Bash Scripting

Devops Básico
Escuevita 2021

Hoja de Ruta

1 Bash Básico

2 Bash Intermedio

3 Bash Scripting

Bash Scripting

Bash = (Bourne Again) Shell

Shell = Intérprete de Comandos de Linux

Scripting = Secuencia de Comandos (mini programa)

-

Bash Scripting = Secuencia de Comandos de Bash

Atajos de teclado (GNOME3)

- Abrir una nueva terminal
  ctrl + t

- Abrir una nueva pestaña de la terminal actual
  ctrl + shift + t

- Cambiar a una pestaña de terminal por su posición
  alt + Número pestaña

- Cortar la ejecución de un comando o programa
  ctrl + c

- Cortar la ejecución de un comando o programa de manera forzosa (también sale de la terminal)
  ctrl + d


Bash Básico

Repaso Comandos Útiles

- Imprimir texto
  echo “texto”
- Crear un archivo nuevo
  touch archivo_nuevo
- Borrar un archivo
  rm archivo
- Copiar un archivo
  cp archivo_origen archivo_destino
- Mover (o renombrar) un archivo
  mv archivo_origen archivo_destino

Repaso Comandos Útiles

- Listar archivos del directorio pasado por parámetro
  ls directorio
- Cambiar de directorio
  cd directorio
- Ver en qué directorio estamos trabajando
  pwd
- Crear un directorio nuevo
  mkdir directorio_nuevo
- Borrar un directorio
  rmdir directorio

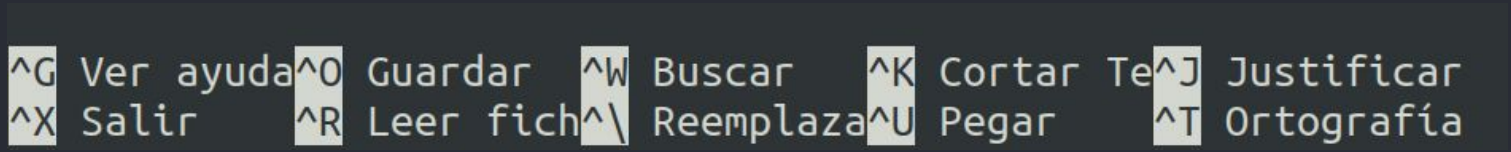
Editores de Texto

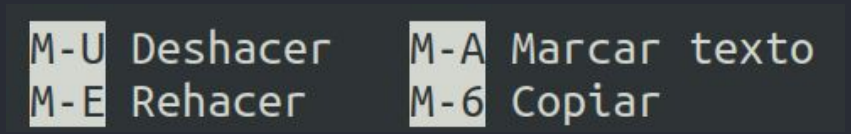
- Editores de texto en Consola:
  - **nano:** sencillo de usar.
  - **VIM:** complejo y potente con múltiples capacidades.
  - **Emacs:** editor de “legado” “más complejo” y “más potente”.

- Recomendación para principiantes:

nano

- Uso de editor de texto **nano:**
  - **nano** archivo
  - Escribir el texto deseado
  - Guardar el texto con ctrl + o (enter)
  - Salir del editor con ctrl + x (enter)
  - En nano las opciones aparece abajo donde:
    - **^** significa **ctrl**
    - **M** significa **shift**





Comandos: visualización archivos

- Imprimir el contenido de un archivo
  cat archivo
- Mostrar las primeras 10 líneas de un archivo
  head archivo
- Mostrar las últimas 10 líneas de un archivo
  tail archivo
- Visualizar un archivo extenso desde el inicio
  more archivo
- Visualizar un archivo extenso desde el inicio (scrolling)
  less archivo_nuevo

Rutas

- Una Ruta o un Path es la forma de identificar de forma unívoca la ubicación de un archivo o directorio específico. Existen dos tipos
  - Path Absoluto: empieza desde el **directorio raíz /**
    /home/user/Documentos/devops/bash_scripting.pdf

  - Path Relativo: empieza desde donde estamos posicionados
    ../Documentos/bash_scripting.pdf

  - El símbolo **“.”** significa directorio actual
  - El símbolo **“..”** significa directorio padre

Comandos: opciones

- La mayoría de los comandos de Bash tiene múltiples “opciones” que permiten brindar más capacidades al comando.

