BASH Scripting

diego.rodriguez@usc.es

Contenidos

- Comandos
- Varibles
- Control de flujo
- Sistema de colas
- Bibliografía

BASH: Bourne-Again SHell

- **Intérprete de comandos** más habitual en los sistemas operativos GNU/Linux actuales.
- Capacidad para interpretar *scripts* que contengan comandos.
- Lenguaje de programación: **BASH scripting**
  - Variables
  - Condicionales
  - Bucles
  - Funciones

Hola mundo!

```
#!/bin/bash
# Script "Hola mundo!"

echo "Hola mundo!" # Listo!
```

- `#!/bin/bash`: intérprete (primera línea!)
- Comentarios: a partir de `#` hasta el final de línea
- `echo`: comando que *repite*
- Ejecución:

```
chmod u+x HolaMundo.sh    # Permiso de ejecución
./HolaMundo.sh            # Ejecución del script
```

Gestionando estructura de ficheros

- `cd` cambio de carpeta (directorio)

```
cd "/home/usuario/ruta absoluta/ejemplo"
cd ../../ruta/./relativa/                # <=> /home/usuario/ruta/relativa/
```

- `ls` muestra el contenido de una carpeta
  - `*` cero o más caracteres.
  - `?` un caracter.

```
ls -1 *.txt ej?.log
```

- `mkdir` crea un directorio
- `rmdir` borra un directorio (vacio)
- `rm` borrado

```
rm fichero.ext ej??.log *.txt
rm -rf directorio              # Mucho CUIDADO con "-rf"
```

Gestionando contenido

- `cat` concatena ficheros e imprime por pantalla
- `>` redirecciona *stdout* a un fichero (sobreescribe)
- `>>` redirecciona *stdout* a un fichero (añade)
- `&>` redirecciona *stdout* y *stderr* a un fichero
- `2>&1` redirecciona *stderr* a *stdout*

```
echo "ola" > ola.txt
echo "y adiós" >> ola.txt
echo "y adiós" >> ola.txt
rm fichero_que_no_existe &> /tmp/errors
rm fichero_que_tampoco_existe >> /tmp/errors 2>&1
```

Otros comandos

- `pwd` print working directory
- `date` imprime la hora/fecha (formato personalizable)

  ```
  date +%Y-%m-%d_%H-%M
  ```

- `man` muestra la *ayuda* de un comando

  ```
  man date
  ```

- `|` *pipe*

  ```
  ls *.sh | cat
  ```

- `;` separarador de comandos

  ```
  ls *.sh > /tmp/kk ; cat /tmp/kk ; rm /tmp/kk
  ```

Búsquedas en texto

- `grep`

  Herramienta de búsqueda en ficheros.

  ```
  grep "texto a buscar" file.txt
  ```

  - Opciones:

    ```
    echo -e "warning\nWarning\nwarnings\nWARNINGS\nOK" > file.txt
    grep -i "warning" file.txt # case-Insensitive
    grep -c "warning" file.txt # Count
    grep -v "warning" file.txt # inVert match
    grep -w "warning" file.txt # whole Word
    grep -n "warning" file.txt # line Number
    grep -r "warning" .        # Recursive (directories)
    grep -lr "warning" .       # List files + Recursive
    ```

  - Combinados:

    ```
    grep -i "warning" file.txt | grep -c "W"
    grep -v "W" file.txt | grep -cw "warning"
    ```

Procesado de texto

- `sed`

  Editor no interactivo.

```
# Sustituciones:
echo uno dos tres | sed s/dos/two/
pwd | sed s,/home/diego,/tmp,

# Borrado:
echo -e "uno\n\ndos\n \ntres" | sed /^$/d
echo -e "uno\n\ndos\n \ntres" | sed /^\ *$/d
```

- `awk`

  Lenguage de procesado de textos.

```
echo uno dos tres | awk '{print $1"-"$3":"$2}'
echo "uno:dos tres:cuatro" | awk -F ":" '{print $2}' | awk '{print $2}'
```

Variables

```
var=123

echo var $var   # var 123
echo ${var}     # 123

new=${var}_ola
echo $new       # 123_ola

ola="ola caracola!" # Si hay espacios se necesitan comillas
```

- Comillas

  ```
  ok="$var"  # Entrecomillado débil
  echo $ok   # 123
  ko='$var'  # Entrecomillado fuerte
  echo $ko   # $var
  ```

- Resultados:

  ```
  search=`cat *.log | grep -c -i error`
  lista_dat=$(ls *.dat)
  ```

Variables internas

- `$HOME` *home* de usuario, /home/nombre.usuario, `~`
- `$PWD` *print working directory*, directorio actual
- `$PATH` Lista **priorizada** para buscar los programas
  - `$LD_LIBRARY_PATH` Lista **priorizada** para buscar las librerías dinámicas
- `$0`, `$1`, `$2`, … argumentos 0 (nombre), 1, 2,… del script
- `$#` número de argumentos del script

```
#!/bin/bash

echo "Ejecutando $0 con $# argumentos"
echo $1
echo $2
echo $3
```

