INF-113 Programación Web I

Programación de Shell Scripts (I)

Basado en diapositivas de Antonio Durán

La Shell

- La shell, o intérprete de comandos, es el programa que interpreta las peticiones del usuario para ejecutar programas.
  - En MS-DOS: COMMAND.COM
  - En GNU/Linux y otros sistemas Unix hay muchas: sh, bash, csh...
    - Bash (Bourne-Again SHell) se ha convertido en el estándar de facto en GNU/Linux.

La Shell

Cuando la shell está lista para recibir órdenes, muestra el *prompt*:

$: para usuarios normales

: cuando entramos con el usuario root

Uso de comandos

$ comando [argumentos]

$ ls

$ls -l

La Shell

- Argumentos con formato largo y corto
  - Formato corto
    - $ ls -r
  - Formato largo
    - $ ls –reverse
- Combinación de argumentos cortos
  - Órdenes equivalentes:
    - $ ls -a -l
    - $ ls -al

La Shell

- Obtener ayuda rápida sobre los argumentos de un comando
  - $ comando –help
    - $ ls –help
- Ayuda en línea: man
  - Páginas de manual para todos los comandos y muchos archivos de configuración
    - $ man ls

Comandos básicos

cd: cambia de directorio

ls: lista los contenidos del directorio

mkdir: crea un directorio

rm: borra un archivo

cp: copia un archivo

mv: mueve o renombra un archivo

pwd: muestra el directorio actual

Comandos básicos

cat: muestra un archivo

tail: muestra el final de un archivo

head: muestra el comienzo de un archivo

more: muestra un archivo, paginando la salida

less: como more, pero navegable hacia detrás

Comandos básicos

grep: filtra las líneas que contienen un patrón

sort: ordena un archivo

cut: corta secciones de las líneas de un archivo

wc: cuenta las líneas, palabras y caracteres de un archivo

uptime: muestra el tiempo de funcionamiento del sistema

Comandos básicos

echo: muestra una línea de texto

file: determina el tipo de archivo

find: busca ficheros

who: muestra los usuarios conectados

w: muestra lo que hacen los usuarios conectados

chown: cambia el propietario de un archivo

Comandos básicos

chgrp: cambia el grupo de un archivo

chmod: cambia los permisos de un archivo

date: muestra la hora y la fecha del sistema

tar: compresión de archivos

Expansión de nombres de ficheros y directorios

Para referirnos a varios archivos, se usan los comodines

?: un carácter

*: cero o más caracteres

[conjunto]: uno de los caracteres de conjunto

[!conjunto]: un carácter que no esté en conjunto

Expansión de nombres de ficheros y directorios

$ ls *.txt

Muestra todos los archivos terminados en .txt.

ls doc?.txt

Muestra doc1.txt, doca.txt...

ls fichero.[ch]

Muestra fichero.c y fichero.h, pero no fichero.o o fichero.cpp

ls [a-c]* : Ficheros empezados por a,b,c

Práctica 9

Experimentar con los comodines

- Mostrar todos los ficheros del directorio que empiecen por a y terminen por p
- Mostrar todos los ficheros terminados en a o b
- Mostrar todos los ficheros que empiecen por un número

Expansión de nombres de ficheros y directorios

El comodín tilde

- Se usa para referirse al directorio del usuario
- ~ representa el directorio del usuario
- ~/Desktop expande a /home/usuario/Desktop
- ~pepe/ referencia al directorio personal del usuario llamado pepe

Entrecomillado

- A veces queremos usar un carácter especial literalmente, es decir, sin significado especial
  - Para ello usamos el entrecomillado
  - También podemos usar caracteres de escape
    - Se precede el carácter por el símbolo \
      - $ echo ~
      - $ echo \~

Entrecomillado

Ejemplo del uso de comillas

$ cd /etc

Intentamos buscar los ficheros con extensión .conf:

$ find . -name *.conf

La forma correcta de conseguirlo

$ find . -name '*.conf'

O, con carácter de escape:

$ find . -name *.conf

Ejecución secuencial y concurrente de comandos

Ejecución de comandos en segundo plano

$ comando &

$ cp -R /home /backups &

Ejecución secuencial de comandos

$ comando1 ; comando2

$ cd /etc ; ls -al

Combinaciones de teclas

Es muy útil conocer algunas de las combinaciones de teclas de BASH si vamos a trabajar con ella frecuentemente

