Deusto
Facultad de Ingeniería
Ingeniaritza Fakultatea

Nombre 1: Asier Marin
Nombre 2: Aitor Ubierna

Laboratorio 1

Parte 1: Comandos de la Shell

Primero pasos con la terminal. Responde a las preguntas en el recuadro.

Pulsar cualquier conjunto de teclas

```
$ klñsdjlksajlkfhsdklhf sdklafhklsdh
```

¿Qué sucede?

Command not found.

¿Dónde estamos en el árbol de directorios y ficheros?

```
$ pwd
```

¿Cómo me cambio a otra parte en el árbol de directorios y ficheros?

```
$ cd "seguido del directorio al que quiero acceder"
```

Ejemplo:

```
$ cd / (todo empieza con la barra "/"→ ruta absoluta)
```

¿Cómo sé dónde estoy ahora?
¿Y si quiero volver dónde estaba?

Con [pwd] sabemos donde estamos. Con cd podemos acceder al directorio siguiente. Con [cd..] volvemos hacia detrás dentro del árbol de direcciones.

¿Cómo puedo saber los ficheros, directorios o enlaces que hay en el directorio dónde estoy?

```
$ ls
```

Vale... ¿y si quiero saber cuánto ocupan, o cuándo se crearon?

```
$ ls -l
```

¿Qué otra información encuentro cuando hago ls -l?

Fechas de archivos modificados por última vez. También aparece la hora y enlaces simbólicos [lrwxrwxrwx] de los ficheros.

```
-rw------- 1 bshotts bshotts 576 Apr 17 1998 weather.txt
drwxr-xr-x 6 bshotts bshotts 1024 Oct 9 1999 web_page
-rw-rw-r-- 1 bshotts bshotts 276480 Feb 11 20:41 web_site.tar
-rw------- 1 bshotts bshotts 5743 Dec 16 1998 xmas_file.txt
---------- ------- ------- -------- ------------ -------------
| | | | | |
| | | | | File Name
| | | | |
| | | | +--- Modification Time
| | | |
| | | +------------- Size (in bytes)
| | |
| | +----------------------- Group
| |
| +-------------------------------- Owner
|
+---------------------------------------------- File Permissions
```

File Name
The name of the file or directory.
Modification Time
The last time the file was modified. If the last modification occurred more than six months in the past, the date and year are displayed. Otherwise, the time of day is shown.
Size
The size of the file in bytes.
Group
The name of the group that has file permissions in addition to the file's owner.
Owner
The name of the user who owns the file.
File Permissions
A representation of the file's access permissions. The first character is the type of file. A "-" indicates a regular (ordinary) file. A "d" indicates a directory. The second set of three characters represent the read, write, and execution rights of the file's owner. The next three represent the rights of the file's group, and the final three represent the rights granted to everybody else.

¿cómo puedo saber si hay más?

```
$ man
```

Hay que decirle a man sobre qué comando queremos más información.

```
$ man man
```

Parece que da ejemplos...¿por qué no probar con los comandos que ya hemos

aprendido?

```
$ man pwd
$ man ls
$ man cd
```

¿Qué otros parámetros acepta el comando ls?

Por ejemplo:

-a, --all
do not ignore entries starting with .

-A, --almost-all
do not list implied . and ..

--author
with -l, print the author of each file

-b, --escape
print C-style escapes for nongraphic characters

--block-size=SIZE
with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below

-B, --ignore-backups
do not list implied entries ending with ~

-c with -lt: sort by, and show, ctime (time of last modification of file status information); with -l: show ctime and sort by name; otherwise: sort by ctime, newest first

-C list entries by columns

--color[=WHEN]
colorize the output; WHEN can be 'always' (default if omitted), 'auto', or 'never'; more info below

¿Y si te digo que puedes concatenar parámetros a los comandos…?
Prueba este:

```
$ ls -la
```

Aparecen dos directorios “raros” . y ..

¿Qué son?

. hace alusión al directorio presente, donde se encuentra.
.. hace alusión al directorio anterior, que se puede mover hacia atrás en la rama.

Deusto
Facultad de Ingeniería
Ingeniaritza Fakultatea

¿Sabes que puedes moverte a esos directorios?

```
$ cd .
$ cd ..
```

Otro "raro" más ~

```
$ cd ~
```

¿Qué pasa con este último?

Te lleva al directorio predeterminado.

Vamos a crear un fichero

```
$ touch miFichero.txt
$ ls -l
```

¿Qué tamaño tiene? ¿A quién pertenece? ¿Qué permisos tiene?
¿Cuándo se creó?

