Talk Show

Práctica 3

Introducción a los Sistemas Operativos

Facultad de Informática
Universidad Nacional de La Plata

2017

¿Qué es un script?

- ¿Para qué lo utilizarías?
- ¿Qué cosas podes hacer mediante un script?
- ¿Qué necesitás para ejecutarlo? ¿Cómo se ejecuta?

Conceptos principales de shell scripting

- Shebang
- Sentencias
- Tipos de variables
- Asignaciones e interpolaciones
- Funciones
- Listas de elementos
- Entrada/Salida
- Sustitución de comandos
- Redirecciones
- Pipes
- Variables especiales

- Línea con sentido especial dentro de un script: da un hint del comando a utilizar para ejecutar el archivo que contiene el script.
- Es opcional. De estar presente, debe ser la primer línea del archivo, y comenzar con #!.
- Todo script retorna un exit status entre 0 y 255 (comando exit).
- Ejemplos:

```
#!/bin/bash
#!/usr/bin/env ruby
```

- Las sentencias son secuencias de comandos, internos o externos a la shell, que se ejecutarán cuando el intérprete (Bash, en nuestro caso) las lea, según el flujo de ejecución del script.
- Todo comando que ingresamos en una terminal es una sentencia, y toda sentencia puede ser un comando que ingresemos manualmente en una terminal.
- Ejemplos:

```
ls
echo "Hola"!
mkdir un_directorio
cat /etc/passwd | cut -d: -f2
```

- Las variables en Bash pueden tener dos tipos: general (simplifica pensarlos como un String) o arreglo.
- Las variables de tipo general se asignan y acceden directamente, mientras que las de tipo arreglo deben declararse con paréntesis alrededor de sus elementos iniciales y luego se acceden mediante el uso de índices (para indicar la posición deseada).
- Ejemplos:

```
var=general
var2="otra de tipo general"
vacio=()
arreglo=(1 dos 3)
echo "var=$var y var2=$var2"
echo "${arreglo[1]} ${#arreglo[*]}"
for i in ${arreglo[*]}; do echo $i; done
```

- Las asignaciones se realizan a través del caracter '='.
- Dicho caracter debe estar pegado a ambos lados de la asignación.
- La interpolación permite sustituir el contenido de una variable dentro de un string literal siempre y cuando utilicemos comillas dobles.
- Ejemplos:

```
x=100
y='100$x'
echo $y # ¿resultado?
y="100$x" # ¿resultado?
echo $y # ¿resultado?
```

- Declaración: f() ... o function f() ... .
- Permiten modularizar nuestro código, deben estar definidas antes de usarse. Deben retornar un valor entre 0 y 255 a través de la sentencia "return". Si esta no existe retorna el exit status del último comando ejecutado. Una vez definidas, se las puede invocar como si fuera un comando interno más a nuestra shell.
- Ejemplos:

```
myFn() { echo Hello world! }
myFn
myFn() { echo $1 }
myFn 'Hello world!' # ¿qué pasa si lo ejecuto: myFn
     Hello world!?
```

- En Bash, las listas de elementos son muy importantes. Nos permiten iterar sobre elementos, definen qué y cuántos argumentos recibe un script/binario, delimitan un comando de sus argumentos, etc.
- Por defecto, los separadores de elementos en una lista son los caracteres blancos (espacio, salto de línea, tabulación). Esto está definido por una variable interna del intérprete ($IFS en Bash)
- Ejemplos:

```
arr=(1 2 3 4 5)
arr=(esto es un string) # ¿cuántos elementos tiene?
arr=("esto es un string") # ¿y ahora?
```

- Mediante un script podemos manipular archivos.
- Todo proceso en un sistema operativo *nix tiene, por defecto, tres archivos abiertos:
  - **Entrada estándar (stdin)**: identificado con la file descriptor **0**.
  - **Salida estándar (stdout)**: identificado con la file descriptor **1**.
  - **Error estándar (stderr)**: identificado con la file descriptor **2**.
- Nos permiten leer y escribir información.

- Muchas veces es necesario obtener el resultado de la evaluación de una sentencia en particular dentro de otra sentencia. Sería como una interpolación de la salida del subcomando en el lugar donde se realiza la sustitución.
- Este comportamiento se da debido a que la salida estándar (stout) del subcomando se reemplaza y se pega (sustituye) en el lugar donde se realiza la sustitución.
- Se puede utilizar $() o comillas francesas.
- Ejemplos:

```
echo "Mi nombre de usuario es $(whoami)"
```

- Las redirecciones permiten ”conectar” la entrada estándar (stdin) o las salidas (stdout y/o stderr) de un comando a un archivo.
- Útiles para guardar el resultado (la salida) de un comando o sus errores en un archivo, o para suministrar el contenido de un archivo como entrada de un comando.
- Pueden ser destructivas o no.
- Ejemplos:

```
echo Esto va a > un_archivo
./mi_script < datos
mysqldump db > dump.sql
mysql db < dump.sql
```

- Así como las redirecciones comunican comandos con archivos, los Pipes (denotados con — en Bash) son un mecanismo de comunicación entre procesos.
- Conectan la salida estándar (stdout) de un comando con la entrada estándar (stdin) del siguiente, formando una comunicación lineal.
- Pueden unirse tantos pipes como se desee, logrando que los comandos involucrados procesen secuencialmente la salida que reciben del comando anterior, y se la pasen al siguiente de la fila.
- Ejemplos:

```
ps -fu $USER | tr -s "  " | cut -'d ' -f3 | xargs kill
    -9
```

- Existen algunas variables que tienen un significado especial, y cuyo valor es dado por Bash.
- Ejemplos:

```
$*
$#
$0
$1 $2 $3 .. $9 ${10} ${11} ...
$?
$$
$_
$USER
$HOME
$PATH
```

- Ejercicio práctico 17
- Ejercicio práctico 22
- Repositorio de ejemplos de la cátedra: https://github.com/unlp-so/shell-scripts