


# Introducción a los Sistemas Operativos

## Shell Scripting

### Práctica 3



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---


---

---

---

### ¿Qué es un shell script?

- Es un archivo con una secuencia de comandos de determinado shell (en nuestro caso **bash**)
- Un script de **bash** contiene
  - Comandos
  - Manejo de variables (no es fuertemente tipado)
  - Comentarios: líneas que comienzan con **#**
  - Estructuras de control
    - **if**
    - **case**
    - **for**
    - **while**
  - Declaración de funciones
  - Puede usar código definido en otros scripts (comando **source**)



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---


---

---

---

### ¿Qué es un shell script?

- En un shell script se puede hacer todo lo que se puede hacer de forma interactiva y viceversa
  - **echo, ls, cat, cd, if, while, etc...**
- ¿Por qué escribir un shell script en vez de un programa en **c/java/python/perl/ruby...**?
  - Es ideal para automatizar tareas que implican usar comandos
  - La ejecución de comandos es directa (con su nombre basta)
  - Es muy simple guardar el resultado de la ejecución de un comando
  - No requiere la instalación de ningún interprete o compilador (todo GNU/Linux tiene al menos un shell)
  - GNU/Linux expone gran parte del sistema como archivos, y es muy fácil manipular archivos desde un shell



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Comandos útiles

(aunque algunos en principio no lo parezcan)

- Imprimir el contenido de un archivo
  - `cat archivo`
- Imprimir texto
  - `echo "Hola mundo"`
- Leer una línea desde entrada estándar en la variable `var`
  - `read var`
- Quedarme con la primer columna de un texto separado por : desde entrada estándar
  - `cut -d: -f1`
- Contar la cantidad de líneas que se leen desde entrada estándar
  - `wc -l`
- Buscar todos los archivos que contengan la cadena `pepe` en el directorio `/tmp`
  - `grep pepe /tmp/*`

**Ayuda sobre comandos:**  
◦ `man` (para comandos externos)  
◦ `help` (para comandos internos al shell)



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Empaquetadores y Compresores

- El concepto de Empaquetado hace referencia a unir varios archivos en uno solo resultante:
  - `tar -cvf archivo.tar archivo1 archivo2 archivo 3`
- Para desempaquetar se utiliza:
  - `tar -xvf archivo.tar`
- La compresión se realiza sobre un único archivo:
  - `gzip archivo.tar` → El archivo resultante será un archivo empaquetado y comprimido → `archivo.tar.gz`
- Para descomprimir se utiliza:
  - `gzip -d archivo.tar.gz` → El archivo resultante es uno empaquetado
- Se puede empaquetar y comprimir con un único comando:
  - `tar -cvzf archivo.tar archivo1 archivo2 archivo 3`
- Para descomprimir y desempaquetar se utiliza:
  - `tar -xvzf archivo.tar`



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Redirecciones

- Al utilizar redirecciones mediante `>` (**Redirección Destructiva**)
  - Si el archivo de destino no existe, se lo crea;
  - Si el archivo existe, se lo **trunca**;
  - y se escribe la información en él.
- En cambio, al utilizar redirecciones mediante `>>` (**Redirección NO Destructiva**)
  - Si el archivo de destino no existe, se lo crea;
  - Pero si el archivo existe, no se lo trunca;
  - y se agrega la información al final del mismo.
  - Funciona como un **append**.

```
cd
ls >> /tmp/lista.txt
cd /tmp
ls >> /tmp/lista.txt
```

¿Qué pasa si cambiamos `>>` por `>`?



