INF-113 Programación Web I

Programación de Shell Scripts (III)

Basado en diapositivas de Antonio Durán

Funciones

Bash tiene funciones como otros lenguajes, aunque algo más limitadas

```
function nombre_funcion {
comando
...
}

o

function nombre_funcion () {
comando
...
}
```

Funciones

Ejemplo de script con una función simple

```
#!/bin/sh

function prueba ()
{
    echo “Esto es una función”
}

prueba
```

Funciones

- La declaración de una función debe preceder a su llamada
  - No hay método para “declarar” una función como en C
- Las funciones pueden recibir argumentos
  - Accedidos en $1, $2 ...
- Pueden retornar un código de salida

Funciones

Ejemplo de script con una función con argumentos

```
#!/bin/sh

function prueba ()
{
    echo “El argumento es $1”
}

prueba hola
```

Funciones

- Argumentos del script / argumentos de la función
  - Necesario pasarlos explicitamente

```
#!/bin/sh

function prueba ()
{
    echo “El argumento es $1”
}

prueba
prueba $1
```

Funciones

exit status

- Las funciones retornan este valor. Puede especificarse el valor con la sentencia return. Si no, será el exit status del último comando ejecutado en la función

return

- Termina una función
- Puede aceptar un entero que indica el exit status de la función

Funciones

Ejemplo de script con una función con argumentos

```
#!/bin/sh

if [ $# -ne 1 ]; then
	echo "Uso: $0 fichero"
	exit 1
fi

function legible ()
{
	if [ -r $1 ]; then
		return 1
	else
		return 0
	fi
}

legible $1
if [ $? -eq 1 ]; then
	echo "El archivo es legible"
else
	echo "El archivo no es legible"
fi
```

Funciones

Visibilidad de las variables

- Las variables declaradas como locales sólo son visibles dentro de la función
- Las no declaradas como locales, se suponen globales
- Antes de que la función sea llamada, las variables globales declaradas en ella tampoco son accesibles desde fuera

Funciones

Variables locales y globales

```
#!/bin/sh

function prueba ()
{
        local local_var="local"
        global_var="global"
}

echo L:$local_var
echo G:$global_var
prueba
echo L:$local_var
echo G:$global_var
```

Práctica

- Escribir un script que contenga una función para sumar dos enteros
  - Recibir los operandos por línea de comandos

sed

sed es un editor de flujo

Se usa para realizar transformaciones de texto simples en el flujo de entrada

Conveniente para procesar “por lotes” la edición de archivos

sed trabaja realizando las operaciones de edición especificadas por el usuario en los datos de entrada

La entrada no se modifica

sed

La entrada se procesa línea a línea

En cada una, por defecto, se imprime lo que queda en el buffer de edición al terminar la transformación

sed

Ejemplos

- $ sed '1d' /etc/services | less
  - Elimina la primera linea de la entrada
  - El 1 indica la primera línea
  - la d indica borrar (delete)
- $ sed '1,10d' /etc/services | less
  - Elimina las 10 primeras lineas
- $ sed '/^#/d' /etc/services | less
  - Elimina las líneas que comiencen con #
  - Se usan expresiones regulares entre / / para seleccionar las líneas

sed

Ejemplos

$ sed -n '/^#/p' /etc/services | less

- Muestra sólo las líneas que comienzan con #
- -n hace que sed no muestre el contenido del buffer si no se pide explicitamente
- la p hace que se imprima la línea (print)

$ sed 's/una/otra/' archivo.txt

- Sustituye 'una' por 'otra' en la primera aparición de 'una' en cada línea
- la s realiza el cambio (substitution)

sed

Ejemplos

- $ sed 's/una/otra/' archivo.txt
  - Sustituye 'una' por 'otra' en la primera aparición de 'una' en cada línea
  - la s realiza el cambio (substitution)
- $ sed 's/una/otra/g' archivo.txt
  - Reemplaza todas las ocurrencias de 'una' por 'otra'.
  - La g después de la última /, indica que el reemplazo debe ser global

sed

Ejemplos

$ sed '1,10s/una/otra/g' archivo.txt

El comando de sustitución también acepta direcciones de línea

$ sed '/^#/s/Antonio/Antonio Duran/g' archivo.txt

Cambia, en todas las líneas que sean un comentario (^# = empieza por #) Antonio por Antonio Duran

sed

Script para cambiar el autor en una serie de ficheros

```
#!/bin/sh

for file in *
do
    sed 's/Antonio Duran/Antonio Duran
Terres/g' $file
done
```

