

# Sesión 12: En línea

Programación para Sistemas

---

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

## *UNIX, Shell y Scripts*

*Fracisco Rosales*

*Ángel Herranz*

# En el capítulo de hoy...

- ¿Cómo nos comunicamos con nuestros programas?
- Mandatos sencillos
- Un poquito de **piping**
- Variables de entorno
- Comandos útiles

# ¿Cómo nos comunicamos con los procesos?

💬 ¿Cómo ponemos en marcha nuestros programas<sup>1</sup>?:

---

<sup>1</sup>programa en ejecución = proceso

# ¿Cómo nos comunicamos con los procesos?

💬 ¿Cómo ponemos en marcha nuestros programas<sup>1</sup>:

línea de comandos

💬 ¿Cómo podemos **enviar** información a nuestro programa?

---

<sup>1</sup>programa en ejecución = proceso

# ¿Cómo nos comunicamos con los procesos?

💬 ¿Cómo ponemos en marcha nuestros programas<sup>1</sup>?:

línea de comandos

💬 ¿Cómo podemos **enviar** información a nuestro programa?

- Parámetros en la línea de comandos
- Entrada estándar: **stdin** (ficheros en general)

---

<sup>1</sup>programa en ejecución = proceso

# ¿Cómo nos comunicamos con los procesos?

💬 ¿Cómo ponemos en marcha nuestros programas<sup>1</sup>?:

línea de comandos

💬 ¿Cómo podemos **enviar** información a nuestro programa?

- Parámetros en la línea de comandos
- Entrada estándar: **stdin** (ficheros en general)
- **Variables de entorno** ⚠️

---

<sup>1</sup>programa en ejecución = proceso

# ¿Cómo nos comunicamos con los procesos?

💬 ¿Cómo ponemos en marcha nuestros programas<sup>1</sup>?:

línea de comandos

💬 ¿Cómo podemos **enviar** información a nuestro programa?

- Parámetros en la línea de comandos
- Entrada estándar: **stdin** (ficheros en general)
- **Variables de entorno** ⚠️

💬 ¿Cómo podemos **recibir** información de nuestro programa?

---

<sup>1</sup>programa en ejecución = proceso



# ¿Cómo nos comunicamos con los procesos?

💬 ¿Cómo ponemos en marcha nuestros programas<sup>1</sup>?:

línea de comandos

💬 ¿Cómo podemos **enviar** información a nuestro programa?

- Parámetros en la línea de comandos
- Entrada estándar: **stdin** (ficheros en general)
- **Variables de entorno** ⚠️

💬 ¿Cómo podemos **recibir** información de nuestro programa?

- Salida estándar: **stdout** (ficheros en general)
- Salida de error: **stderr**

---

<sup>1</sup>programa en ejecución = proceso

# ¿Cómo nos comunicamos con los procesos?

💬 ¿Cómo ponemos en marcha nuestros programas<sup>1</sup>?:

línea de comandos

💬 ¿Cómo podemos **enviar** información a nuestro programa?

- Parámetros en la línea de comandos
- Entrada estándar: **stdin** (ficheros en general)
- **Variables de entorno** ⚠️

💬 ¿Cómo podemos **recibir** información de nuestro programa?

- Salida estándar: **stdout** (ficheros en general)
- Salida de error: **stderr**
- Estado de terminación: **exit** o **return en main**

---

<sup>1</sup>programa en ejecución = proceso

# Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [...]]]
```

# Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [...]]]
```

```
$ ls -al / ~
```

# Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [...]]]
```

```
$ ls -al / ~
```

```
$ variable=valor mandato [arg1 [arg2 [...]]]
```

# Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [...]]]
```

```
$ ls -al / ~
```

```
$ variable=valor mandato [arg1 [arg2 [...]]]
```

```
$ MAX_OUTPUT=100 ./secuencia -30 0 -3
```

 Sin espacios alrededor de =

# Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [...]]]
```

```
$ ls -al / ~
```

```
$ variable=valor mandato [arg1 [arg2 [...]]]
```

```
$ MAX_OUTPUT=100 ./secuencia -30 0 -3
```

 Sin espacios alrededor de =

Equivalente a