- El formato de los comandos de Bash por lo general respeta el siguiente patrón:

  comando -[opciones] [objetivo1] [objetivo2]

- Ejemplos:
  - Listar los archivos de un directorio con formato “largo”:
    ls -l directorio

  - Copiar todos los archivos y directorio dentro de un directorio de forma **r**ecursiva:
    cp -r directorio_origen directorio_destino

Repaso Comandos Útiles

- Mostrar las primeras **N** líneas de un archivo
  head -N archivo
- Mostrar las últimas **N** líneas de un archivo
  tail -N archivo
- Quedarme con la primer columna de un archivo separado por **,**
  cut -d”,” -f1 archivo
- Contar la cantidad de líneas de un archivo:
  wc -l archivo
- Ordenar un archivo alfabéticamente y omitir duplicados
  sort -u archivo

Comandos: búsqueda

- Buscar una palabra dentro de un archivo
  grep palabra archivo

- Buscar una palabra dentro de todos los archivos de un directorio de forma **r**ecursiva
  grep -r palabra directorio

- Buscar en un directorio recursivamente archivos por su nombre
  find directorio **-name** archivo

- Buscar en un directorio recursivamente archivos que pesan más de 1 megabyte
  find directorio **-size** +1M

Comandos: comodines

- En los comandos de Bash existen algunos caracteres especiales que sirven de “comodines”
- Buscar en un directorio recursivamente archivos que en su nombre tienen la palabra “.jpg”
  find directorio **-name** “***.jpg”
- Buscar una palabra dentro de todos los archivos del directorio actual de forma **NO r**ecursiva
  grep palabra ***
- Copiar todos los archivos y directorios del directorio actual de forma **r**ecursiva al directorio destino
  cp -r * directorio_destino

Comandos: opciones combinadas

- Muchas veces es posible combinar las “opciones” de los comandos teniendo así afectos múltiples:
    - Listar los archivos de un directorio con formato “largo” y ordenados de arriba a abajo según fecha de modificación:
      ls -lt directorio
    - Borrar todos los archivos (y directorios) de un directorio de forma **r**ecursiva y **f**orzando su borrado:
      rm -rf directorio
    - Buscar en un directorio recursivamente archivos que pesan más de 5 megabyte y en su nombre tienen la palabra “.jpg”
      find directorio **-name** “*.jpg” **-size** +5M

Comandos: empaquetado y compresión

- **Empaquetado:** se unen varios archivos en uno solo (tar)
  tar -cvf empaquetado.tar archivo1 archivo2 archivo3
  tar -xvf empaquetado.tar

- **Compresión:** se reduce el tamaño del archivo (gzip/bzip2/7zip/Etc)
  gzip empaquetado.tar
  gzip -d empaquetado.tar.gz

- **Las dos:** el comando **tar** puede invocar a **gzip** (argumento z)
  tar -cvzf compripaquetado.tar.gz archivo1 archivo2 archivo3
  tar -xvzf compripaquetado.tar.gz

Comandos: empaquetado y compresión

- **Anatomía del comando tar:**

  tar -cvzf compripaquetado.tar.gz archivo1 archivo2 archivo3

  - **opción c**: compresión (empaquetado)
  - **opción v**: verbose (logea sus acciones en pantalla)
  - **opción z**: utiliza gzip para comprimir
  - **opción f**: genera un archivo (siempre va y siempre al final)

  tar -xvzf compripaquetado.tar.gz

  - **opción x**: extracción (desempaqueta y/o descomprime)

