

# Sesión 13: `man` `bash`

Programación para Sistemas

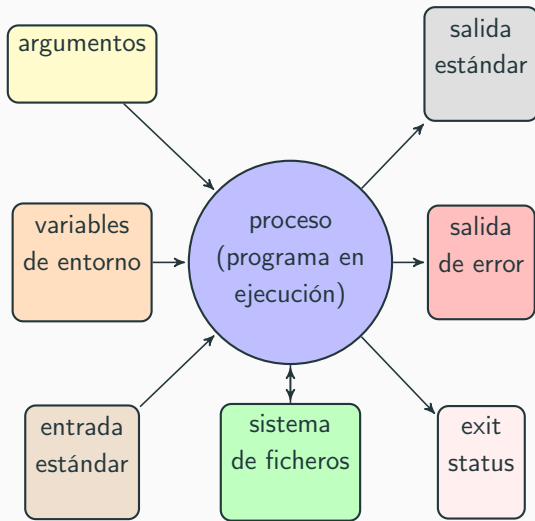
---

Ángel Herranz

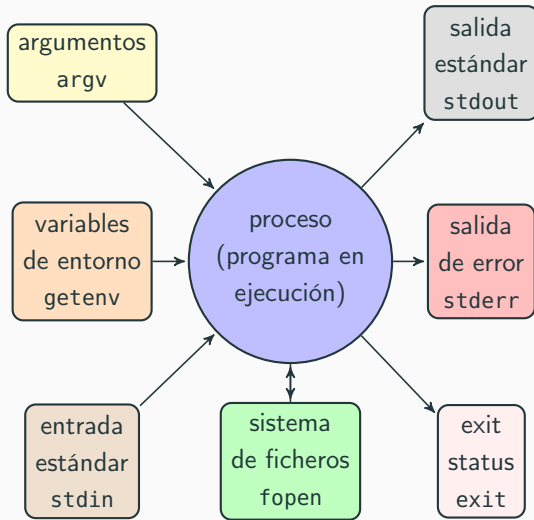
2020-2021

Universidad Politécnica de Madrid

# Recordatorio i

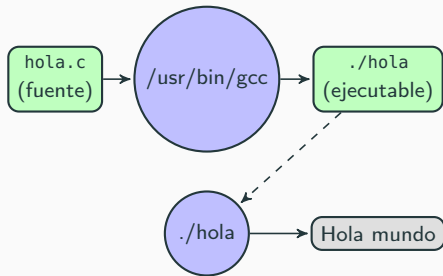


# Recordatorio i



# Recordatorio ii

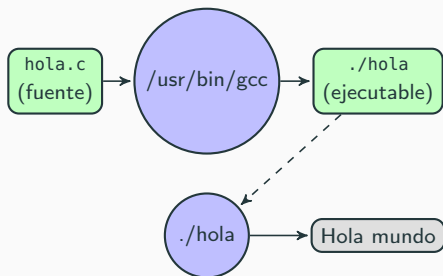
C (compilación y ejecución)



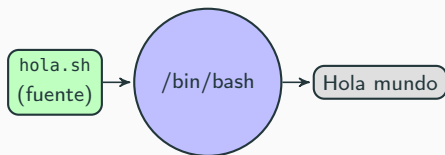
Bash (interpretado)

# Recordatorio ii

C (compilación y ejecución)



Bash (interpretado)



# Recordatorio iii

1. Crear el fichero script con tu editor favorito:

```
#!/bin/bash  
echo Hola mundo
```

2. Darle permisos

```
$ chmod +x hola.sh
```

3. Ejecutar

```
$ ./hola.sh
```

```
Hola mundo
```

## man bash

- Expansión,
- redirección,
- funciones,
- control de flujo, y
- operadores de control.
- *+ Scripts*

# Expansión: *parámetros* (aka variables)

🕒 3' 🔍 man bash y busca la sección *EXPANSION*, en concreto *Parameter Expansion*:



# Expansión: *parámetros* (aka variables)

- 🕒 3' 🔍 man bash y busca la sección *EXPANSION*, en concreto *Parameter Expansion*:

`${parameter}`

- Se puede escribir sin llaves: `${A} ≡ $A`
- *Parameter Expansion* (variantes):

`${parameter:-word}`

`${parameter:?word}`

`${parameter:offset:length}`

`${#parameter}`

etc.

- 🕒 2' 💻 FECHA=2019-12-17 y entonces ¿qué significa `${#FECHA}` o `${FECHA:2:2}`?