Práctica

Usando sed, escribir un script que, partiendo de un mensaje predefinido que tenemos en un archivo de texto, lo personalice para cada usuario alumno del sistema

Como no tenemos acceso a los directorios personales de los otros usuarios, dejarlos en /tmp con nombres del tipo: mensaje.nombre_usuario

Mensaje predefinido: Estimado USUARIO, bienvenido al sistema!

find

- find se usa para buscar ficheros que coincidan con un determinado criterio de búsqueda
- Podemos buscar ficheros por nombre, propietario, grupo, tipo, permisos, fecha y otros criterios
- En la búsqueda se recorren todos los subdirectorios

find

Sitaxis

$ find donde-buscar criterio-búsqueda acción-a-realizar

Todos los argumentos son opcionales y tienen valores por defecto:

donde-buscar: directorio actual

criterio-búsqueda: ninguno, mostrar todos

acción-a-realizar: imprimir los nombres (-print)

find

Algunos ejemplos con el mismo resultado:

```
$ find
$ find .
$ find . -print
```

Búsqueda con criterio y con ruta específica:

```
$ find / -name nombre_fichero
$ find / -name nombre 2>/dev/null
```

find

- Podemos usar comodines en los criterios de búsqueda
  - $ find . -name scr*
    - Necesario escapar el *
- Búsqueda usando dos criterios:
  - $ find / -type f -mtime -7
    - Encuentra los ficheros ordinarios (tipo f) que haya sido modificados (mtime) hace 7 o menos días

find

También podemos ejecutar comandos sobre los ficheros encontrados

$ find / -name core -exec /bin/rm {} \;

- Ejecuta el comando pasado
- {} representa al nombre del fichero que estamos tratando en ese momento
- Necesario \; para terminar el comando

xargs

Otra forma más eficiente de hacer lo mismo que antes es usando el comando xargs

- Convierte un flujo de entrada en una lista de argumentos
- Muy útil cuando obtenemos errores de “Demasiados argumentos”

$ find . -name core | xargs rm

- Más eficiente, ya que rm se ejecuta una vez, en lugar de una vez por cada archivo

find

Ejemplo de un script de otro día en una línea:

$ find . -name *.html -exec mv {} {}.old \;

Busca, en el directorio actual, todos los ficheros .html

Los renombra a .html.old

Problema: find procesa los subdirectorios -> podemos usar la opción maxdepth:

$ find . -maxdepth 1 -name *.html -exec mv {} {}.old \;

Práctica

- Escribir un script que reciba un parámetro con el nombre de usuario
- Debe copiar todos los ficheros regulares que pertenecen a ese usuario en todo el sistema de ficheros, a una nueva carpeta en /tmp
  - No hay que mantener la estructura de directorios
- Descartar los errores

Práctica

Escribir un script que añada una línea así:

Autor: Nombre del Alumno

en la segunda línea de cada script creado en el curso.

