ParmigianoReg
www.wuolah.com/student/ParmigianoReg



# resumenFSPractica.pdf

*Resumen de comandos de Bash*

**1° Fundamentos del Software**

**Grado en Ingeniería Informática**

**Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**
**Universidad de Granada**

# 1 Metacarácteres de archivo

| | |
|---|---|
| `?` | Representa cualquier carácter simple en la posición en la que se indique |
| `*` | Representa cualquier secuencia de cero o más carácteres |
| `[]` | Designan un carácter o rango de caracteres que representan un carácter simple a través de una lista de caracteres o mediante un rango, en cuyo caso, mostramos el primer y último carácter del rango separados por un guión "-". |
| `{}` | Sustituyen conjuntos de palabras separadas por comas que comparten partes comunes. |
| `~` | Se usa para abreviar el camino absoluto (path) del directorio HOME. |

# 2 Metacarácteres de redirección

| | |
|---|---|
| `< [archivo]` | Redirecciona la entrada de una orden para que la obtenga del archivo nombre. |
| `> [nombre]` | Redirige la salida de una orden para que la escriba en el archivo nombre. Si dicho archivo ya existe, lo sobreescribe. |
| `&> [nombre]` | La salida estándar se combina con la salida de error estándar y ambas se escriben en el archivo nombre. |
| `>> [nombre]` | Funciona igual que el metacarácter "`>`" pero añade la salida estándar al final del contenido del archivo nombre. |
| `&>> [nombre]` | Igual que el metacarácter "`&>`", pero añadiendo las dos salidas combinadas al final del archivo nombre. |
| `2> [nombre]` | Redirige la salida de error estándar a un archivo. |
| `\|` | Crea un cauce entre dos órdenes. La salida de una de ellas se utiliza como entrada de la otra. |
| `\|&` | Crea un cauce entre dos órdenes utilizando las dos salidas (estándar y error) de una de ellas como entrada de la otra. |