Comandos: empaquetado y compresión

- **Compresión con zip:** se reduce el tamaño del archivo
  zip -r archivo_coprimido.zip directorio
  unzip archivo_coprimido.zip

  - Visualizar el contenido el comprimido zip sin extraerlo
    zipinfo archivo_coprimido.zip

  - Visualizar el contenido de archivos comprimidos con gzip sin extraerlo
    zcat archivo _ comprimido.tar.gz

Comandos: entradas y salidas

- Los procesos (programas en ejecución) normalmente cuentan con 3 “archivos” abiertos:
  - **stdin**: entrada estándar, por defecto el teclado.
  - **stdout**: salida estándar, por defecto el monitor (consola).
  - **stderr**: error estándar, por defecto la **stdout**.
- Se identificar en el **S.O.** con un número, el *file descriptor*:
  - El número **0** para la **stdin**
  - El número **1** para la **stdout**
  - El número **2** para la **stderr**

- Ejemplo:
  - Es posible desviar la **stderr** de un proceso mediante una **redirección**:
    cat archivo_inexsistente **2>** error_cat.txt

Comandos: redirecciones

- Los comandos de Bash pueden “redirecciones” su salida estándar de varias formas:
  - Redirección destructiva:
    comando **>** *archivo*
    - Si *archivo* no existe, se crea.
    - Si *archivo* existe, sobreescribe (borra todo y escribe).
  - Redirección **no** destructiva:
    comando **>>** *archivo*
    - Si *archivo* no existe, se crea.
    - Si *archivo* existe, agrega al final sin borrar nada.

- Ejemplo:
  - Sobrescribir la fecha y hora actual en un archivo llamado fecha.txt
    date > fecha.txt

  - Crear un archivo y escribir dentro
    > archivo_nuevo

Comandos: redirecciones

- Redirección para utilizar un archivo como stdin (que no sea el teclado) en un comando
  comando **<** archivo

- Ejemplos:
  - Buscar una palabra en un archivo
    grep palabra **<** archivo

  - Realizar una acción por cada línea en un archivo
    while read linea; do; echo "$linea"; done **<** archivo

Comandos: redirecciones

- Pipes (tuberías): redirecciones para utilizar la stdout de un comando como stdin de otro comando
  comando1 | comando2 | comando3

- Ejemplos:
  - Filtrar los comandos del historial que tenga una palabra específica
    history **|** grep palabra

  - Cortar un archivo por su segunda columna delimitando por el carácter “:” y obtener las filas con valores únicos en orden alfabético
    cat archivo **|** cut -d: -f2 **|** sort -u


Bash Intermedio

Expresiones Regulares ~

- Las expresiones regulares o regexp son un medio para describir patrones de texto.
  - Expresiones Alternativas: una palabra o la otra
    grep -E **“palabra1|palabra2”** archivo

  - Contenedores: se busca un grupos de elementos
    grep -E “[0-5]” archivo
    grep -E “[a-o]” archivo
    grep -E “[a b c]” archivo

  - Cuantificadores: cantidad de veces que ese elemento aparece
    - “?” el elemento aparece 0-1 veces
    - “*” el elemento aparece 0-N veces
    - “+” el elemento aparece al menos 1 vez
    - “{N}” el elemento aparece N veces
    - “{N,M}” el elemento aparece entre N y M veces

Expresiones Regulares ~

- Puntos de anclaje
  - “^” inicio de línea
  - “$” fin de línea
  - “\b” límite de palabra

Ejemplo

- Buscar líneas que empiezan con “B”
  grep -E “^B” archivo
- Buscar líneas que empiezan con “s”
  grep -E “s$” archivo
- Buscar palabras que empiezan con “alg”
  grep -E “\balg” archivo
- Buscar palabras que terminan con “d”
  grep -E “\bd” archivo