Primero se obtiene permisos con sudo su, y ya se puede crear:
-rw-r--r-- 1 root root 0 Sep 18 16:31 miFichero.txt

Introducir texto

Sed valientes :/

```
$ nano miFichero.txt
$ vim miFichero.txt
$ emacs miFichero.txt
```

Recomiendo usar **nano** por defecto

Mostrar texto en la consola shell

```
$ echo "hola mundo"
```

¿Si quiero meter el "hola mundo" en el fichero ficheroQueVoyACrear.txt?

```
$ echo "hola mundo" > ficheroQueVoyACrear.txt
```

Dos formas de ver el contenido del fichero.

```
$ cat miFichero.txt
$ less miFichero.txt
```

¿Qué diferencias hay entre esas dos formas de ver el contenido?

Uno te muestra el contenido en la misma consola. El otro te visualiza en otra pantalla o editor.

¿Si quiero meter otra línea en mi fichero ficheroQueVoyACrear.txt?

```
$ q
```

¿Qué tipo de fichero es ficheroQueVoyACrear.txt?

```
$ file miFichero.txt
```

¿Cómo sabe file el tipo de fichero que se pasa por parámetro ?

```
$ man file
```

Comandos típicos:

Copiar

```
$ cp miFichero.txt miFichero_copiado.txt
```

Mover

```
$ mv ficheroQueVoyACrear.txt ./../ficheroQueVoyACrear_movido.txt
```

Mover también se usa para renombrar un fichero o un directorio.

```
$ mv ficheroQueVoyACrear.txt otroNombreParaElFichero.txt
```

Borrar

```
$ rm ficheroQueVoyACrear.txt
$ rm ./../ficheroQueVoyACrear_movido.txt
```

Crear un directorio

```
$ mkdir miDirectorioNuevo
$ ls -l
```

¿Y si lo quiero borrar? Pista: consultar el manual.

rmdir /miDirectorioNuevo.

En la shell se pueden concatenar varios comandos

Por ejemplo, quiero mostrar el contenido de un fichero y ordenar sus líneas.

Mostrar el contenido:

```
$ less
$ cat
```

Ordenar líneas:

```
$ sort
```

Unimos los dos

```
$ cat ficheroConVariasLineas.txt | sort
```

¿Qué otras concatenaciones se te ocurren

¿Te has preguntado por qué puedes ejecutar comandos con solo poner su nombre?

En realidad esos comandos son programas que hacen llamadas de sistema (Kernel)...pero

¿dónde están esos programas?

Pista:

```
$ echo "$PATH"
```

Están todos en el directorio usr/.

Redireccionamientos

Una redirección consiste en trasladar la información de un tipo a otro, por ejemplo de la salida estándar a la entrada estándar o del error estándar a la salida estándar. Esto lo logramos usando el símbolo >.

Por ejemplo, para redireccionar la salida de un comando y volcarla a un archivo bastaría con ejecutar:

```
$ ls -la ~ > archivo.txt
```

Sin embargo, cada vez que ejecutemos ese comando el contenido de archivo.txt será reemplazado por la salida del comando ls. Si queremos agregar la salida del comando al archivo, en lugar de reemplazarla, entonces ejecutamos:

```
$ ls -la ~ >> archivo.txt
```

Tuberias

También podemos redireccionar la salida estándar del comando *cat* y pasarla como entrada estándar del comando *wc* para contar las líneas y palabras de un archivo:

```
$ cat archivo.txt | wc
```

Ejercicio: Lista los archivos terminados en .txt (pista: usa el comando grep)

ls *.txt.

Parte 2: Shell Scripting

- Manual de Shell Scripting en castellano:
  http://trajano.us.es/~fjfj/shell/shellscript.htm

El intérprete de comandos o shell es un programa que permite a los usuarios interactuar con el sistema, procesando las órdenes que se le indican. Los comandos invocables desde el Shell pueden clasificarse en internos (corresponden en realidad a órdenes interpretadas por el propio shell) y externos (corresponden a ficheros ejecutables externos al shell).

Además de comandos, los shells ofrecen otros elementos para mejorar su funcionalidad, tales como variables, funciones o estructuras de control. El conjunto de comandos internos y elementos disponibles, así como su sintaxis, dependerá del shell concreto empleado.

Además de utilizar el shell desde la línea de comandos (basada en el prompt como la indicación del shell para anunciar que espera una orden del usuario), puede emplearse para la interpretación de shellscripts. Un shell-script o "guión de órdenes" es un fichero de texto que contiene un conjunto de comandos y órdenes interpretables por el shell.