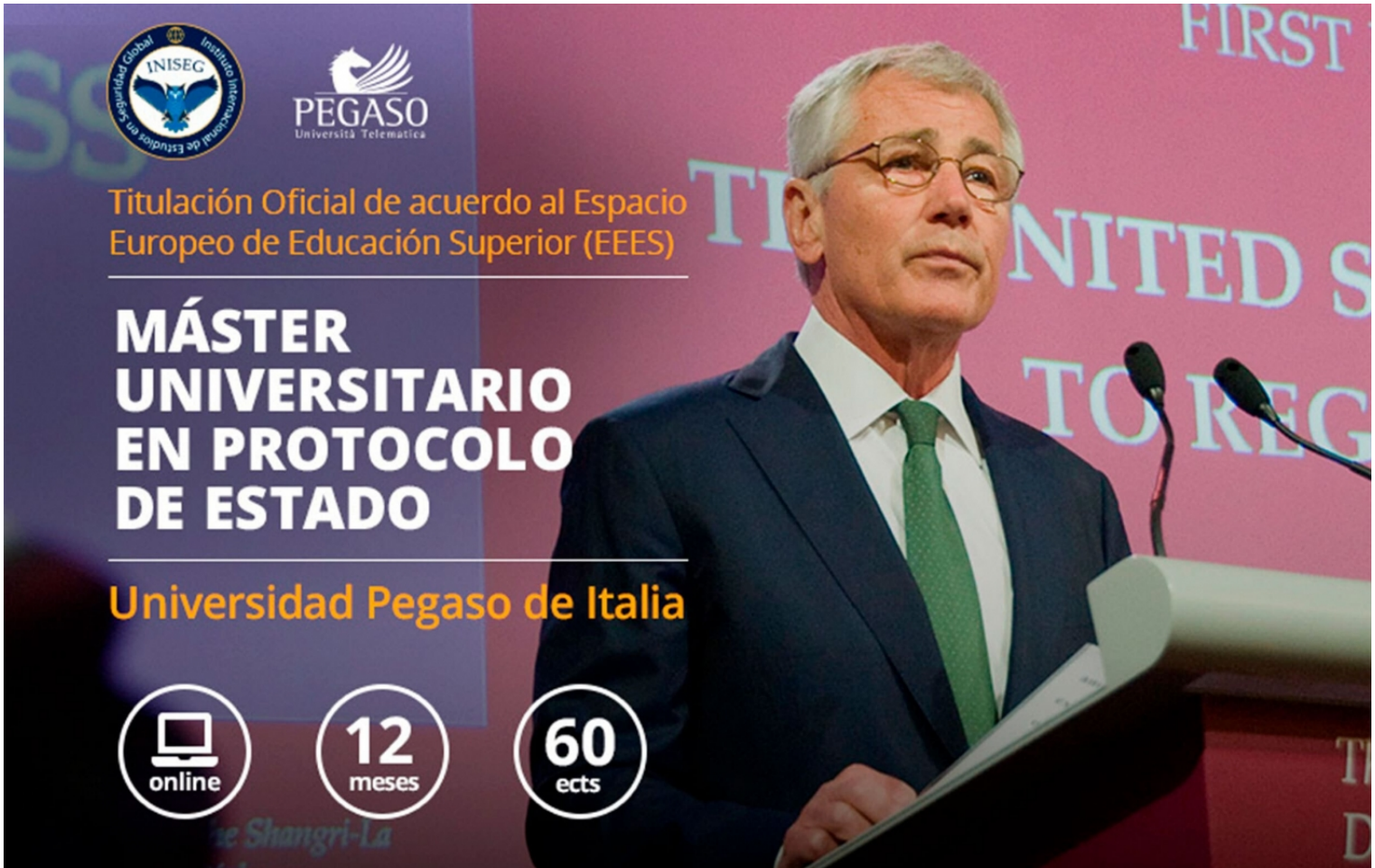
# 3 Comandos básicos

Como siempre, todos estos comandos poseen miles de opciones, siempre se recomienda o buscar en el manual o una búsqueda rápida en Google.

## 3.1 alias

Permite condensar uno o varios comandos en uno solo. Es posible utilizar el ";" para separar los comandos a utilizar.

```
alias [nombre]='[comandos]'
```



Titulación Oficial de acuerdo al Espacio Europeo de Educación Superior (EEES)

# MÁSTER UNIVERSITARIO EN PROTOCOLO DE ESTADO

## Universidad Pegaso de Italia

online | 12 meses | 60 ects

## Salidas Profesionales, Académicas y de Investigación:

- Investigación en protocolo
- Doctorado en protocolo
- Profesorado de protocolo
- Dirección y planificación estratégica de imagen.
- Dirección de protocolo y ceremonial en instituciones públicas y privadas
- Director de gabinete de protocolo.
- Director de agencias y empresas de protocolo y organización de eventos.
- Técnico en protocolo en instituciones públicas y privadas.
- Técnico en empresas y agencias de organización de actos y/o eventos.
- Consultor/asesor de ceremonial, protocolo y organización de actos.
- Asistente Personal (Personal Assitant)
- Organizador de congresos y reuniones.
- Atención a personal VIP.
- Entre otras

**Próxima Convocatoria:**

OCTUBRE 2020

**Dirigido a:**

- Profesionales del Derecho, Ciencias Politicas, Sociales, Económicas, Empresariales y Sociales
- Profesionales de la Ciencias de Información
- Fuerzas Armadas, eclesiásticos, Empresas, Universidades, Organizaciones Internacionales, Diplomacia, entre otras.
- Interesados en el mundo del protocolo estatal e internacional
- Fuerzas y Cuerpos de Seguridad del Estado y Estudiantes Universitarios en Protocolo, Turismo, Humanidades.
- Técnicos de la Administración, funcionarios

