Sesión 11: *Scripts*

Programación para Sistemas

Ángel Herranz

2022-2023

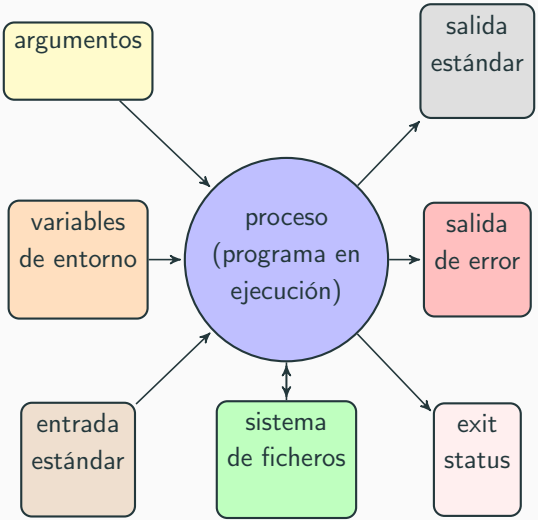
Universidad Politécnica de Madrid

Encuestas

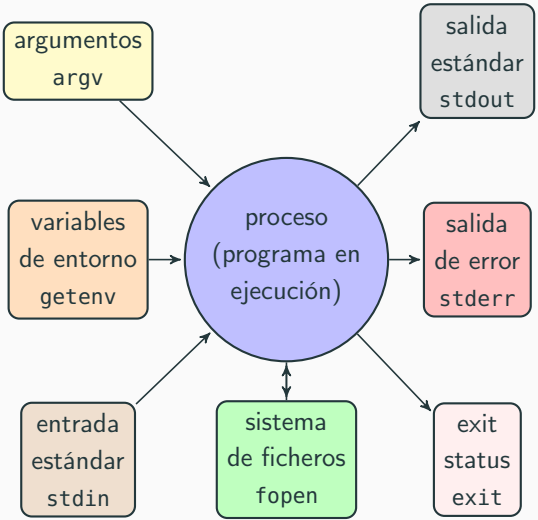
http://servicios.upm.es/encuestas

Es importante para identificar problemas y mejorar

Recordatorio i



Recordatorio i



Recordatorio ii

`man bash`

- Expansión (`${...}`),
- redirección (`<`, `>`, `|`),
- control de flujo (**`if`**, **`for`**, **`case`**), y
- operadores de control (`;`, `&`, `&&`, `||`).

Recordatorio iii

Usa el manual para saber lo que hacen los siguientes *clásicos*

awk, basename, **cat**, cut, **echo**, find, grep, head, join, locate, paste, **sed**, seq, sort, tail, tee, uniq, wc

¡y unos cuantos más!

Nota: en rojo programas muy potentes

En el capítulo de hoy...

Programas en Bash:

Scripts

C es un lenguaje compilado

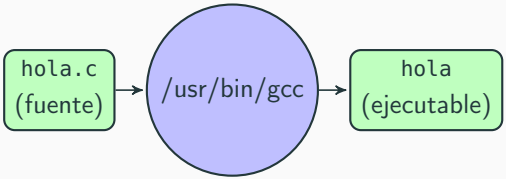
1. Compilación

```
$ gcc -o hola hola.c
```

C es un lenguaje compilado

1. Compilación

```
$ gcc -o hola hola.c
```



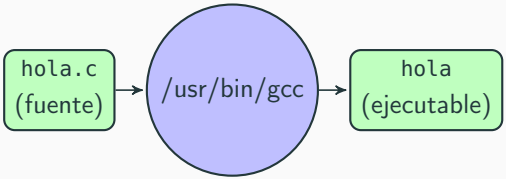
2. Ejecución

```
$ ./hola
```

C es un lenguaje compilado

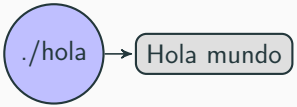
1. Compilación

```
$ gcc -o hola hola.c
```



2. Ejecución

```
$ ./hola
```



Bash es un lenguaje interpretado

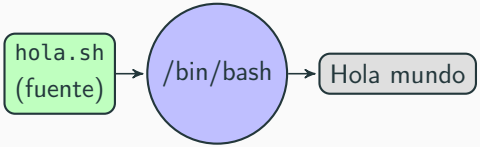
1. Ejecución

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

Bash es un lenguaje interpretado

1. Ejecución

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

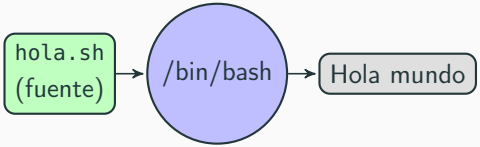


- ¡Y no hay más pasos!
- Vamos a aprender a hacer el `hola.sh`
- Otros lenguajes interpretados:

Bash es un lenguaje interpretado

1. Ejecución

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

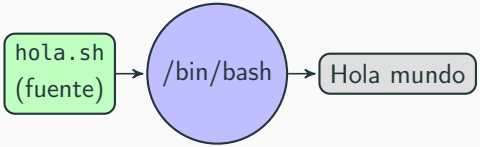


- ¡Y no hay más pasos!
- Vamos a aprender a hacer el `hola.sh`
- Otros lenguajes interpretados: PHP (`php`),

Bash es un lenguaje interpretado

1. Ejecución

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

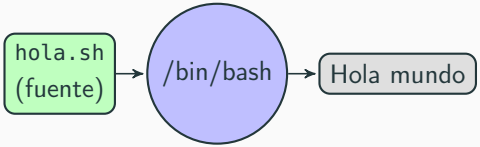


- ¡Y no hay más pasos!
- Vamos a aprender a hacer el `hola.sh`
- Otros lenguajes interpretados: PHP (`php`), Python (`python`),

Bash es un lenguaje interpretado

1. Ejecución

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



- ¡Y no hay más pasos!
- Vamos a aprender a hacer el `hola.sh`
- Otros lenguajes interpretados: PHP (`php`), Python (`python`), Javascript (`node`), etc.