Usar head y tail

Dejar intactos los originales

Crear los nuevos en otra carpeta

Práctica

Escribir un script que reciba un nombre de comando por parámetro y mate todos los procesos de dicho comando

Usar:

ps -e (lista procesos)

tr -s (comprime espacios)

grep (elige líneas)

cut (obtiene trozos de línea)

xargs (convierte un flujo en argumentos)

kill (mata procesos)

awk

Sirve para procesar líneas de texto

Cuenta con un sencillo lenguaje de programación

Hay dos formas de usarlo:

```
$ awk -f fuente.awk fichero_entrada
$ awk 'código fuente' fichero_entrada
```

awk

Estructura del lenguaje awk:

```
BEGIN { accion }
/patron/ { accion }
END { accion }
```

awk

Forma de funcionamiento:

- Al comenzar la ejecución, se evaluará la acción entre llaves que hay después de BEGIN
- Para cada campo de texto (por defecto líneas) se evalúa si se ajusta al patrón, y de ser así, ejecuta la acción correspondiente
- Finalmente, se procesa la acción de END

awk

El patrón es una expresión regular. Algunos ejemplos de las que podemos usar:

- f[1-3]x: casará con cualquier línea que comience por f, siga con 1, 2 o 3 y termine en x
- (doc)+([1-3])*: casará con cualquier línea que comience por doc repetido una o más veces, seguido, cero o más veces, de un dígito entre 1 y 3

awk

En cuanto a las acciones:

- El acceso a la línea actual se hace con variables especiales:
  - $0 representa la línea completa
  - $1, $2... representa los campos
- Hay bucles, expresiones condicionales, etc
- Para imprimir resultados, tenemos print y printf, y otras variables especiales como NR (número de línea) o NF (número de campos)

awk

Comprobar si el argumento es númerico

VAR=`echo $1 | awk '/^([0-9])+$/'`

if [-z VAR] -> No es numerico

Usando awk en lugar de cut:

$ awk -F":" '{ print $1 " " $3 }' /etc/passwd

Imprime los campos 1 y 3 de cada línea, usando ':' como separadaor

awk

Podemos transformar la salida a nuestro gusto:

```
$ awk -F":" '{ print "Usuario: " $1 "\tUid: " $3 }' /etc/passwd
Usuario: root   Uid: 0
Usuario: daemon Uid: 1
Usuario: bin    Uid: 2
Usuario: sys    Uid: 3
...
```

awk

Usando un archivo para guardar el patrón

```
$ awk -f pares.awk /etc/passwd
```

Muestra las líneas pares del fichero

Contenido de pares.awk:

```
{
        if ( NR % 2 == 0)
        {
                print $0
        }
}
```

Práctica

Escribir un guión awk que muestre cuantas líneas de un fichero contienen la palabra “for”

- Crear el script awk como un fichero separado
- Usar acciones de BEGIN (para inicializar) y END (para mostrar)
- Usar patrón (for) y acción (sumar ocurrencias)

awk

Usar variables de shell en los scripts awk. Dos formas:

- Usar las variables en el código awk, para que sean sustituidas. El script awk debe ir inline:

```
awk “/$VAR/” '{ print }' fichero
```

- Ejecutar awk con -v para asignar las variables:

```
awk -v VAR=1 -f file.awk fichero
```

awk

Ejemplo para mostrar sólo las líneas elegidas de un fichero. lneas.sh:

```
#!/bin/sh


if [ $# -ne 3 ]; then
        echo "Uso: $0 linea_inicial linea_final
fichero"
        exit 1
fi

awk -v min=$1 -v max=$2 -f lineas.awk $3
```

awk

Ejemplo para mostrar sólo las líneas elegidas de un fichero. lineas.awk:

```
BEGIN {
n = 0
}
{
        n++;
        if ((n >= min) && (n <= max))
                print
}
```

Práctica

Escribir un script, usando en él awk para procesar /etc/passwd, para obtener el número UID del usuario recibido como parámetro.

- No usar fichero separado para el script
  - Necesario acceso a la variable en el patrón
- Usar -F para establecer separador
- Formato de salida:
  - Usuario: user Uid: 10001

Bibliografía

Advanced Bash-Scripting Guide

http://tldp.org/LDP/abs/abs-guide.pdf

El shell Bash

http://dymas.ii.uam.es/~flh/macprog/bash.pdf

Bash Programming Introduction How-To

http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html

Sed mediante ejemplos

http://www.gentoo.org/doc/es/articles/l-sed1.xml

Bibliografía

Awk by example

http://www.ibm.com/developerworks/library/l-awk1.html