Control de flujo

- if-then-else-fi
- case-in-esac
- for-in-do-done
- while-do-done
- until-do-done

Control de flujo: if-then-else-fi

- Comparadores: `==`, `!=`, `-eq`, `-lt`, `-z`,…
- Cuidado con la sintaxis: ¡dejar **espacios**!

```
if [ 1 == 2 ]; then echo "TRUE"; else echo "FALSE"; fi
var=a
if [ "a" == "$var" ]; then echo "TRUE"; else echo "FALSE"; fi

#!/bin/bash
if [ "$#" -ne 2 ]; then
  echo "Please, give me two args"
else
  echo $2 $1
fi
```

```
var=ola
if [ -z "$var" ]; then echo "var está vacía"; else echo "var contiene '$var'"; fi
```

Control de flujo: for-in-do-done

```
for i in 1 dos 3 cuatro,cinco
do
  echo $i
done
```

```
for name in A B C; do touch new_${name}file.txt; done
```

```
STEP=4; for i in $(seq 1 $STEP 10); do echo $i; done
```

```
for matrix_size in 100 200 500 1000
do
  ./parametric.exe ${matrix_size}
done
```

```
for matsize in 100 200 500 1000
do
  cat conf_PARAM.json | sed s/xxx_MATRIX_SIZE_xxx/${matsize}/g > conf.json
  ./parametric.exe > output_${matsize}.txt
done
```

Argumentos del script: checking

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
  echo "Illegal number of parameters"
  echo "USE:"
  echo "  $0 argument"
  exit
fi
```

Argumentos del script: parsing

```
PROGRAMNAME=$0
while [[ $# -ge 1 ]]
```

```
do
  key="$1"
  case $key in
    -e|--extension)
    EXTENSION="$2"
    shift # past argument
    ;;
    -s|--searchpath)
    SEARCHPATH="$2"
    shift # past argument
    ;;
    -h|--help)
    echo "Usage:"
    echo "  $PROGRAMNAME -e EXT -s PATH [-h]"
    exit
    ;;
    *)
    echo "(Unknown option: $1)"
    ;;
  esac
  shift # past argument or value
done
echo `ls -1 ${SEARCHPATH}/*.${EXTENSION} | wc -l`
```

Cluster de computación

Cluster beowulf, Wikipedia

Sistema de colas

- `qsub`: enviar un trabajo al sistema de colas (se le asigna un ID).

  ```
  qsub -N name script.sh
  ```

  - `-N name`: asinamos el nombre *name* al trabajo dentro del sistema de colas (por defecto, nombre del script).

- `qstat`: consultar el sistema de colas.

  ```
  qstat
  qstat -j ID
  ```

  - `-j ID`: información específica del trabajo ID.

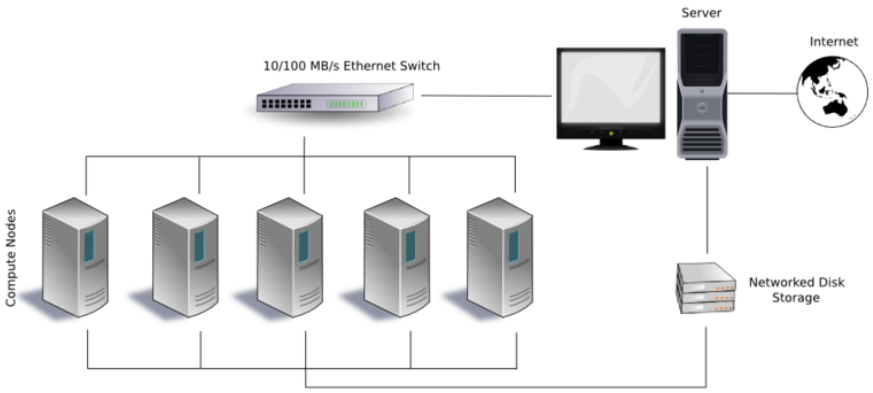

Figure 1: Esquema cluster, Wikipedia

Opciones de `qsub`

- Se le pueden pasar argumentos al script:

  ```
  qsub script.sh con tres argumentos
  ```

- Envia un correo (`-m`)

  ```
  qsub -m ea -M mi.cuenta@cor.reo script.sh
  ```

  - cuando termine el trabajo (end, `e`),
  - o cuando haya algún error (aborted, `a`),
  - al correo indicado despues de `-M`

- `-cwd` (*changue working directory*): la ejecución remota se sitúa en el directorio en el que se ejecuta `qsub`.

  ```
  qsub -cwd script.sh
  ```

  - Por defecto, las ejecuciones remotas se inician en `$HOME`

¿Dónde consultar?

- BASH:
  - Advanced Bash-Scripting Guide, TLDP
  - Bash para principiantes, Wikipedia
  - Gentoo tool reference