Scripting

Introducción

Iñaki Alegria Roberto Cortiñas Mikel Larrea

Resumen

El intérprete de comandos y su utilización de modo automatizado –*scripting*– son fundamentales en la administración de sistemas y otras tareas. En este documento se presentan los fundamentos básicos de *scripting*, junto con una pequeña referencia de los comandos básicos.

Índice

1. Conceptos básicos

1.1. Introducción

Todos los sistemas operativos ofrecen la posibilidad de utilizar las funciones del propio sistema operativo o las de otros programas. Por ejemplo, en *Windows* la aplicación de nombre *Explorer* permite acceder a un conjunto de recursos del sistema operativo. En ocasiones a este tipo de interfaces se les denomina *shell*.

En los sistemas de tipo *Unix* es habitual utilizar como *shell* los denominados intérpretes de comandos (IC). Los intérpretes de comandos son herramientas muy potentes, aún a pesar de que no suelen tener interfaz gráfico (o quizá por eso mismo). Además de ofrecer la posibilidad de acceder a las funciones del sistema operativo, permiten agrupar las tareas. De este modo, encadenando tareas sencillas (comandos), se implementan tareas complejas, permitiendo así su automatización y ejecución desatendida. El modo más sencillo de implementar este tipo de tareas es mediante el denominado *shell scripting*, basado en la utilización de cuasi-lenguajes de programación que permiten enlazar las tareas sencillas mencionadas anteriormente.

Un intérprete de comandos es un mero programa; recibe datos de entrada, los procesa y devuelve una salida. Por ejemplo, puede recibir por la entrada estándar (teclado) los datos introducidos por el usuario (instrucción a ejecutar), los procesa (mediante llamadas internas le indica al sistema operativo que debe ejecutar la instrucción) y, por último, muestra el resultado (de la ejecución de la instrucción) al usuario por la salida estándar (terminal).

Como programa que es, también finaliza en algún momento. Por ejemplo, cuando desde el entorno gráfico se abre un terminal, dentro de la ventana gráfica correspondiente se ejecuta el intérprete de comandos por defecto. Al cerrar la ventana, el intérprete de comandos también finaliza. Este mismo proceso se sigue cuando se inicia una sesión (*login*) y al terminarla (*logout*).

Intérprete de comandos

- Interfaz para la introducción de instrucciones
- Es un programa normal
  - Terminal
  - *login* al iniciar y *logout* al terminar
- Entrada: sentencias
  - *prompt*: $ o # (+ información adicional)
  - Interpretadas
- Salida: resultado de la ejecución
- Las sentencias se pueden acumular
  - Sentencias simples ⇒ complejas
- Interactivos o en modo *batch* (*shell scripting*)

Introducción

- *sh*: portabilidad
- Interpretado
- Estructura básica

Opciones

- *sh*: *Bourne Shell*
- *Bash*: *Bourne-Again Shell*
- *Dash*: *Debian Almquist Shell*
- *ksh*: *Korn Shell*
- ...

Criterios

Portabilidad y eficiencia

```
$ time bash -c 'for i in *; do echo "$i">/dev/null; done'↲
real    0m0.008s
user    0m0.000s
sys 0m0.008s
$ time dash -c 'for i in *; do echo "$i">/dev/null; done'↲
real    0m0.004s
user    0m0.000s
sys 0m0.004s
```

Ventajas

- Flexibilidad frente a los lenguajes compilados
- Aproximación *top down*
  - Estructuración por piezas
  - Complejidad ⇒ tareas simples
  - Filosofía de *Unix*

  Ejemplo: fichero de inicio en */etc/rc.d* (*Linux*)

Desventajas

- Menor eficiencia que lenguajes compilados
  - Procesos orientados a cálculo
  - Acceso al sistema de ficheros
- Programación: características avanzadas, gestión de variables (declaración y tipado), estructuras de datos complejas...
- Bibliotecas gráficas
- Características avanzadas en comunicación
- Problemas de licencia (código no protegido)

Intérpretes de comandos: ¿cuál utilizar?

Hoy en día se puede elegir entre muchos intérpretes de comandos. A la hora de seleccionar uno, debemos tener en cuenta qué criterios seguir, como por ejemplo facilidad de uso, potencia o portabilidad. Una recomendación: utilizar dos. El primero para trabajar de modo interactivo y el segundo para programar *script*s.