Permiten un trabajo mucho más rápido

Hacen mucho más amigable el uso de la consola

Combinaciones de teclas

- El historial de comandos
  - Bash mantiene un histórico de comandos, donde se van almacenando los comandos que vamos ejecutando.
    - ~/.bash_history
      - Nota: ficheros que empiezan con punto son ocultos
  - Se puede usar el comando history para ver el historial
  - La forma más usual de usar el history es con los cursores de arriba y abajo

Combinaciones de teclas

Ejecutar comandos anteriores

- !! : ejecutar el último comando
- !n : ejecutar el comando número n
- !cadena : ejecutar el último comando que empiece por cadena

Combinaciones de teclas

- Teclas de control de terminal
  - Las interpreta el terminal, no Bash
  - Control-C: para el comando actual
    - Ejemplo: find / y pararlo antes del fin
  - Control-D: final del flujo de entrada
    - Ejemplo: cat y terminar cuando deseemos
  - Control-U: borra desde la posición actual hasta el principio de la línea
  - Control-W: borra desde la posición actual al principio de la palabra

Combinaciones de teclas

Buscar en el historial

- Control-R busca hacia atrás en el historial, un comando que contenga un determinado texto
- Extremadamente útil una vez que te acostumbras
- Al pulsar las teclas, el prompt cambia de forma, y nos va mostrando el comando del histórico que cumple el patrón dado
- Si pulsamos Enter, se ejecuta el comando
- Si pulsamos ESC, deja editarlo
- Si pulsamos Control-G se limpia la línea

Combinaciones de teclas

- Autocompletar con el tabulador
  - Ayuda a terminar de rellenar un comando con el nombre de un comando, de una variable, de un fichero o un directorio.
  - Para ello se siguen estas reglas cuando se pulsa el tabulador:
    - Si no hay nada que empiece por el texto de la palabra que precede al cursor se produce un pitido que informa del problema

Combinaciones de teclas

Se siguen estas reglas cuando se pulsa el tabulador:

- Si hay un comando (en el PATH), una variable o un nombre de fichero que comienza con el texto escrito, Bash completa la palabra
- Si hay más de una forma de completar la palabra, Bash completa lo más que puede y emite un pitido
- Si al haber más de una posibilidad para completar, pulsamos dos veces el tabulador, se muestran todas las posibles candidatas

Práctica 10

- Practicar el uso del tabulador:
  - Escribir *ca* en la terminal y pulsar TAB para ver todos los comandos que empiezan por *ca*
  - Utilizarlo para completar el nombre de los ficheros:
    - $cat [TAB]
      - Comprobar como muestra las posibilidades
- Practicar el uso de Ctrl-R para repetir comandos

Editores de texto

En consola

- nano
- vim

En modo gráfico, cualquier editor que pueda guardar en modo texto

- gedit
- OpenOffice
- ....

Shell scripts

Secuencia de comandos y estructuras de control que especifican en que modo deben ejecutarse dichos comandos.

El script “Hola Mundo”:

```
#!/bin/bash

echo Hola Mundo
```

El Script holamundo.sh

Prímera línea del script

#/bin/bash

Indica el programa que debe ejecutar el script. Si no se encuentra el programa, se producirá un error.

echo Hola Mundo

Ejecuta el comando echo con los argumentos Hola Mundo, haciendo que se muestren por pantalla.

Ejecución de scripts

Podemos ejecutar el script de dos formas:

Invocando al intérprete para que ejecute el script:

bash holamundo.sh o

source holamundo.sh

Ejecutando directamente el script:

./holamundo.sh

Para ello necesitamos tener permiso de ejecución

chmod +x holamundo.sh

./ indica ejecutar desde el directorio actual

Permisos

Basados en usuarios y grupos

Cada archivo, tiene un propietario y pertenece a un grupo

Se asignan permisos para el propietario, para el grupo y para los demás usuarios

Permisos

Hay 3 tipos de permisos

- r: Lectura
  - Permite leer un archivo
  - En directorio, permite listar sus contenidos
- w: Escritura
  - Permite modificar un archivo
- x: Ejecución
  - Permite ejecutar un archivo
  - En un directorio, permite el acceso a él (cd)

Permisos

Para ver los permisos de un archivo, usamos el comando ls:

$ ls -l

-rwxr-xr-x 1 root root 4036 2007-12-13 10:29 index.php

9 letras indican los permisos:

rwxr-x-r-x

Las tres primeras son los permisos del propietario, las siguientes del grupo y las últimas, del resto de los usuarios

Permisos

rwx

Permisos de lectura, escritura y ejecución para el propietario

r-x

Permisos de lectura y ejecución para el grupo

r-x

Permisos de lectura y ejecución para el resto de los usuarios

Permisos

-rwxr-xr-x

La primera letra indica el tipo de archivo:

- :archivo regular

d: directorio

l: enlace

Permisos

Modificación de permisos

- $ chmod [ugoa]+-[rwx] fichero
  - Añade o quita permisos a los existentes
    - u: usuario
    - g: grupo
    - o: otros
    - a: todos
  - $ chmod u+x script.sh
- $ chmod modo fichero
  - modo son tres números en octal
  - $ chmod 755 script.sh

Práctica 11

Crear un directorio para guardar las prácticas que iremos realizando llamado scripts.

$ mkdir scripts

Abrir un nuevo fichero de texto y copiar el script holamundo.sh.

En consola: nano, vim, ...

En modo gráfico: gedit, gvim, openoffice...

Darle permisos de ejecución y probar el script

Syntax Highlighting

Es muy útil usar editores que coloreen el código, ya que es mucho más fácil de leer

En Vim:

Añadir a ~/.vimrc

```
syntax on
```

En Nano:

Copiar el fichero nanorc-bash a ~/.nanorc

Comentarios

Todo lo que se encuentre en una línea detrás del carácter '#' se considera un comentario y es ignorado por el intérprete.

```
#!/bin/sh

#Esto es un comentario

echo Hola Mundo # imprime hola mundo
```

El Script backup1.sh

#!/bin/bash

Crea backup del directorio personal

tar -zcf /tmp/mi-backup.tgz ~/

Crea un archivo comprimido en formato tgz con los contenidos del directorio del usuario (~/) y lo deja en /tmp

echo “Copia de seguridad realizada”

Se informa al usuario de que se ha realizado la copia

Redirección de E/S

Todo proceso abre 3 ficheros estándar:

- stdin (descriptor de fichero 0)
  - El lugar desde el que el programa espera leer su entrada. Por defecto es el teclado
- stdout (descriptor de fichero 1)
  - El lugar donde el programa escribirá su salida. Por defecto es la pantalla.
- stderr (descriptor de fichero 2)
  - Es el lugar donde el programa escribe sus mensajes de error. Por defecto es la pantalla.

Redirección de E/S

- Estos ficheros pueden redirigirse de varias formas
  - Redireccionamiento de salida estándar de proceso a fichero.
    - La salida que cualquier comando genere en stdout, se puede redireccionar a cualquier otro fichero:
    - ls -al > ls.txt
      - Crea el fichero ls.txt en el directorio actual, con el resultado de la ejecución del comando ls. Se suprime la salida normal por pantalla de ls, por estar redirigida.

Redirección de E/S

- Redireccionamiento de salida estándar de proceso a fichero.
  - ls -al > ls.txt
    - Si el fichero ls.txt existe, se sobrescribe
  - ls -al >> ls.txt
    - Si el fichero no existe, se crea nuevo
    - Si el fichero existe, se añade la salida del comando al final del mismo.

Redirección de E/S

- Redireccionamiento de salida de errores de proceso a fichero.
  - Se redirecciona la salida de errores del comando a otro fichero
  - # grep hola /tmp/* 2>errores.txt
    - En el fichero errores.txt encotramos los mensajes de error generados por grep en su ejecución
- El fichero especial /dev/null actúa de “agujero negro” donde podemos mandar las salidas no deseadas
  - # grep hola /tmp/* 2>/dev/null

grep

- Programa extremadamente útil en la creación de scripts o en el trabajo en consola
- Muestra por pantalla las líneas que contienen un patrón
  - grep bash holamundo.sh
    - Líneas que contienen “bash”
  - grep -v bash holamundo.sh
    - Líneas que NO contienen “bash”

Redirección de E/S

- Redireccionamiento de salida de errores a la salida estándar
  - `# grep hola /tmp/*  2>&1`
    - La salida de errores se envía al mismo fichero que la salida estándar
- Redireccionamiento de salida estándar a la salida de errores
  - `# grep hola /tmp/*  1>&2`
    - La salida estándar se envía al mismo fichero que la salida de errores