```
$ export MAX_OUTPUT=100
```

```
$ ./secuencia -30 0 -3
```

```
$ unset MAX_OUTPUT
```

# Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [...]]]
```

```
$ ls -al / ~
```

```
$ variable=valor mandato [arg1 [arg2 [...]]]
```

```
$ MAX_OUTPUT=100 ./secuencia -30 0 -3
```



Sin espacios alrededor de =

Equivalente a

Y luego...

```
$ export MAX_OUTPUT=100
```

```
#include <stdlib.h>
```

```
$ ./secuencia -30 0 -3
```

```
...
```

```
$ unset MAX_OUTPUT
```

```
char *limit =
```

```
    getenv("MAX_OUTPUT");
```



# Un uso más interesante del *exit status*

- 📄 Escribir un programa que termine *mal* (mal.c)
- 📄 Ejecutar y comprobar *cómo de mal ha terminado*

# Un uso más interesante del *exit status*

- 📄 Escribir un programa que termine *mal* (mal.c)
- 📄 Ejecutar y comprobar *cómo de mal ha terminado*

```
$ ./mal
```

```
$ echo $?
```

```
255
```

```
$ echo $?
```

```
0
```

- Y esto, ¿Para qué sirve?

# Un uso más interesante del *exit status*

- 📄 Escribir un programa que termine *mal* (mal.c)
- 📄 Ejecutar y comprobar *cómo de mal ha terminado*

```
$ ./mal
$ echo $?
255
$ echo $?
0
```

- Y esto, ¿Para qué sirve?

```
$ if ./mal; then echo BIEN; else echo MAL; fi
MAL
```

- 💬 ¡Atención a la sintaxis del *if*!

# Explorar el sistema de ficheros i

- En UNIX no hay *unidades de disco* (C:, D:, ...)
- Todo empieza en el directorio raíz: /

# Explorar el sistema de ficheros i

- En UNIX no hay *unidades de disco* (C:, D:, ...)
- Todo empieza en el directorio raíz: /



Nombrado de ficheros y directorios

- **Absoluto**: el nombre empieza por /
- **Relativo**: el nombre **no** empieza por / y se convierte en absoluto concatenándolo al *working directory*.
- Elementos especiales: ~, ., ...

## Explorar el sistema de ficheros ii

¡Busca la **equivalencia** con las **carpetas**!

- Empezar en / (**cd** /)
- Ejecutar `ls -al`
- Entrar en cada directorio y ejecutar `ls -al`
- Moverse a ~ (**cd** ~)
- Ejecutar `ls -al`
- Entrar en cada directorio y ejecutar `ls -al`

## Explorar el sistema de ficheros ii

¡Busca la **equivalencia** con las **carpetas**!

- Empezar en / (**cd** /)
- Ejecutar `ls -al`
- Entrar en cada directorio y ejecutar `ls -al`
- Moverse a ~ (**cd** ~)
- Ejecutar `ls -al`
- Entrar en cada directorio y ejecutar `ls -al`

⌚ 1'  <http://refspecs.linuxfoundation.org/fhs.shtml>

## Explorar el sistema de ficheros iii

 ¿Dónde están los discos?



## Explorar el sistema de ficheros iii

### ❓ ¿Dónde están los discos?

- Los discos están en `/dev` (ej. `/dev/sda`, `/dev/sdb1`, etc.)
- Pero **no se pueden usar**
- Primero hay que **montarlos**:

```
$ mount /dev/sda1 /home
```

```
$
```

- Y entonces el directorio `/home` **es realmente** la partición 1 del disco `/dev/sda`

## Explorar el sistema de ficheros iii

❓ ¿Dónde están los discos?