- Interactivo: como se trabaja directamente, será conveniente que tenga las siguientes características: edición de comandos, histórico, completado cuasi-automático de comandos y parámetros,.... Todas estas características facilitarán el trabajo en el terminal. En este caso se ha elegido *bash*.
- Para programar *script*s: en general, los *script*s están pensados para ser ejecutados sin la presencia del usuario. Esto implica que las características mencionadas en el caso anterior no son importantes aquí. En este caso el criterio principal es la portabilidad, es decir, poder utilizar el mismo *script* en la mayor cantidad de sistemas sin tener que adaptar el código (reduciendo así costes de mantenimiento). Por ejemplo, *bash* (*Bourne Again Shell*) es un intérprete de comandos con características muy interesantes para la programación de *script*s, pero puede haber incompatibilidades al intentar ejecutar sus *script*s en sistemas que no sean *Linux* (no tienen por qué tener *bash* instalado). Por este motivo es preferible apostar por intérpretes de comandos más sencillos pero también más portables, aún a costa de perder capacidades interesantes, para asegurar la compatibilidad con la mayor cantidad posible de sistemas. En este documento se utilizará *sh* (*Bourne Shell*) para programar *script*s.

  Nota: se podría decir que *sh* es el precursor de los actuales *shell*. Si bien carece de algunas de las funcionalidades avanzadas de estos últimos, es muy adecuado para la programación de *script*s ya que está disponible por defecto en muchos sistemas. En ocasiones no se encuentra directamente instalado, sino que que se emula a través de otro IC más potente. Por ejemplo, en sistemas *Linux* es habitual que *sh* se ejecute a través de un modo compatible de *bash*. En *Ubuntu*, en cambio, *sh* se emula por medio de *dash* (*Debian Almquist Shell*), IC propuesto como sustituto a *sh*[1]. Si se desea cambiar en *Ubuntu* el IC que emula *sh*, bastará con ejecutar *sudo dpkg-reconfigure dash*: seleccionando *Sí*, quedará ligado a *dash*, de lo contrario a *bash*.

[1] En realidad, *dash* proviene de *ash* (*Almquist Shell*) propuesto en el sistema BSD.

Identificando el intérprete de comandos (IC)

¿Cómo saber qué IC estamos utilizando?

```
$ echo $0↲
bash            # El programa en ejecución es Bash
$ sh↲   # Ejecutar el interprete sh
$ echo $0↲      # Mostrar el programa en ejecución
sh
$ exit↲          # Terminar el programa en ejecución (sh), (comando exit)
$ echo $0↲
bash              # Se ha regresado al proceso original
```

Como se ha mencionado anteriormente, un IC es un programa normal. Esto implica que puede ser ejecutado como cualquier otro; es decir, desde el IC estándar del sistema podemos lanzar aquel que se desee en ese momento. Por ejemplo:

Como se verá más adelante, precisamente gracias a esta característica se puede utilizar el IC que se desee para programar *script*s, independientemente del IC que se esté utilizando por defecto.

1.2. Sentencias

Utilización básica

Cuando se desea ejecutar un programa, se deberá indicar al IC qué programa se quiere ejecutar. En ocasiones, también será necesario indicar otros datos, como las opciones de ejecución de ese programa o los argumentos que necesita.

Supongamos que se desea visualizar los ficheros del directorio principal del usuario:

Estructura de una sentencia

Ejemplo:

```
$ ls -l /home/user       # Comentario
```

Estructura

- Invitación o *prompt*
- Comando
  - Camino absoluto o relativo
  - Permiso de ejecución
- Opciones
- Argumentos: obligatorios (en ocasiones) u optativos
- Comentario/nota: por medio del carácter #[2]

[2] No confundir con el mismo carácter cuando aparece en el *prompt*.