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Entrada/Salida/Error estándar

- Todo programa en GNU/Linux tiene 3 archivos abiertos con un número de *descriptor*
  - 0 entrada estándar (por defecto el teclado)
  - 1 salida estándar (por defecto la pantalla)
  - 2 error estándar (por defecto la pantalla)
- Es posible:
  - desviar la salida de un comando a un archivo con `>`  
`ls > lista.txt`
  - Volcar los errores a un archivo con `2>`  
`find . -name foo 2> errores.txt`
  - Redirigir la salida de un comando hacia otro mediante `|` (tuberías o pipes)  
`ls | grep .sh`
  - Se pueden encadenar y combinar  
`ls | grep .sh > scripts.txt`



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Uso de Pipes |

- El `|` nos permite comunicar dos procesos por medio de un pipe o tubería desde la shell.
- El pipe conecta stdout (salida estándar) del primer comando con la stdin (entrada estándar) del segundo comando.
- Por ejemplo:
  - `ls | more`
    - Se ejecuta el comando `ls` y la salida del mismo, es enviada como entrada del comando `more`
  - Se pueden anidar tantos pipes como se deseen:
  - ¿Cómo haríamos si quisiéramos contar la cantidad de usuarios del sistema que en su nombre de usuario aparece una letra a?

`cat /etc/passwd | cut -d: -f1 | grep a | wc -l`



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## ¿Como creamos un shell script?

- Creamos un archivo con cualquier editor de texto
- Opcionalmente, en la primera línea indicamos cual será el intérprete del comando (en nuestro caso bash) → `#!/bin/bash`
- Si no especificamos el intérprete explícitamente, se asume el intérprete por defecto del usuario que ejecuta el script. ¿Qué problemas puede traer esto?
- El resto del archivo se compone de líneas de comandos, tal cual como si los estuviéramos usando de forma interactiva.
- Al guardar el script (por convención) utilizaremos `.sh` a modo de terminación de los archivos para indicar que son shell scripts.



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Primer ejemplo - Hola mundo

```
#!/bin/bash

# Si la primer línea de mi script comienza
# con la cadena #! se interpretará como el
# path al intérprete a utilizar (podría ser
# python, perl, php, etc...)

# Ahora el script en sí
echo "Hola mundo"
```

Ejemplo:  
1holaMundo.sh



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Ejecución del script anterior

Para ejecutar un script tenemos varias formas de hacerlo:

- Desde la línea de comandos podemos ejecutarlo pasándoselo al comando `bash` como argumento:  
\$ `bash miscript.sh`
- Le podemos dar permisos de ejecución e invocarlo utilizando su path *relativo*:  
\$ `./miscript.sh`
- ...o mediante su path absoluto:  
\$ `/home/yo/miscript.sh`
- O podemos también ejecutarlo en modo *debug* (depuración)  
\$ `bash -x miscript.sh`

¿Por que hay que ejecutarlo usando ruta relativa o completa?

→ ¿Qué ocurre si en `/bin` se agrega un script con el mismo nombre?



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Variables

- `bash` no es fuertemente tipado.
  - Las variables pueden cambiar su tipo durante la ejecución de nuestros scripts.
- Las variables tienen básicamente dos tipos:
  - String
  - Array
- No precisan declaración, se las utiliza directamente.
- Las asignaciones tienen la forma (notar que **no hay espacios** entre el signo `=` y los operandos):  
`variable=valor`
- Como en el resto de las situaciones referentes al shell, `bash` es **case sensitive**.
- Para referenciar su valor se utiliza el símbolo `$` y opcionalmente `{ }`



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Variables (cont)

- Los nombres de las variables pueden contener mayúsculas, minúsculas, números y el símbolo `_` (*underscore*), pero **no pueden empezar con un número**.

```
NOMBRE="Fulano De Tal"
facultad=Informatica
carrera_1="Licenciatura en Sistemas"
carrera_2="Licenciatura en Informatica"
echo El alumno $NOMBRE de la Facultad de $facultad
curso $carrera_1 y $carrera_2
# imprime:
# El alumno Fulano De Tal de la Facultad de
# Informática curso Licenciatura en Sistemas
# y Licenciatura en Informatica
```

Ejemplo:

`2carreras.sh`



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Variables (cont)

```
#!/bin/bash

nombre=Carlos
echo "Hola $nombre" # Hola Carlos
echo Hola ${nombre} # Hola Carlos

nombre=5
echo "Hola $nombre" # Hola 5
```

Ejemplo:

`3variables.sh`



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Variables: Arreglos

- Creación:**  
`arreglo_a=()` # Se crea vacío  
`arreglo_b=(1 2 3 5 8 13 21)` # Inicializado
- Asignación** de un valor en una posición concreta:  
`arreglo_b[2]=spam`
- Acceso** a un valor del arreglo (En este caso las llaves **no son opcionales**):  
`${arreglo_b[2]}`
- Acceso a todos** los valores del arreglo (Es indistinto usar `@` ó `*` como índice):  
`${arreglo[@]}` # == `${arreglo[*]}`



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Variables: Arreglos (cont)

- **Tamaño del arreglo:**  
`${#arreglo[@]} # == ${#arreglo[*]}`
- **Borrado de un elemento** (no elimina la posición, solamente la deja vacía):  
`unset arreglo[2]`
- Los índices en los arreglos comienzan en 0



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Ejemplo: Arreglos

```
#!/bin/bash

arreglo=(1 2 3 5 8 13 21)
arreglo[2]=spam
echo "El Primer elemento es ${arreglo[0]}"
echo "El tercer elemento es ${arreglo[2]}"
echo "La Longitud: ${#arreglo[*]}"
echo "Todos sus elementos: ${arreglo[*]}"
```

Ejemplo:  
`4arreglos.sh`



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Uso de comillas y expansión de parámetros

- En ocasiones, se pueden omitir las comillas al ingresar valores string. Por ejemplo:
  - si el string se compone de una sola palabra,
  - en el parámetro para el comando `echo`,
  - o cuando sabemos que el valor de una variable no tiene espacios o es un número.
- A la hora de utilizar las comillas, tenemos dos tipos de comillas que se comportan distinto:
  - *Comillas dobles*: los string delimitados por estas comillas son evaluados y se reemplazan en ellos expresiones o se expanden variables.
  - *Comillas simples*: Se toman como literales, no se efectúa ningún reemplazo en ellas.

Ejemplo:  
`nombre=Pepe`  
`echo "$nombre" # imprime Pepe`  
`echo '$nombre' # imprime $nombre`



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Uso de comillas y expansión de parámetros

### Ejemplo

Un ejemplo:

```
variable="un texto de varias palabras"
variable_2=UnaSolaPalabra

echo "Podemos leer $variable"
echo 'No podemos leer $variable'

variable_3="Asi concateno $variable_2 a otro
string"

¿Qué se imprime en cada caso?
¿Cuál es el valor de variable_3?
```



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Sustitución de comandos

- Permite utilizar la salida de un comando como si fuese una cadena de texto normal.
  - Permite guardarlo en variables o utilizarlos directamente.
- Se la puede utilizar de dos formas, cada una con distintas reglas:
  - `$(comando_valido)`
  - ``comando_valido``

**Nota:** La primer forma resulta más clara y posee reglas de anidamiento de comandos más sencillas.

Ejemplo:

```
arch="$(ls)" # == arch="\ls`" == arch=`ls`
mis_archivos="$(ls /home/$(whoami))"
```



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Estructuras de control

• Selección de alternativas:

◦ <b>Decisión:</b>	◦ <b>Selección:</b>
<pre>if [ condition ] then   block fi</pre>	<pre>case \$variable in   "valor 1")     block     ;;   "valor 2")     block     ;;   *)     block     ;; esac</pre>



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Estructuras de control (cont)

- Menú de opciones:

```
select variable in alternativa1 alternativa2
alternativa3
do
    # en $variable está el valor elegido
    block
done
```

**Ejemplo:**

```
select action in new exit
do
    case $action in
        "new")
            echo "Selected option is NEW"
            ;;
        "exit")
            exit 0
            ;;
        esac
done
```

1) new  
2) exit  
#?