- Los discos están en `/dev` (ej. `/dev/sda`, `/dev/sdb1`, etc.)
- Pero **no se pueden usar**
- Primero hay que **montarlos**:

```
$ mount /dev/sda1 /home
```

```
$
```

- Y entonces el directorio `/home` **es realmente** la partición 1 del disco `/dev/sda`

🔍 **cat** `/etc/fstab`

# ¿Qué hay detrás de los mandatos conocidos?

- ¿Qué hace `ls`?

`man ls`

# ¿Qué hay detrás de los mandatos conocidos?

- ¿Qué hace `ls`?

`man ls`

- ¿Qué hace `cd`?

`man cd`

# ¿Qué hay detrás de los mandatos conocidos?

- ¿Qué hace `ls`?

`man ls`

- ¿Qué hace `cd`?

`man cd`



¿Por qué no hay manual de **`cd`**?


# ¿Qué hay detrás de los mandatos conocidos?

- ¿Qué hace `ls`?

`man ls`

- ¿Qué hace `cd`?

`man cd`

 ¿Por qué no hay manual de `cd`?

- `cd` es un *built in command* de Bash:

`man ls`

- Los mandatos de Bash pueden ser  
programas  
o  
built in commands
- Explorar el manual: `man bash` y `man PROGRAMA`

# ¿Dónde están los programas?

- Los programas están en el sistema de ficheros

 `which ls`

 ¿Y si hay dos programas con el mismo nombre?

Variable de entorno **PATH**

---


<sup>2</sup>Separadas por el caracter ":" en Unix



# ¿Dónde están los programas?


- Los programas están en el sistema de ficheros

 **which** **ls**

 ¿Y si hay dos programas con el mismo nombre?

## Variable de entorno **PATH**

- Un **path** es una lista de directorios<sup>2</sup>

 Mira y cambia el **PATH**

```
$ echo $PATH
```

```
$ ls
```

```
$ which ls
```

```
$ PATH=
```

```
$ echo $PATH
```

```
$ ls
```

```
$ which ls
```

---

<sup>2</sup>Separadas por el caracter ":" en Unix

# Sobre las variables de entorno i

- Los programas usan variables de entorno, algunas son **comunes**:

PATH, PS1, USER, SHELL, PWD, HOSTNAME,  
LANG, EDITOR, etc.

- Pero cada programador puede **definir** las suyas:

JAVA\_HOME, CLASSPATH

- Todo lo que se haga con ellas **se pierde** entre sesiones

# Sobre las variables de entorno

- Se establecen al arrancar Bash
- `/etc/profile` *The systemwide initialization file, executed for login shells*
- `/etc/bash.bashrc` *The systemwide per-interactive-shell startup file*
- `~/.bash_profile` *The personal initialization file, executed for login shells*
- `~/.bashrc` *The individual per-interactive-shell startup file*
- `~/.bash_logout` *The individual login shell cleanup file, executed when a login shell exits*
- `/etc/bash.bash.logout` *The systemwide login shell cleanup file, executed when a login shell exits*

# echo

🕒 1' 🔍 man **echo**

# echo

🕒 1' 🔍 man **echo**

💻 Ejecutar y *diseccionar*

```
$ echo Fíjate en los    espacios
```

```
$ echo "En un lugar de la mancha..." > quijote.txt
```

# echo

🕒 1' 🔍 man **echo**

💻 Ejecutar y *diseccionar*

\$ **echo** Fíjate en los      espacios

\$ **echo** "En un lugar de la mancha..." > quijote.txt

🏠 ¿Te atreves a programar echo en C?

🕒 1' 🔍 man **echo**

💻 Ejecutar y *diseccionar*

\$ **echo** Fíjate en los espacios

\$ **echo** "En un lugar de la mancha..." > quijote.txt

🏠 ¿Te atreves a programar echo en C?

💬 ¿Qué es echo? ¿Dónde está? ¿Por qué aparece en negrita en estas transparencias? ¿Has probado which? ¿Has mirado en man bash?

# cat i

- ¿Qué hace cat?

🕒 1' 🔍 man **cat**



# cat i

- ¿Qué hace cat?