En más detalle

- Carácter *$*: carácter mostrado por el IC para indicar que está preparado para recibir órdenes (motivo por el que se denomina *prompt*). Normalmente con el superadministrador, es decir, el usuario *root*, se muestra el carácter *#*. En cualquier caso, el carácter que aparece en el *prompt* es configurable (ver variable *PS1* del entorno del proceso) .
- Ubicación del programa o *path*. Hay dos opciones: ubicación absoluta (desde la raíz del sistema de ficheros) o relativa (desde el directorio actual o a través de la variable de entorno *PATH*). En el ejemplo se utiliza *ls* (relativa), pero también se podría utilizar */bin/ls* (absoluta).
- Opciones de ejecución del programa. En muchas ocasiones, los programas a ejecutar disponen de opciones para modificar el modo en que leen los datos de entrada, los procesan o muestran los resultados de salida. En ocasiones son implícitas y no es necesario indicarlas (se utilizan valores por defecto), pero a menudo es necesario indicarlas de modo explícito. Para indicar opciones se utiliza el carácter *-*. En el ejemplo se utiliza *-l* (*long*) para indicar que la salida del programa deberá mostrar el detalle de cada fichero mostrado (la opción por defecto es mostrar sólo el nombre de cada fichero).
- Argumentos: datos de entrada del programa. En el ejemplo */home/user*, para indicar que se desea obtener la lista de ficheros de ese directorio.
- Carácter *#* para indicar comentarios. Todo texto a partir de este carácter hasta el final será omitido por el IC. Normalmente sólo se utiliza en los *script*s para documentarlos. Como se ha indicado anteriormente, el mismo carácter también aparece en ocasiones en el *prompt*, pero no deben ser confundidos. En el caso del *prompt* es el IC el que los escribe, mientras que en el caso de un comentario es el usuario.

Secuencia de sentencias

- Una instrucción por cada línea
- Excepciones:
  - Una sentencia en varias líneas

```
$ echo "Hola, ¿qué tal?"↲
Hola, ¿qué tal?
$ echo "Hola , \
> ¿qué tal?"↲
Hola, ¿qué tal?
```

  - Varias sentencias en una línea

```
$ echo -n "Hola" ; echo ", ¿qué tal?"↲
Hola, ¿qué tal?
```

1.3. Entrada y salida

Entrada/salida

Tres estándar; por defecto del teclado y a la pantalla

- Entrada estándar (*stdin*)

```
$ cat↲
hola↲      # Texto introducido por el usuario
hola       # leído por cat y escrito en pantalla
```

- Salida estándar (*stdout*)

```
$ echo "Texto"↲
Texto
```

- Salida estándar de errores (*stderr*)

```
$ ls no-existe↲
ls: no se puede acceder a 'no-existe': No existe el fichero o directorio
```

Redireccionando la entrada/salida

- *stdin*: $<$

```
cat < fichero
mail usuario < fichero
```

- *stdout*: $>$ (ó *1*$>$).

```
ls > fich     # o ls 1> fich
```

Append $>>$

```
ls >> fich
```

- *stderr*: *2*$>$

```
ls no-existe 2> errores
ls no-existe 2>> errores # append
```

Combinando la entrada y salida

- Las redirecciones se pueden combinar:

```
ls > listado 2>errores
```

- La entrada y salidas también se pueden referenciar mediante el carácter *&*: *&0*, *&1*, *&2* respectivamente.

```
ls > listado 2>&1 # los errores también a stdout
```

- Duplicando la salida: comando *tee*

```
ls | tee listado
```

Pipes

- *comando1* $\xrightarrow{datos}$ *comando2*

```
ls | more
```

De lo contrario:

```
ls > file-temp
more < file-temp
rm file-temp
```

Ejemplo:

```
ps -aux | grep user | wc -l > results
```

Dispositivos especiales

Se pueden utilizar en las redirecciones (ej. pantalla */dev/tty*)

- Nulo: */dev/null*

```
cat fichero > /dev/null  # Sin salida
cat /dev/null > log_fich # Para vaciar el fichero
```

- Aleatorio: */dev/random*
- Ceros: */dev/zero*

Ejercicios

1. Averigua cuál es el intérprete de comandos por defecto en tu sistema.
2. Lanza *sh* desde el terminal, comprueba que realmente se está ejecutando y regresa al IC original.
3. Guarda los nombres de las entradas (contenido) del directorio */etc* en el fichero de nombre *file-tmp*.
4. Añade al fichero del punto anterior los nombres de las entradas (contenido) del directorio */bin*.
5. En una sola instrucción, guarda los nombres de las entradas (contenido) de los directorios */etc* y */usr* en el fichero *file-tmp*.