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Estructuras de control (cont)

- Iteración - bloques FOR:

- Al estilo de C:

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

- Con lista de valores (notar diferencia con Pascal!!!):

```
for i in value1 value2 value3 valueN;
do
    block
done
```



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Estructuras de control (cont)

- Iteración - Bloques WHILE y REPEAT (until):

```
while [ condition ] #Mientras se cumpla
                    # la condición
```

```
do
    block
done
```

```
until [ condition ] #Mientras no se
                    #cumpla la condición
```

```
do
    block
done
```



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---



## Evaluación de condiciones lógicas

- Las condiciones lógicas normalmente se evalúan mediante:
  - [ **condition** ]
  - test **condition**
- Operadores para **condition**:

Operador	Entre Variables String	Entre Números
Igualdad	\$nombre == "Maria"	\$nombre -eq "Maria"
Desigualdad	\$nombre != "Maria"	\$nombre -ne "Maria"
Mayor	A > Z	2 -gt 3
Mayor o igual	A >= Z	2 -ge 3
Menor	A < Z	2 -lt 3
Menor o igual	A <= Z	2 -le 3
Negación	! expresion	! expresion



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Estructuras de control: ejemplos

```
if [ "$USER" == root ]
then
    echo "super user"
else
    echo "Ud. es $USER"
fi

n=0
while [ $n -ne 5 ]; do
    echo $n
    let n++
done

for archivo in $(ls)
do
    echo "- $archivo"
done

for ((i=0; i<5; i++))
do
    echo $i
done
```

- Adicionalmente:
  - break [n] corta la ejecución de n niveles de bucles
  - continue [n] salta a la siguiente iteración del enésimo loop que contiene esta instrucción



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Estructuras de control: break

```
#!/bin/bash
# Imprime los numeros del 1 al 5
# (no es un código para nada elegante)
i=0
# true es un comando que siempre retorna 0
while true
do
    let i++ # Incrementa i en 1
    if [ $i -eq 6 ]; then
        break # Corta el loop (while)
    fi
    echo $i
done
```



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Estructuras de control: *continue*

```
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
```

¿Qué hace el script anterior?



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Estructuras de control: condiciones complejas

```
# AND
if [ $a = $b ] && [ $a = $c ]
then
    #...

# OR
if [ $a = $b ] || [ $a = $c ]
then
    #...
```



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Argumentos y valor de retorno

- Los scripts pueden recibir argumentos en su invocación
  - Para accederlos, se utilizan variables *especiales*:
    - \$0 contiene la invocación al script
    - \$1, \$2, \$3, ... contienen cada uno de los argumentos
    - \$# contiene la cantidad de argumentos recibidos
    - \$\* contiene la lista de todos los argumentos
    - \$? contiene en todo momento el valor de retorno del **último** comando ejecutado
- ```
if [ $# -ne 2 ]; then
    exit 1 # Error
else
    echo "Nombre: $1, Apellido: $2"
fi
exit 0      # Funcionó correctamente
```



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Función exit

- Causa la terminación de un script
- Puede devolver cualquier valor entre 0 y 255:
  - El valor 0 indica que el script se ejecutó con éxito
  - Un valor distinto indica un código de error
- Se puede consultar el exit status imprimiendo la variable \$?



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Funciones

- Se pueden declarar de 2 formas
  - `function nombre { block }`
  - `nombre() { block }`
- Con la sentencia `return` se retorna un valor entre 0 y 255
- El valor retornado se puede evaluar con la variable \$?
- Reciben argumentos en las variables \$1, \$2, etc.



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Funciones: ejemplos

```
# recibe 2 argumentos y retorna:
# 1 si el primero es el mayor
# 0 en caso contrario
mayor() {
    echo "Se van a comparar los valores $*"
    if [ $1 -gt $2 ]; then
        echo "$1 es el mayor"
        return 1
    fi
    echo "$2 es el mayor"
    return 0
}
mayor 5 6 # Invocación
echo $?   # Imprime el resultado
```



Facultad de Informática  
UNIVERSIDAD NACIONAL DE LA PLATA

---

---

---

---

---

---

---

---

## Variables: alcance y visibilidad

- Las variables no inicializadas son reemplazadas por **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 `export`

```
export  
export VARIABLE_GLOBAL="Mi var global"  
comando  
# comando verá entre sus variables de  
# entorno a VARIABLE_GLOBAL
```



---

---

---

---

---

---

---