**Colaboradores**

EIP Escuela Internacional de Protocolo

EUFB Escola Universitària Formatic Barcelona

www.iniseg.es

+(34) 912 141 926
informacion@iniseg.es

Calle Emilia Pardo Bazán 1,
28903 Getafe, Madrid

```
alias python='python3'
```

Para quitar un alias se utiliza unalias.

```
unalias [nombre]
unalias python
```

## 3.2 man

Muestra el manual del comando proporcionado. No todos los comandos poseen un manual, o bien utilizan otra instrucción.

```
man [comando]
man ls
```

## 3.3 ls

Lista los contenidos del directorio que se le proporciona al comando.

| | |
|---|---|
| `-C` | Lista los directorios en formato multicolumna. |
| `-l` | Formato largo |
| `-r` | Lista en orden inverso |
| `-R` | Lista subdirectorios recursivamente, además del directorio actual. |
| `-t` | Lista de acuerdo con la fecha de modificación de los archivos |

```
ls [directorio] [opciones]
ls $HOME
```

## 3.4 cd

Cambia el directorio de trabajo de la terminal.

```
cd [directorio]
cd Destkop/
```

## 3.5 pwd

Imprime el camino absoluto al directorio actual.

```
pwd
```

## 3.6 mkdir

Crea un directorio a partir del nombre que se le proporciona al comando.

```
mkdir [nombre]
mkdir foo
```

### 3.7 rmdir

Elimina un directorio con el nombre proporcionado si el directorio está vacío.

```
rmdir [nombre]
rmdir bar
```

### 3.8 cat

Muestra el contenido de un archivo de texto plano.

```
cat [archivo(s)]
cat foobar.txt
```

### 3.9 cp

Copia el [archivo1] en [archivo2], si [archivo2] no existe, se crea.

```
cp [archivo1] [archivo2]
```

### 3.10 mv

Renombra archivos (el archivo fuente puede ser archivo o directorio, al igual que el destino) y puede mover de lugar un archivo o directorio

```
mv [fuente] [destino]
```

### 3.11 file

Muestra el tipo de archivo dado como argumento

```
file [archivo]
```

### 3.12 more

Visualiza un archivo fraccionándolo una pantalla cada vez (existen otros paginadores como page, pg, etc.). Antes de usar esta orden es conveniente usar la orden file para comprobar que se trata de un archivo ASCII.

```
more [archivo]
```

### 3.13 rm

Borra archivos y directorios con contenido

```
rm [directorio_o_archivos]
```

### 3.14 touch

Si existen los archivos dados como argumentos se modifican su fecha y hora. En caso contrario, se crean con la fecha actual del sistema.

```
touch [archivo]
```

### 3.15 tail

Muestra la parte final del contenido de un archivo dado como argumento. Por defecto muestra 10 líneas.

```
tail [archivo]
tail -n [num] [archivo]
```

### 3.16 head

Muestra la parte inicial del contenido de un archivo dado como argumento. Por defecto muestra 10 líneas.

```
head [archivo]
head -n [num] [archivo]
```

### 3.17 sort

Ordena

```
sort [archivo]
```

## 4 Comandos avanzados

### 4.1 Imprimir a consola: printf

```
printf "[texto]" $variable1 $variable2 ...
printf "Esta computadora se llama %s y el usuario
   es %s \n" $HOSTNAME $USER
```

- Secuencias de escape:

| | |
|---|---|
| \b | Espacio atrás |
| \n | Nueva línea |
| \t | Tabulador |
| \’ | Carácter de comilla simple |
| \\ | Barra invertida |

- Códigos de formato:

| | |
|---|---|
| %d | Número entero. |
| %id | Número entero, i es un número que permite justificar hacia la derecha o izquiera. |
| %f | Número real. |
| %i.jf | Número real, dónde i es un número indica el numero de justificación y j la cantidad de decimales. |
| %s | Cadena de texto. |
| %x | Representa el número en hexadecimal. |
| %o | Representa el número en octal. |