Variables

- Bash soporta números racionales (enteros y con decimales), strings (cadenas de caracteres) y arreglos
- Los nombres de las variables son *Case Sensitive*, es decir, son sensibles a las mayúscula y minúsculas.
    - Por ejemplo: la variable **NOMBRE** no es la misma que las variables **Nombre** y **nombre**.
- Para declarar una variable se debe escribir el nombre y su valor luego del carácter igual “**=**” y **SIN DEJAR ESPACIOS**.
    - Ejemplos:
      shell=”Bash”
      nombre_usuario=”Alan Turing”
      id_usuario=42

Variables

- Para acceder al valor de una variable en Bash se debe anteponer el carácter **$**

- Ejemplos:
  - Es posible incluir las variables dentro de la llamada a un comando Bash:

    echo “La Shell que usa el usuario **$nombre_usuario** es **$shell** y su id es **$id_usuario**”

    cat /etc/passwd | grep **$nombre_usuario**

Variables: Arreglos

- En Bash se pueden utilizar Arreglos
  - Crear un arreglo vacío
    arreglo_a=**()**
  - Crear un arreglo inicializado con valores
    arreglo_b=**(0 1 1 2 3 5 8 13 21 34 55)**
  - Asignación de un valor en una posición específica
    arreglo_b[2]=valor
  - Acceso a un valor del arreglo
    echo **${arreglo_b[2]}**
    variable=**${arreglo_b[2]}**

Variables: Arreglos

- En **Bash** los índices de los arreglos comienzan en 0 pero en otras Shells esto puede cambiar, por ejemplo en ZSH comienzan en 1.

- Acceso a todos los valores del arreglo
  ${arreglo[@]}

- Obtener el tamaño total del arreglo
  ${#arreglo[@]}

- Borrado de un elemento (reduce el tamaño pero no elimina la posición, la deja vacía)
  unset arreglo[2]

Variables: Arreglos

- Ejemplos:

```
arreglo=(0 1 1 2 3 5 8 13 21 34 55)
arreglo[2]=spam
echo “El primer elemento es ${arreglo[0]}”
echo “El tercer elemento es ${arreglo[2]}”
echo “La longitud es: ${#arreglo[@]}”
echo  “Todos los elementos: ${arreglo[@]}”
```

Reemplazo de Comandos

- Permite utilizar la salida de un comando como una cadena de texto normal
- Permite guardarlo en variables o utilizarlos directamente
- Ejemplos:
  - Guardar los archivos del directorio actual en una variable
    variable=**”$(ls)”**

  - Guardar en una variable la cantidad de procesos con un nombre en ejecución actualmente
    variable=**”$(ps -A | grep nombre | wc -l)”**

  - Guardar en una variable la cantidad de memoria RAM en uso actualmente
    variable=**”$(free -mh | awk -F" " '{print $3}' | head -2 | tail -1)”**

Bash Scripting

Cómo hacer Script de Bash

- Crear un archivo con cualquier editor de texto (nano).
- En la primer línea es importante indicar el intérprete
  #!/bin/bash
  Esta línea, conocida como SheBang, especifica el intérprete que se utilizará para ejecutar el *script*. Si no se especifica, se utiliza el intérprete por defecto del usuario.
  - ¿Qué problemas puede traer esto?
- Dentro del archivo se pueden escribir una serie de comandos, de a misma manera que los escribiríamos en la terminal.
- No es necesario guardar el archivo con **extensión .sh** en su nombre, pero es recomendable.
  - **¿Qué pasa si no tiene extensión sh?**
- No es necesario **asignar permisos de ejecución** al archivo, pero es recomendable.
  - **¿Qué pasa si no tiene permisos de ejecución?**

Un primer Script

- Ejemplo de Script de Bash:

  #!/bin/bash

  #Esto es un comentario
  #Los comentarios empiezan con el carácter numeral “#”

  echo “Hola mundo desde mi Bash Script”

Ejecución del Script

- Para ejecutar el script podemos
  - Ejecutarlo a través de una llamada a Bash
    bash /path/absoluto/mi_script.sh
    bash ../path/relativo/mi_script.sh