En su forma más básica, un shell-script puede ser un simple fichero de texto que contenga uno o varios comandos. Para ayudar a la identificación del contenido a partir del nombre del archivo, es habitual que los shell scripts tengan la extensión ".sh", por lo que seguiremos este criterio a lo largo de la práctica (pero recuerde que es algo meramente informativo y opcional).

Además de comandos, los shell-scripts pueden contener otros elementos, aportados por el shell para mejorar la funcionalidad de los scripts. De forma resumida, la estructura básica de un shell-script es la siguiente:

Un shell script no deja de ser una secuencia de llamadas al sistema con un fin determinado.

Además de comandos, los shell-scripts pueden contener otros elementos, aportados por el shell para mejorar la funcionalidad de los scripts. De forma resumida, la estructura básica de un shell-script es la siguiente:

script_ejemplo.sh

```
#!/bin/bash                                  <-- Shebang
# Esto no se interpreta                      <-- Comentarios
echo Hola                                    <-- Contenido
ps w
echo "Proceso lee el script: $$"
```

Actividades guiadas

1. Lo primero que tenemos que hacer para crear nuestro primer shell script es abrir un editor de textos.

2. Lo segundo comenzar todos los scripts con la siguiente línea de texto:
a. #!/bin/bash → Con esto le decimos al script que se tiene que ejecutar a través de la bash

3. Escribir nuestro conjunto de llamadas a la Shell

4. Guardar el fichero con un nombre que describa el objeto del script. No dar ningún tipo de extensión al fichero.

5. Cambiar los permisos para convertir el fichero de solo lectura y escritura a ejecutable

```
$ chmod u+x nombreDeFicheroAEjecutar
```

6. Ejecutar el fichero

```
$ bash ./nombreDeFicheroAEjecutar
```

Ejemplos de shell scripts:

```
#!/bin/bash
echo "mi primer script"
ls -l
echo "se acaba el script"
```

```
#!/bin/bash
echo "Se va a mostrar el parámetro que pongas a continuación del
nombre del script"
echo $1
```

```
#!/bin/bash
read -p "Your name? " name
if [ $name = $USER ]; then
echo "Hello, me."
else
echo "Hello, $name."
fi
```

```
#!/bin/bash
prefix=ICO
for i in `ls -1 *.txt`;
do
echo "Now working on $i"
newName = $prefix-$i
mv $i $newName;
echo “New name is $newName”
done
```

Entrada standard

```
echo " Introduzca una cadena "
read CAD
echo " Cadena introducida: $CAD "
```

Bucle for

```
#!/bin/sh

for i in 1 2 3; do
    echo "Iteracion: $i"
done > ficherosalida
```

While

```
#!/bin/sh

CONTADOR=0
while [ $CONTADOR – lt 3 ]; do
    echo "Contador: $CONTADOR "
    CONTADOR=$(($CONTADOR+1))
done
```

Ejercicios

Incluir el código del fichero, así como el resultado que se os muestra en la terminar al ejecutar el mismo.

1. Ejecuta el siguiente script y comenta su funcionalidad.

```
#!/bin/bash

for file in `ls /`; do
  echo "Fichero: $file"
done
```

Muestra todos los ficheros y directorios de la ubicación actual. Ha utilizado el bucle for para recorrer todos los elementos utilizando el comando[ls /]. La barra [/] le indica que quiere mostrar todos los elementos de ese mismo lugar.

Para finalizar el bucle se utiliza el comando done.

2. Escribe un script con la siguiente funcionalidad
    - Reciba un número n por parámetro.
    - El programa tendrá que sumar todos los números entre 1 y n. Posteriormente mostrara el resultado de la suma por pantalla

```
#!/bin/bash
echo -n "ENTER INTEGER VALUE [0-9]: "
read stringValue
echo "I'M GOING TO VERIFY IF IS A INTEGER...."
if [[ ! $stringValue =~ ^[0-9]+$ ]] ; then
   echo "No good MEN :("
   exit
fi
((n = $stringValue))
((total = 0))
for (( i = 1; i <= $n; i++ ))
do
echo "i = " $i
(( total = $total + $i ))
done
echo "RESULT" $total
```

3. Programa que lea palabras y las guarde en un fichero llamado *fichero3.txt*, hasta que se escriba ":q"

```
#!/bin/bash

touch fichero3.txt
echo -n "INTRODUCE CADENA: "
read cadena
while [ !"$cadena" = ":q" ]; do
      read $cadena
      echo "Copiando  $cadena "
      $cadena ~ >> fichero3.txt
done
```