Tu primer script: `hola.sh` i

Crea el fichero `hola.sh` con tu editor favorito:

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

- Bash es un lenguaje muy diferente a Java o a C
- las variables no se declaran, se *expanden* con `$` (ej. `${PATH}`),
- no tiene tipos, ni tiene módulos o paquetes (aunque puedes incluir otro fichero con **`source`**),
- las sentencias son lo que podemos escribir en la línea de comandos

Tu primer script: `hola.sh` ii

- Ejecuta tu *script*

Tu primer script: `hola.sh` ii

Ejecuta tu *script*

```
$ ./hola.sh
bash: ./hola.sh: Permission denied
```

¿Qué ocurre?

```
$ ls -l hola.sh
-rw-rw-r-- 1 angel angel 28 dic 11 10:09 hola.sh
```

Añadir permisos de ejecución con `chmod` y ejecutar

```
$ ./hola.sh
Hola mundo
$ ./hola.sh
```

Primera línea del *script*: `#!/bin/bash`

Convención en UNIX

Cuando se va a ejecutar cualquier fichero de texto,

Primera línea del *script*: `#!/bin/bash`

Convención en UNIX

Cuando se va a ejecutar cualquier fichero de texto, UNIX mira la primera línea y busca `#!`,

Primera línea del *script*: `#!/bin/bash`

Convención en UNIX

Cuando se va a ejecutar cualquier fichero de texto, UNIX mira la primera línea y busca `#!`*, extrae el nombre del programa especificado detrás*

Primera línea del *script*: `#!/bin/bash`

Convención en UNIX

Cuando se va a ejecutar cualquier fichero de texto, UNIX mira la primera línea y busca `#!`*, extrae el nombre del programa especificado detrás (en general suele ser un interprete),*

Primera línea del *script*: `#!/bin/bash`

Convención en UNIX

Cuando se va a ejecutar cualquier fichero de texto, UNIX mira la primera línea y busca `#!`*, extrae el nombre del programa especificado detrás (en general suele ser un interprete), y entonces, coge todo el fichero y se lo pasa a dicho programa por la entrada estándar*

Primera línea del *script*: `#!/bin/bash`

Convención en UNIX

Cuando se va a ejecutar cualquier fichero de texto, UNIX mira la primera línea y busca `#!`*, extrae el nombre del programa especificado detrás (en general suele ser un interprete), y entonces, coge todo el fichero y se lo pasa a dicho programa por la entrada estándar*

¿Qué va a hacer este programa `hola.cat`?

```
#!/bin/cat
Hola mundo
```

¿Dónde está el `main`?

No hay `main`

El *script* empieza ejecutando
la primera línea, luego la segunda, etc.

Argumentos y *exit status* de un *script*

Parámetros:

- `$0`, `$1`, `$2`, `$3`, ... (argv)
- `$#` (argc)
- `$*` (todos)

Exit status

- **`exit`/`return`**

Practiquemos esto y lo que viene con nuestro `hola.sh`

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 $?
```

shift

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.
```

shift

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
```

Script para *descargar* Youtube a MP3 i

- Crear un *script* `yourmp3.sh` con la siguiente funcionalidad:
  - admita como argumentos URLS de videos de Youtube,
  - descargue los videos (usar `youtube-dl`[1]),
  - los pase a MP3 (usar `ffmpeg`),
  - borre los videos,
  - saque por la salida estándar los nombres de los MP3,
  - saque por la salida error cualquier problema encontrado, y
  - de un código de terminación (*exit status*) apropiado

[1] Sólo vamos a usar la funcionalidad de descarga, `youtube-dl` sabe hacer muchas más cosas, entre otras pasar a mp3 ;)

Script para *descargar* Youtube a MP3 ii

Como siempre, empezamos por un *script* que hace "nada"

```
#!/bin/bash
exit 0
```

Script para *descargar* Youtube a MP3 iii

1. Descargar y entender los programas a usar

```
$ curl -L https://yt-dl.org/downloads/latest/youtube-dl -o youtube-dl
$ chmod +x youtube-dl
$ ./youtube-sl
```

2. Ofrecer ayuda al usuario cuando ejecute el *script*, así:

```
$ ./yourmp3.sh
USO: ./yourmp3.sh URL (URL con video de youtube)
```

3. Descargar el video (si algo va mal terminar con *exit status* no 0)
4. Transformar el fichero a mp3 (cuidado porque no es sencillo conocer el nombre del fichero, si algo va mal terminar con *exit status* no 0)

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%.*}` (ver manual de Bash)
- 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 ! ./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
```