🕒 1' 🔍 man **cat**

💻 Ejecutar y *diseccionar*

```
$ cat quijote.txt
```

```
$ cat
```

💬 ¿Qué ocurre?

- ¿Qué hace cat?

🕒 1' 🔍 man **cat**

💻 Ejecutar y *diseccionar*

```
$ cat quijote.txt
```

```
$ cat
```

💬 ¿Qué ocurre? Prueba a escribir

Los animales son felices mientras  
tengan salud y suficiente comida.

Ctrl-d

## Ejecutar y *disecccionar*

```
$ cat < quijote.txt
```

```
$ cat > la_conquista.txt
```

Los animales son felices mientras  
tengan salud y suficiente comida.

Ctrl-d

```
$ cat quijote.txt la_conquista.txt
```

```
$ cat quijote.txt la_conquista.txt > dos_libros.txt
```

⌚ 2' 🔍 ¿Qué es read?

② 2'  ¿Qué es read?

 Ejecutar y explorar:

```
$ read -p "¿Qué edad tienes?" EDAD
```

 La respuesta queda en la variable de entorno EDAD:

```
$ echo $EDAD
```

- *¿Crawling?*

---

<sup>3</sup>Instalado en triqui, instalable en Ubuntu con `apt-get install curl`

# Crawling i

- ¿Crawling?
- Usaremos el programar **cURL**<sup>3</sup>: `man curl`

```
$ curl http://www.fi.upm.es
```

```
$ curl http://www.fi.upm.es | grep href
```

---

<sup>3</sup>Instalado en triqui, instalable en Ubuntu con `apt-get install curl`

# Crawling i

- ¿Crawling?
- Usaremos el programar **cURL**<sup>3</sup>: `man curl`

```
$ curl http://www.fi.upm.es
```

```
$ curl http://www.fi.upm.es | grep href
```

**Q** ¿Qué es grep?

---

<sup>3</sup>Instalado en triqui, instalable en Ubuntu con `apt-get install curl`



# Crawling i

- ¿Crawling?
- Usaremos el programar **cURL**<sup>3</sup>: `man curl`

```
$ curl http://www.fi.upm.es
```

```
$ curl http://www.fi.upm.es | grep href
```

## Q ¿Qué es grep?

- *Crawling*:

```
$ curl -s http://www.fi.upm.es | grep -Po '(?<=href=")[^"]*(?=")'
```

---

<sup>3</sup>Instalado en triqui, instalable en Ubuntu con `apt-get install curl`

- *Almost magic:*

```
$ curl -s http://www.fi.upm.es |  
  grep -Po '(?<=href=")[^"]*(?=")' |  
  sort |  
  uniq
```

 man sort y man uniq

# Algo más útil: MP3 de youtube i

- <https://www.youtube.com/watch?v=ukKQw578Lm8>
- Necesitaremos dos programas
- *youtube-dl* - *download videos from youtube.com or other video platforms*
- *ffmpeg* - *ffmpeg video converter*
- Necesitaremos la versión más actual de *youtube-dl*:

```
$ cd tmp
$ curl -L https://yt-dl.org/downloads/latest/youtube-dl -o youtube-dl
$ chmod +x youtube-dl
$ ls -l youtube-dl
```

# Algo más útil: MP3 de youtube ii

- Descargamos el video

```
$ ./youtube-dl https://www.youtube.com/watch?v=ukKQw578Lm8
```

- Eso ha generado el fichero

```
TheLogicalSongporRogerHodgson-Letra.-ukKQw578Lm8.webm
```

- Guardamos el nombre sin extensión en una variable:

```
$ VIDEO="The Logical Song por Roger Hodgson -Letra.-ukKQw578Lm8"
```

- Lo pasamos a MP3

```
$ ffmpeg -i "$VIDEO.webm" -vn -ab 128k -ar 44100 -y "$VIDEO.mp3"
```

# ¿Por qué tanto entusiasmo?

Se puede  
automatizar

# ¿Por qué tanto entusiasmo?

Se puede  
automatizar  
en  
Scripts