Redirección de E/S

- Redireccionamiento de salida de errores y la salida estándar
  - # grep hola /tmp/* &> /dev/null
    - Se redirige toda la salida del programa. Éste ejemplo podría ser útil en scripts pensados para ejecutar con cron, en los que no nos interese su salida.
  - # grep hola /tmp/* >resultado.txt 2>errores.txt
    - Se redirige la salida estándar al fichero de resultados y los errores hacia otro fichero

El Script backup2.sh

#!/bin/bash

tar -zcf /tmp/mi-backup.tgz ~/
  2>/tmp/errores.txt

Crea un archivo comprimido, dejando los errores producidos en el fichero indicado.

echo “Copia de seguridad realizada”

echo “Consulte el fichero para comprobar que se hizo correctamente”

Redirección de E/S

- Redireccionamiento de entrada
  - Cualquier comando que lea su entrada de stdin puede redireccionarla para que venga de otro fichero
  - $ read a < fichero
- Comando cat
  - Copia la entrada estándar en la salida estándar
    - $ cat
    - $ cat < fichero
    - $ cat fichero #aqui cat no usa la entrada estándar

Práctica 12

Escribir un script llamado “fecha.sh” que cree un fichero con la fecha, el listado de todos los usuarios conectados en el sistema y el tiempo que lleva funcionando.

Comandos:

- date
- who
- uptime

Redirección de E/S

Tuberías (pipes)

Las tuberías permiten usar la salida de un programa como la entrada para otro programa

#ls -al | less

Se ejecuta el comando ls y, en lugar de mostrar su salida, se envía como entrada al comando less, que pagina el resultado y permite navegar por él.

Otra forma de obtener el mismo resultado que al ejecutar ls *.txt usando tuberías:

#ls -al |grep “\.txt$”

grep

Para patrones más complejos que una palabra, se usan “”

Para escapar los caracteres especiales, se usa \

\.

$ significa al final de la línea

^ significa al principio de la línea

grep “^#” *

Redirección de E/S

- Tuberías (pipes)
  - Muy útiles para la filosofía de las utilidades Unix: comandos que hacen pocas cosas se unen entre sí para lograr los resultados deseados
  - $ cat alumnos.txt | sort | uniq | lp
    - Se ordena el fichero
    - Se quitan las líneas repetidas
    - Se manda imprimir el resultado

Redirección de E/S

Comandos útiles para desarrollar scripts:

cut

Imprime la parte seleccionada de una línea

Opciones:

-c: Muestra solo los caracteres de esas posiciones

```
$ echo HOLA|cut -c2-3
OL
```

-f: Selecciona solo los campos elegidos

-d: indica el carácter delimitador (TAB por defecto)

```
$ echo “1 2 3”|cut -f1,3 -d“ “
1 3
```

--complement: complementa la selección

Redirección de E/S

Comandos útiles para desarrollar scripts:

tr [opcion] Conjunto_de_caracteres1 [CC2]

Modifica o borra caracteres de la entrada estándar, escribiendo en la salida estándar

Sin opciones, se reemplazan las ocurrencias del CC1 por las del CC2:

```
$ echo HOLA|tr H B
BOLA
$ echo HOLA|tr A-Z a-z
hola
```

Redirección de E/S

Comandos útiles para desarrollar scripts:

tr [opcion] Conjunto_de_caracteres1 [CC2]

Modifica o borra caracteres de la entrada estándar, escribiendo en la salida estándar

Opciones

-d: borra los caracteres de CC1

```
echo HOLA|tr -d H,A
OL
```

-s: reemplaza cada repetición de un carácter por una sola ocurrencia de ese carácter

```
echo XXX|tr -s X
X
```

--complement: complementa la selección

Práctica 13

Modificar el script anterior, llamado “redireccionar.sh”, para refinar la salida

- Que no se muestre la hora ni el uso horario en la salida del comando date
  - cut
- Que se ordene la lista de usuarios conectados y se eliminen los repetidos
  - sort, uniq
- Que las estadísticas de uptime aparezcan en mayúsculas
  - tr