## 4.2 Búsqueda de archivos: find

```
find [directorio] [opciones-de-busqueda]
find . -size 10
find . -type f
```

- Entre las opciones de búsqueda se encuentra:

| | |
|---|---|
| -name "[nombre]" | Busca un archivo o archivos con el nombre especificado. Pueden usarse metacarácteres. |
| -atime [+/-/ ][n] | Si el archivo fue accedido hace más, menos o exactamente n días. Existen variantes para minutos, horas, etc. |
| -type [f/d] | Tipo de archivo, f si es archivo y d si es carpeta. |
| -size [+/-/ ][n] | Si el archivo tiene un tamaño mayor, menor o igual a n. La unidad por defecto sin especificar es de 512 bytes. |
| -user [nombre] | Si el archivo le pertenece a un usuario en particular. |

## 4.3 Búsqueda dentro de uno o más archivos: grep

```
grep [opciones] [patron] [archivo(s)]
grep mundo * # Busca la palabra "mundo" en todos
   los archivos del directorio actual
```

- Entre las opciones de búsqueda se encuentra:

| | |
|---|---|
| -x | Localiza líneas que coincidan totalmente, desde el principio hasta el final de línea, con el patrón especificado. |
| -v | Selecciona todas las líneas que no contengan el patrón especificado. |
| -c | Produce solamente un recuento de las líneas coincidentes. |
| -i | Ignora las distinciones entre mayúsculas y minúsculas. |

### 4.4 Consulta de archivos: test

```
test [expresion] [archivo]
```

- Entre las expresiones se encuentra:

| | |
|---|---|
| `-b [archivo]` | Archivo existe y es un dispositivo de bloques. |
| `-c [archivo]` | Archivo existe y es un dispositivo de caracteres. |
| `-d [archivo]` | Archivo existe y es un directorio. |
| `-e [archivo]` | Archivo existe. |
| `-f [archivo]` | Archivo existe y es un archivo plano o regular. |
| `-G [archivo]` | Archivo existe y es propiedad del mismo grupo del usuario. |
| `-h [archivo]` | Archivo existe y es enlace simbólico. |
| `-O [archivo]` | Archivo existe y es propiedad del usuario. |
| `-r [archivo]` | Archivo existe y el usuario tiene permiso de lectura sobre él. |
| `-s [archivo]` | Archivo existe y no es vacío. |
| `-w [archivo]` | Archivo existe y el usuario tiene permiso de escritura sobre él. |
| `-x [archivo]` | Archivo existe y el usuario tiene permiso de ejecución sobre él. |
| `[f1] -nt [f2]` | Archivo f1 es más reciente que archivo f2, según la fecha de modificación, o si f1 existe y f2 no. |
| `[f1] -ot [f2]` | Archivo f1 es más antiguo que archivo f2, según la fecha de modificación, o si f2 existe y f1 no. |

## 5 Modificación de permisos

Permite modificar los permisos del archivo o archivos proporcionados.

| | |
|---|---|
| `u` | Propietario |
| `g` | Grupo |
| `o` | Resto de usuarios |
| `a` | Todos |

| | |
|---|---|
| `r` | Escritura |
| `w` | Lectura |
| `x` | Ejecución |

```
chmod [grupo_usuarios]+/-[permisos] [archivo(s)]
$ chmod g+w ej2
$ chmod u+x,g-w ej2
$ chmod g+x ej*
```

## 6 Variables

### 6.1 Crear variable

- Asignación de un número

```
var=1
```

- Asignación de una cadena

```
var="texto"
var='texto'
```

- Asignación de otra variable

```
varNueva=$varVieja
```

- Asignación de la ejecución de un comando

```
var=`[comando]`
var=`ls` # Ejemplo: Almacenara lo que muestre
    ls
```

- Asignación por una expresión aritmética simple

```
var=`expr $numero + 1` # "numero" es otra
    variable
```