# Expansión: *command substitution*

🕒 3' 💻 Guardar en FECHA el resultado de `date +%Y-%m-%d`

---

<sup>1</sup>Comillas invertidas

# Expansión: *command substitution*

🕒 3' 💻 Guardar en FECHA el resultado de `date +%Y-%m-%d`

- man bash y busca **Command Substitution**:

`$(command)`

---

<sup>1</sup>Comillas invertidas

# Expansión: *command substitution*

🕒 3' 💻 Guardar en FECHA el resultado de `date +%Y-%m-%d`

- man bash y busca **Command Substitution**:

`$(command) ≡ 'command'`<sup>1</sup>

---

<sup>1</sup>Comillas invertidas

# Expansión: *command substitution*

🕒 3' 💻 Guardar en FECHA el resultado de `date +%Y-%m-%d`

- man bash y busca **Command Substitution**:

`$(command) ≡ 'command'`<sup>1</sup>

- Es una expansión **extraordinariamente útil**
- Ejemplo:

`$ FECHA='date +%Y-%m-%d'`

---

<sup>1</sup>Comillas invertidas

# Expansión: *command substitution*

🕒 3' 🖥 Guardar en FECHA el resultado de `date +%Y- %m- %d`

- `man bash` y busca **Command Substitution**:

`$(command) ≡ 'command'`<sup>1</sup>

- Es una expansión **extraordinariamente útil**
- Ejemplo:

```
$ FECHA='date +%Y- %m- %d'
```

- Y luego...

```
$ echo ${FECHA}
```

```
$ echo ${FECHA:5:2}
```

---

<sup>1</sup>Comillas invertidas

No es necesario aprenderse el manual

No es necesario aprenderse el manual

Pero es necesario aprender a usarlo



En el manual de Bash se puede leer la siguiente descripción sobre expansión de variables (en realidad es más compleja pero estas líneas son suficientes):

```
${!prefix*}
```

Nombres que encajan con prefijo. Expande a los nombre de variables que empiezan por prefix separados por espacios y ordenados por orden alfabético.

**Se pide** escribir las cuatro líneas de la salida estándar resultado de la ejecución de los siguientes mandatos Bash, suponiendo que no hay otras variables definidas que empiecen por “MIV”:

```
$ MIVUNO=
$ MIVDOS=
$ MIVTRES=
$ MIVCUATRO=
$ MIVCINCO=
$ echo ${!MIV*}
$ echo ${!MIVC*}
$ echo ${!MIVT*}
$ echo ${!MIVU*}
```

# Redirección

Q man bash y busca la sección *REDIRECTION*, en concreto *Redirecting Input* y *Redirecting output*:

# Redirección

Q man bash y busca la sección *REDIRECTION*, en concreto *Redirecting Input* y *Redirecting output*:

**command** < file

**command** > file

**command** >> file

**command** 2> file

**command** 2>> file

# Redirección

- Q man bash y busca la sección *REDIRECTION*, en concreto *Redirecting Input* y *Redirecting output*:

**command** < file

**command** > file

**command** >> file

**command** 2> file

**command** 2>> file

- Otras redirecciones *muy* interesantes:

**command** 2>&1

**command** > file 2>&1  $\equiv$  **command** &> file

# Redirección

- Q man bash y busca la sección *REDIRECTION*, en concreto *Redirecting Input* y *Redirecting output*:

**command** < file

**command** > file

**command** >> file

**command** 2> file

**command** 2>> file

- Otras redirecciones *muy* interesantes:

**command** 2>&1

**command** > file 2>&1  $\equiv$  **command** &> file

- *Más*: *here documents*, *duplicating file descriptors*, etc.

## +Scripts

---

# Recordatorio iii (script a mano)

1. Crear el fichero script con tu editor favorito:

```
#!/bin/bash  
echo Hola mundo
```

2. Darle permisos

```
$ chmod +x hola.sh
```

3. Ejecutar

```
$ ./hola.sh
```

```
Hola mundo
```

# Funciones

- **Declaración** (tres alternativas equivalentes)

```
function f() {  
  ...  
}
```

```
function f {  
  ...  
}
```

```
f() {  
  ...  
}
```

- **Uso:** igual que cualquier programa

f arg1 arg2 arg3 ...

⚠ Cuidado con *tapar* un programar (imagina que llamas ls a una función)

📄 Implementa un script que haga uso de una función. Por ejemplo “usage” que dice cómo usar el script.



# Funciones: parámetros y return

- **Parámetros:** son implícitos

```
function f() {  
    echo "Num args = $#"  
    echo "Arg 0 = $0"  
    echo "Arg 1 = $1"  
    ...  
}  
f 1 dos III
```

- **return:** se refleja en el exit status

```
function f() {  
    return 1;  
}  
f  
echo $?
```

## man bash y buscar **shift**

`shift [n]`

The positional parameters from `n+1 ...` are renamed to `$1 ....` Parameters represented by the numbers `$#` down to `$(n+1)` are unset. `n` must be a non-negative number less than or equal to `$#`. If `n` is 0, no parameters are changed. If `n` is not given, it is assumed to be 1. If `n` is greater than `$#`, the positional parameters are not changed. The return status is greater than zero if `n` is greater than `$#` or less than zero; otherwise 0.