Variables

Se pueden usar variables como en otros lenguajes de programación

No hay tipos de datos

Una variable puede contener un número, un carácter o una cadena de caracteres

No hay necesidad de declararlas, se crean al asignarles un valor

Variables

Para asignar un valor a una variable, se usa su nombre y el símbolo de asignación:

VAR=”Hola Mundo”

No puede haber espacios entre la variable y el valor asignado

Para obtener el valor de la variable, se referencia precedida del símbolo $

echo $VAR

Variables

Tipos de asignación

- VAR=”Hola Mundo” #asignación normal
- let a=16+5 #asignación con let
  - Se evalúa la expresión y se asigna el resultado a la variable
- for i in 1 2 3 #asignación en bucles
  - En cada pasada, se asigna un valor
- echo -n “Dame la IP: “; read IP #con read
  - se obtiene el valor de la entrada estándar

Variables

- Sustitución de variables
  - Los $ se interpretan en el interior de las comillas dobles
    - echo “$VAR”
  - Los $ NO se interpretan en el interior de comillas simples
    - echo '$VAR'

Variables

VAR="A B C"

Comando	Salida	Explicación
echo VAR	VAR	No se referencia la variable
echo $VAR	A B C	echo recibe 3 argumentos
echo ${VAR}	A B C	Igual que el anterior
echo "$VAR"	A B C	echo recibe 1 solo arg
echo '$VAR '	$VAR	No se interpreta la var

Variables

En realidad, la forma de referirnos a las variables que hemos usado es una simplificación

$VAR es la simplificación de ${VAR}

Podemos usar la forma simplificada siempre que no existan ambigüedades

Siempre que la variable vaya seguida de una letra, guión o dígito, es necesario usar ${VAR}

nombre=Antonio

Apellido=Duran

echo “$nombre_$apellido”

Correcto: echo “${nombre}_$apellido”

Variables

La expresión $(comando) obtiene el resultado de la ejecución de un comando para asignarlo a una variable:

```
DIA=$(date +%d)
echo $DIA
```

También se puede hacer:

```
DIA=`date +%d`
echo $DIA
```

El Script backup3.sh

#!/bin/bash

OF=/tmp/mi-backup-$(date +%Y%m%d).tgz

Crea una variable con el nombre del fichero añadiéndole la fecha de hoy

tar -zcf $OF ~/

Crea el fichero cuyo nombre es el contenido de la variable

Práctica 14

Escribir un script “usuario.sh” que pida el nombre al usuario, lo guarde en una variable, y posteriormente lo imprima por pantalla.

- Usar read para leer el nombre de teclado

Argumentos

$ ./script a1 a2 a3 a4 a5 ...

Para acceder a los argumentos de un script se usan variables:

- $1, $2, $3 ...: Parámetros
- $0: nombre del script
- $*, $@: Todos los parámetros separados por espacios
- $#: Número de argumentos pasados

Argumentos

$ ./script a1 a2 a3 a4 a5 ...

Con el comando shift se reasignan los argumentos perdiendo todos una posición:

$1 <-- $2, $2 <-- $3 ...

Para acceder al parámetro 10,11.. debemos usar ${10}, ${11}..

Argumentos

Script para crear un usuario, recibiendo como parámetros el nombre, el grupo y la contraseña.

E_PASS=`mkpasswd -S “XX” $3`

Ejecuta el comando mkpasswd para obtener la clave cifrada, y la deja en la variable E_PASS

useradd $1 -g $2 -p $E_PASS

Añade un usuario al sistema, llamado $1, con grupo $2 y la contrasela previamente encriptada

Práctica 15

Escribir un script “infofile.sh” que reciba tres nombres de fichero por parámetro

- Para el primero, se mostrará el tipo de archivo (file)
- Para el segundo, se mostrará toda su información (stat)
- Para el tercero, se muestran las líneas que tiene (wc)

exit y exit status

exit: comando para terminar un script correctamente. Puede aceptar un parámetro, que será pasado al script padre.

exit 1

Todo comando ejecutado, devuelve al terminar un *exit status*

0 si terminó correctamente

no-cero en caso contrario

exit y exit status

Se puede consultar el exit status de un comando

$ ls

$ echo $?

0 # ls se ejecuta correctamente

$ xx

$echo $?

127 # Se produce un error al no existir

$ grep xxx fichero.txt

$ echo $?

1 # error por no encontrar la cadena