- Asignación de expresiones más complejas

```
var=$(( 2*4+8 ))
var=$[ 2*4+8 ]
let var=2*4+8
```

## 6.2 Borrar variable

- 
```
unset var
```

## 6.3 Exportar variable

- 
```
export var
```

## 6.4 Vectores

- Creación de un vector

```
vector=(item0 item1 ... itemN)
colores=(amarillo azul rojo)
```

- Acceder a un elemento

```
${vector[i]}
var=${colores[1]} # Azul
```

- Imprimir todos los contenidos de un vector

```
echo ${vector[*]}
```

# 7 Guiones

- Al inicio se debe colocar `!#/bin/bash`
- Variables parametrizadas:

| | |
|---|---|
| `$#` | Número de parámetros. |
| `$0` | Nombre del guión. |
| `$1...$9` | Argumentos del guión, se pueden referenciar directamente. |
| `${n>9}` | Cuando hay más de 9 argumentos, de 10 en adelante se deben referenciar así. |
| `$*` | Todos los argumentos. |

## 7.1 if/else

```
if [cond]; then
    [codigo]
elif [cond2]; then
    [codigo2]
else
    [codigo3]
fi
```

- Para evaluar las condiciones se ha de usar (( )) o [ ], si se desea realizar por valor numérico se debe hacer con un =, si es cadena con ==. Los que menos fastidian son los [[ ]].

## 7.2 read

- Imprimir por pantalla al leer:

```
read -p "Esto sale en pantalla" [variable]
```

- Leer sin que se necesite tocar el Enter. En este caso espera a un solo carácter:

```
read -n 1 [variable]
```

## 7.3 Bucles

### 7.3.1 for

- Forma genérica

```
for var [in lista]
do
    ...
done
```

- Usando metacarácteres

```
for var in {1..9}
do
    ...
done
```

- Usando seq

```
for var in `seq x y`    # Donde x < y
do
    ...
done
```

- Estilo C++

```
for (( i=0; i<x; i++ ))
do
    ...
done
```

#### 7.3.2 while

```
while [condicion]
do
    ...
done
```

#### 7.3.3 until

```
until [condicion]
do
    ...
done
```

- En efecto es como un while pero con la condición invertida, es decir, el bucle se realiza mientras la condición no se cumple. Una vez que se cumple, finaliza el bucle.

### 7.4 case

```
case [expresion] in
    patron1 )   #Se pueden colocar mas
        ... ;;  # patrones separandolos con |
    patron2 )
        ... ;;
    patron3|patron4 )
```

```
        ... ;;
esac
```

### 7.5 Funciones

```
function _[nombre de la funcion]
{
    ...
}

_[nombre de la funcion]()
{
    ...
}
```

- Los parámetros se acceden como si fueran otro guión, es decir, tienen sus propias variables `$#, $1,` etc.
- Para declarar variables locales se añade local al principio.
- Para devolver un valor, se utiliza return.

## 8 Control de trabajos en Bash

- `jobs`
  Lista los trabajos bajo el control del usuario
- `<ejecutable> &`
  Manda la ejecución del ejecutable a segundo plano.
- `ps`
  Muestra el estado de todos los procesos del sistema.
- `kill`
  Manda señales a los procesos.
  - Señales esenciales: (Se ven con `kill -l`)

| | | |
|---|---|---|
| 15 | `SIGTERM` | Termina un proceso excepto si está pausado. |
| 9 | `SIGKILL` | Termina un proceso independiente de su estado. |
| 18 | `SIGCONT` | Reanuda la ejecución si estaba pausado. |
| 19 | `SIGSTOP` | Pausa la ejecución del proceso. |

  - ```
    kill <pid>       # <
    kill -15 <pid>   # < Equivalentes

    kill -9 <pid>
    kill -19 <pid>
    kill -18 <pid>
    ```
    ```

# 9 Compilación

- Generar archivo objeto:

```
g++ -c [archivo].cpp
```

- Generar ejecutable a partir de objeto:

```
g++ -o [ejectuable] [archivo].o
```

- Generar ejecutable con módulos por partes:

```
g++ -c [libreria1].cpp
g++ -c [libreria2].cpp
g++ -c [archivo].cpp
g++ -o [ejecutable] [archivo].o [libreria1].o
   [libreria2].o
```

- Generar ejecutable con módulos directamente:

```
g++ [archivo].cpp [libreria1].cpp [libreria2].
   cpp -o [ejecutable]
```

- Generar ejecutable para depuración:

```
g++ -g -o [ejecutable] [archivo].o
```

- Generar ejecutable con bibliotecas:

```
g++ -I./ -L./ -o [ejecutable] [archivo].o -l[
   biblioteca]
```

  - -I Especifica el directorio dónde está la cabecera.
  - -L Especifica el directorio dónde está la biblioteca.
  - -l Especifica que el nombre de la biblioteca inicia con lib y finaliza en .a

## 9.1 Bibliotecas

- Generar biblioteca a partir de archivos objeto:

```
ar -rvs lib[biblioteca].a [libreria1].o [
   libreria2].o [libreria3].o
```

# 10 Makefiles

## 10.1 Invocación

Si el archivo se llama makefile, Makefile o makefileGNU se invoca llamando:

```
make
```

Si posee otro nombre, se especifica:

```
make -f [nombreMakefile]
```

## 10.2 Sintaxis genérica

```
variables

objetivo1: dependencia1 ... dependenciaM
    orden1
    orden2
    ...
    ordenN

objetivo2: ...
    ...
```

- Cada objetivo, junto a las dependencias y las órdenes se denomina una regla.
- Si una dependencia tiene una modificación, esto hará que make recompile cada objetivo que posea esa dependencia.
- Si una regla que no posee órdenes, make simplemente chequeará que las dependencias estén actualizadas, esto se llama un objetivo simbólico.
- Una regla sin dependencias se le llama una regla virtual, para poder invocarlas se debe especificar en make el nombre del objetivo.

  ```
  make clean
  ```

## 10.3 Variables especiales

# 11 Depurador GDB

- Iniciar depurador:

  ```
  gdb [ejecutableConFlag]
  ```

- Iniciar depurador con un guión:

| | |
|---|---|
| `$@` | Representa el nombre del objetivo de la regla en la que nos encontramos. |
| `$<` | Representa la primera dependencia de la regla en la que nos encontramos. |
| `$<` | Representa las dependencias de la presente regla que hayan sido actualizadas (modificadas) dentro del objetivo de la regla y separadas por un espacio en blanco. |
| `$^` | Representa todas las dependencias separadas por un espacio en blanco.. |

```
gdb -x [nombreGuion].gdb [ejecutableConFlag]
```

- Iniciar programa dentro de depurador:

```
run
```

- Visualizar el código:

```
list [n]     # n es nro de linea
l [n]        # otra forma
```

- Breakpoints:
  - Creación:

```
break [linea_o_funcion]
b [linea_o_funcion]
```

  - Condicionales:

```
break [numero_del_break] if [variable] [
    condicion]
```

  - Visualizar breaks:

```
info b
```

  - Eliminarlo:

```
delete [numero_del_break]
```

  - Desactivarlo:

```
disable [numero_del_break]
```

Mientras se está en un breakpoint, se puede:

- Avanzar línea a línea:

```
step
s
```

- Avanzar pero no entrar en funciones:

```
next
n
```

- Continuar ejecución hasta siguiente breakpoint:

```
continue [N]
```

Se pueden saltar los siguientes N breakpoints si se añade ese parámetro.

- Visualizar una variable cada vez que se llegue a un breakpoint:

```
display [nombreVariable]
```

- Borrar el display:

```
delete display [numero_de_display]
```

- Visualizar las variables locales del programa:

```
info locals
```

- Cambiar el valor de una variable:

```
set variable [variable]=[valor]
```