  - Ejecutarlo llamando directamente al script (debe tener permisos de ejecución)
    ./mi_script.sh
    /path/absoluto/mi_script.sh

  - Ejecutarlo con Bash y modo *debug*
    bash -x mi_script.sh

Estructuras de Control

Decisión (if)

```
if [ condición ]
then
        bloque
fi
```

Ejemplo

```
if [ “$(whoami)” = “root” ]
then
        apt update
        apt upgrade -y
fi
```

Selección (case)

```
case $variable in
“valor 1”)
block
;;
“valor 2”)
block
;;
*)
block
;;
esac
```

Ejemplo

```
case $opcion in
“agregar”)
funcion_agregar
;;
“borrar”)
funcion_borrar
;;
*)
funcion_menu
;;
esac
```

Estructuras de Control

Menú de Opciones (select)

- Permite que se seleccione un valor de entre un grupo de variables para utilizarlo como se desee dentro de la estructura como por ejemplo dentro de un **case**

```
select variable in opcion1 opcion2 opcion3
do
        # en variable está el valor elegido de las opciones
        bloque
done
```

Estructuras de Control

Ejemplo

```
select  accion in Nuevo Salir
do
        case $accion in
                “Nuevo”)
                        echo “La opción seleccionada fue Nuevo”
        ;;
                “Exit”)
                        exit 0
        ;;
        esac
done
```

Estructuras de Control

Iteración (for)

```
for (( i=0; i < 10; i++ ))
do
        bloque
done
```

Ejemplo

```
for puerto in 80 8080 443 20 21 22 25 143;
do
        echo “Se va a probar el puerto $puerto”
done
```

Iteración por lista de elementos (for in)

```
for elemento in elem1 elem2 elem3 elemN;
do
        bloque
done
```

Estructuras de Control

mientras (while)

```
while [ condicion ]
do
        bloque
done
```

Ejemplo

```
while [ $(who | grep root | wc -l ) -eq 0 ]
do
        echo “Esperando que logee root”
done
```

hasta (until)

```
until  [ condicion ]
do
        bloque
done
```

Ejemplo

```
until  [ $(ps -A | grep firefox | wc -l ) -eq 0 ]
do
        echo “Hasta que deje de ejecutarse Firefox”
done
```

Evaluación de Condiciones Lógicas

- Las condiciones lógicas normalmente se evalúan mediante
  [condicion **]**
  test condicion

Operador	Con Strings	Con números
Igualdad	“$nombre” = “Maria”	$edad -eq 20
Desigualdad	“$nombre” != “Maria”	$edad -ne 20
Mayor	A > Z	5 -gt 20
Mayor o Igual	A >= Z	5 -ge 20
Menor	A < Z	5 -lt 20
Mayor o Igual	A <= Z	5 -le 20

Cálculo

Menú de Opciones (select)

- En bash para realizar un cálculo hay distintas estrategias
  - Comando let
    i=0
    let i=i+5
    let i++
  - Paréntesis
    i=0
    i=$(**(**i + 5**)**)
  - Comando bc
    i=0
    i=$(echo “$i + 5” | **bc**)

Estructuras de Control

Break y continue

```
#!/bin/bash
i=0
while true
do
        let i++
        if [ $i -eq 6 ]; then
                break #Corta el while
        elif [ $i -eq 3 ]; then
                continue #Salta una iteración
        fi
echo $i
done
```

Condiciones Compuestas

Break y continue

- Dentro de las estructuras de control se pueden anidar condiciones mediante operadores lógicos
    - El operador **AND** se utiliza con los caracteres **&&**
      if [$a = $b] **&&** [$c -gt $d]
      then
      ...
    - El operador **OR** se utiliza con los caracteres **||**
      if [$a = $b] **||** [$c -gt $d]
      then
      ...