## man bash y buscar **shift**

`shift [n]`

The positional parameters from `n+1 ...` are renamed to `$1 ....` Parameters represented by the numbers `$#` down to `$#-n+1` are unset. `n` must be a non-negative number less than or equal to `$#`. If `n` is `0`, no parameters are changed. If `n` is not given, it is assumed to be `1`. If `n` is greater than `$#`, the positional parameters are not changed. The return status is greater than zero if `n` is greater than `$#` or less than zero; otherwise `0`.

- Muy usado para iterar y recorrer todos los argumentos

 Escribir en la salida estándar una línea por argumento

# Funciones: ámbito

- **Parámetros**: no se hacen explícitos  
     `$#, $0, $1, $2, ...`
- Variables **globales** por defecto
- Pero se pueden **definir locales**

```
X=2
function f() {
    X=1
    echo $X
    ...
}
f
echo $X
```

```
X=2
function f() {
    local X=1
    echo $X
    ...
}
f
echo $X
```

# Control de flujo i

- **if**

**if** command; **then** command; **else** command; **fi**

≡

**if** list; **then** list; **else** list; **fi**

- **for**

**for** name **in** words: **do** list; **done**

- Otras estructuras en el **manual**:

**case**, **while**, etc.

# Control de flujo ii

- El programa **test**: man **test**
- Comprobaciones sobre **ficheros y strings**:

## SYNOPSIS

test EXPRESSION

...

EXPRESSION sets exit status. It is one of:

...

-n STRING

the length of STRING is nonzero

STRING1 = STRING2


the strings are equal

...

-e FILE

FILE exists

# Control de flujo iii

 Ejecuta `which` [

 ¿Qué es `which` [?

 Ejecuta

```
$ [
```

```
$ [ ]
```

```
$ [ -n "Hola"]
```


```
$ [ -z "Hola"]
```

```
$ test -n "Hola"
```

```
$ test -z "Hola"
```

 ¿Qué está pasando?

# Control de flujo iii

 Ejecuta `which [`

 ¿Qué es `which [?`

 Ejecuta

```
$ [
```

```
$ [ ]
```

```
$ [ -n "Hola"]
```

```
$ [ -z "Hola"]
```

```
$ test -n "Hola"
```

```
$ test -z "Hola"
```

 ¿Qué está pasando?

```
$ if [ -e $VIDEO ]; echo "$VIDEO existe"; fi
```



# Control de flujo iv

- Operadores de control

`command ; command` (también `command ;`)

`command & command` (también `command &`)

`command | command`

`command && command`

`command || command`

# Control de flujo iv

- Operadores de control

**command ; command** (también **command ;**)

**command & command** (también **command &**)

**command | command**

**command && command**

**command || command**



Observad las equivalencias que usan los *hackers*

```
command1 && command2
```

```
command1 || exit 1  
command2
```

```
if command1; then command2; fi
```

```
if command1; then  
    command2;  
else  
    exit 1;  
fi
```

# Pistas para yourmp3.sh i

- youtube-dl --get-filename URL saca por la salida estándar el nombre del fichero a descargar pero no lo descarga
- Nombre del fichero sin extensión: \${VIDEO%.\*}
- Recuerda la línea para ffmpeg

```
ffmpeg -i "$VIDEO" -vn -ab 128k -ar 44100 -y "$MP3"
```

- Las comillas importan por los posibles espacios en el nombre del fichero
- Por si hiciera falta:
  - Extensión de un fichero: \${VIDEO##\*.}
  - Echa un ojo al programa basename

# Pistas para yourmp3.sh ii

```

TMPDATA=${0}.tmp
URL=$1
if [ -i ./youtube-dl --get-filename ${URL} 2> /dev/null ] > $TMPDATA; then
    echo "Problema descargando ${URL}" 1&>2
    exit 1
fi
./youtube-dl ${URL}
# Cuidado con el nombre del fichero, la extensión puede ser distinta
# a la del nombre indicado en $TMPDATA
VIDEO=$(cat $TMPDATA)
VIDEO="${VIDEO%.*}"
MP3="${VIDEO%.*}.mp3"
VIDEO=$(echo "${VIDEO}".*)
ffmpeg -i "$VIDEO" -vn -ab 128k -ar 44100 -y "$MP3"
shift
# Y a volver a empezar
[ -z $# ] || exit 0
```