sustituir texto → echo ${cadena//texto/mundo} #cambia la palabra texto por mundo en la variable cadena.
Más ejemplos:
https://linuxcenter.es/component/k2/item/144-consejos-para-manejar-texto-directamente-en-bash

Variables

Manejo de cadenas de texto:
Con los corchetes se puede filtrar por caracteres variables, por ejemplo
ls [ab]* → listará todos los archivos que empiezan por a o por b.
Vale cualquier tipo de caracteres
ls [ABC][123]* → muestra los que empiezan por A, B ó C y el segundo carácter es 1, 2 ó 3

Más ejemplos:
https://linuxcenter.es/component/k2/item/144-consejos-para-manejar-texto-directamente-en-bash

Chequeo

Una cosa importante es la comprobación de que tenemos bien construído el script.
La ejecución bash -x mi_script.sh permite ver la ejecución línea a línea (igual que si ponemos set -x en el script) y si añadimos -v, con detalles.
Es una buena opción redirigir los comandos más sensibles a un *log*, para comprobar sus salidas.

```
cp mi_archivo.txt destino.txt &>> registro_de_copias.log
```

Chequeo

Una herramienta interesante es shellcheck
Analiza el script y realiza una crítica constructiva con recomendaciones de mejora, y sobre todo revisa la sintaxis.
Hay que instalarla: apt install shellcheck

También puede ser conveniente contar con una chuleta de comandos de bash, para ayudar a la programación y a la comprobación.

Hay muchas, aquí se puede descargar una:
https://www.lawebdelprogramador.com/pdf/7754-chuleta-shell-script.html

Ejercicios

1.- Modificar el path para no tener que poner ./ al ejecutar scripts.

2.- Script para actualizar el sistema sin que pida confirmación, registrando el proceso.

3.- Crear un script que copie los archivos de configuración de ssh (/etc/ssh) en la carpeta de descargas del usuario en curso. Que quede registrado el evento en el archivo copias.log, en el home del usuario, donde se incluirá a continuación la fecha en que se realizó la copia.

4.- Realizar un script que pida dos números y entregue a la salida la suma, resta, multiplicación y división de ambos.

5.- Crear un script que comprima los archivos de la carpeta del curso, y los copie en la carpeta /mnt/comprimidos/ previamente creada con los permisos adecuados, en un archivo con el nombre del mes y año en curso.

BASH
THE BOURNE-AGAIN SHELL

ESCUELA IT

Ejercicios

6.- Elaborar un script que genere un archivo (con nombre resultado.txt) con las líneas del archivo /var/log/auth.log correspondientes al mes de mayo con referencias del comando sudo. Guardar dicho archivo en el directorio home del usuario, con permisos totales para el propietario, solo lectura para el grupo, y ninguno para el resto.Al finalizar debe mostrar el resultado.

ESCUELA IT

1.- PATH=$PATH:./

2.-

```
#!/bin/bash
set -o errexit
set -o pipefail
set -o nounset
clear
echo Vamos a actualizar
sleep 2 #Una prueba de retardo
echo Actualizando
#Los paréntesis son para que el log recoja la salida de las dos órdenes.
(apt update && apt upgrade -y) &>> actualiza.log
echo Terminado
```

(Hay que ejecutarlo con sudo)

ESCUELA IT

3.-

```
#!/bin/bash
set -o errexit
set -o pipefail
set -o nounset
echo Iniciando copia
cp -r /etc/ssh/ ~/Descargas/ >> ~/copias.log
#Alternativa: /home/$USERNAME/Descargas
date >> ~/copias.log
echo Finalizado
```

ESCUELA IT

4.-

```
#!/bin/bash
read -p "Escriba dos números seguidos, por favor " num1 num2
echo Los números escritos son $num1 y $num2
echo La suma de ambos es:
suma=$(echo "scale=2; $num1+$num2" | bc)
echo $suma
echo La resta:
resta=$(echo "scale=2; $num1-$num2" | bc)
echo $resta
echo El producto:
producto=$(echo "scale=2; $num1*$num2" | bc)
echo $producto
echo La división es:
(echo "scale=2; $num1/$num2" | bc)
echo $division
```

ESCUELA IT

5.-
Creamos la carpeta y asignamos permisos:

```
sudo mkdir /mnt/comprimidos
chmod 777 /mnt/comprimidos
```

```
#!/bin/bash
set -o errexit
set -o pipefail
set -o nounset
fecha=$(date +%m_%Y)
tar cvzf "/mnt/comprimidos/$fecha.gz" ~/curso
echo Finalizado
```

ESCUELA IT

6.-

```
#!/bin/bash
echo Comenzamos
cat /var/log/auth.log | grep May | grep sudo > /home/$USERNAME/resultado.txt
chmod 740 /home/$USERNAME/resultado.txt
echo Finalizada la creación
cat /home/$USERNAME/resultado.txt
echo FIN
```

ESCUELA IT

BASH

THE BOURNE-AGAIN SHELL

GRACIAS!!

Angel Luis Calvo

ESCUELA IT