Argumentos y valores de retorno

Argumentos

- Los Script pueden recibir argumentos en su invocación-
- Para accederlos, se utilizan variables especiales:
  - **$0** contiene la invocación del script.
  - **$1**, **$2**, **$3**, ... contienen cada uno de los argumentos.
  - **$#** contiene la cantidad de argumentos recibidos.
  - **$*** contiene la lista de todos los argumentos recibidos.
  - **$?** contiene en todo momento el valor de retorno del último comando ejecutado.
- Ejemplo
  - Se invoca un script con 2 argumentos, se imprimen y se sale del script con código 0 (significa que se ejecutó correctamente)

    ./mi_script.sh **argumento1 arg2**

    #!/bin/bash
    echo “El argumento 1 es **$1** y el argumento 2 es **$2**”
    exit 0

Argumentos y valores de retorno

Valores de retorno

- Para terminar un script usualmente se utiliza la función **exit**
  - Causa la terminación inmediata del script
  - Puede devolver cualquier valor **entre 0 y 255**
    - El valor **0** indica que el script se ejecutó de forma **exitosa**
    - Un valor **distinto a 0** indica un código de **error**
    - Es posible **consultar** el código de **exit** a través de la variable **$?**
- Ejemplo

./mi_script.sh argumento1 arg2

```
#!/bin/bash
if [ $# -eq 2 ]; then
        echo “El argumento 1 es $1 y el argumento 2 es $2”
        exit 0
fi
echo “Se deben pasar exactamente 2 argumentos”
exit 1
```

Funciones

- Las funciones permiten modularizar el comportamiento de los *scripts*
- Se pueden declarar de 2 formas
  - function nombre_funcion { bloque }
  - nombre_funcion() { bloque }
- Con la sentencia **return** se retorna un valor entre 0 y 255
- No pueden retornar otra cosa distinta (strings, arreglos, valores > 255)
- El valor de retorno se puede evaluar mediante la variable **$?**
- Pueden recibir argumentos en las variables $1, $2, etc.

Funciones (ejemplo)

```
#!/bin/bash

mayor() {
        if [ $# -ne 2 ]; then
                echo " Se deben pasar 2 argumentos"
                exit 0
        fi
        echo "Se van a comparar los valores $*"

        if [ $1 -gt $2 ]; then
                echo "$1 es el mayor"
                return 1
        elif [ $ -lt $2 ]; then
                echo "$2 es el mayor"
                return 2
        fi
        echo "$1 y $2 son iguales"
        return 3
}
mayor 5 6 #invocacion de la función con sus argumentos
echo $? #imprime el estado de retorno de la función
```

Variables: Alcance y Visibilidad

- Las variables no inicializadas son reemplazadas por un valor nulo o 0, según el contexto de evaluación.
- Por defecto las variables son globales.
- Una variable local a una función se define con **local**

```
test () {
        local variable
}
```

- Las variables de entorno son heredadas por los procesos hijos.
- Para exponer una variable global a los procesos hijos se usa el comando **export**:

```
export VARIABLEGLOBAL=”Mi var global”
comando
#comando verá entre sus variables de entorno a VARIABLEGLOBAL
```

Variables de Entorno

- Las variables de Entorno son variables que sirven para facilitar a los procesos Unix el almacenamiento de cierta información de configuración
- En Unix cada proceso ejecutado o en ejecución declara sus propias variables de entorno y las pasa a sus procesos Hijos
    - Es por esto que al visualizar qué variables de entorno tenemos declaradas en nuestra terminal aparecen tantas variables
      printenv

    - En un Linux contenerizado habrá menos variable de entorno porque se ejecutaron menos procesos anidados

- Es posible declarar nuestras propias variables de entorno pero a menos que dicha declaración persista en el inicio de algún proceso que utilicemos no “persistirá” adecuadamente en cada invocación del mismo. Por ejemplo: la terminal o la shell
  export mi_variable=”Esta es mi variable de entorno a esta terminal”


¿Preguntas?