2. Trabajando con el intérprete de comandos: *bash*

Introducción

Introducción

- *Bourne-Again Shell*
- Mayor interactividad
- *Scripting*: más capacidades pero menos portabilidad

Edición

Edición de comandos (I)

- Edición del comando actual editable
  - Teclas de cursor
  - Combinación de teclas (más adelante)
- Histórico de comandos (*.bash_history*)
  - Comando *history*
  - Repetir sentencia previa: *num-comando*, *nombre-parcial*, (último comando)
  - Teclas de cursor

Edición de comandos (II)

- Completado automático de comandos/parámetros
  - */etc/bash_completion*, */etc/bash_completion.d/*
  - Tecla TAB
  - Combinación de teclas
- Control de procesos:

  ctrl+c detener la ejecución del proceso actual
  ctrl+z detener (pausar) la ejecución del proceso actual (*fg, bg, jobs*)
  ctrl+d fin de la entrada de un proceso (ej. *read*)
- Modificable con *stty*

Combinaciones de teclas en terminal

ctrl+	a/e	mover el cursor al comienzo/final de la línea
	t	intercambiar los dos caracteres bajo el *cursor*
	u/k	borrar desde el *cursor* hasta el comienzo/final de la línea
	l	borrar el terminal (comando *clear*)
	p/n	obtener la línea anterior/siguiente en el histórico
	r	buscar en el histórico
	s/q	detener/recuperar la entrada estándar

Varios

- Paginación en terminal: *shift + PageUp / PageDown*
- Gestión del portapapeles en sistemas *Unix*:
  - Copiar: seleccionar texto con el ratón
  - Pegar: botón central del ratón

Similitudes con el editor *vim*

Teclas de acceso rápido man y *less*

j	Avanzar una línea en la pantalla
f	Avanzar una página en la pantalla
k	Retroceder una línea en la pantalla
b	Retroceder una página en la pantalla
g	Ir a la primera línea
G	Ir a la última línea
/	Buscar texto
n	Repetir la última búsqueda (hacia delante)
N	Repetir la última búsqueda (hacia atrás)
q	Salir (*quit*)

Similitudes con el editor *vim*

Teclas de acceso rápido avanzadas man y *less*

- Editar directamente el fichero: pulsar *v*
- Utilizar marcas

ma	Marcar la posición actual de pantalla en el marcador *a* (*a*, *b*, *c*...)
'a	posicionar la pantalla en la posición determinada por el marcador *a*

- Vigilar los cambios que se realicen al fichero. Pulsar *F* al visualizar el fichero (comportamiento similar a *tail -f fichero*)
- Ver varios archivos al mismo tiempo

:e	Ver más ficheros
:n (*next*) ó :p (*previous*)	Moverse entre los ficheros que se están visualizando

Influencia de los editores en la línea de comandos

- Por defecto se utilizan semejanzas a *Emacs*
- Se puede cambiar a *vi/vim*

```
set -o vi
# set –o emacs # por defecto
```

2.1. Gestión de tareas en un terminal

Tareas en el terminal

- Ejecución secuencial de órdenes en primer plano (*foreground*): una a una

```
$ com1↲
$ com2↲
```

- Ejecución de órdenes en segundo plano (*background*). Utilizar &

```
$ com1 &↲
$ com2 &↲
$
```

  Ventaja: los procesos se ejecutan concurrentemente

Comandos para gestionar tareas

jobs Mostrar la lista de tareas

```
$ jobs↲
[1]+  Stopped                 gedit
[2]-  Running                 kate &
```

fg (*foreground*) Traer a primer plano una tarea que está en segundo plano

bg (*background*) Poner en marcha en segundo plano una tarea que está suspendida

Gestión de tareas

- Enviar a segundo plano

```
$ com1 &↲
# o
$ com1↲ # y pulsar ctrl+z (stop)
$ bg com1↲ # o bg num_tarea (info mostrada en jobs)
```

  Ejemplo: estamos trabajando con un editor de texto en el terminal y queremos ejecutar un comando sin detener el editor. ¿Cómo?

- Operaciones con una tarea que está en segundo plano:
  - Ponerla en marcha (si está parada) *bg num-tarea*
  - Traerla a primer plano *fg num-tarea* (sin parámetros, la última)
- Para ver el número de tarea ⇒ *jobs*[3]
- Detener una tarea. Opciones:
  - Si está en primer plano ⇒ ctrl+c
  - Si está en segundo plano ⇒ *kill %num-tarea* o *kill PID*

[3] La opción *-l* también nos permite ver su PID.

3. Programación de *Script*s

3.1. Estructura de un script

Estructura de un *script*

- Cabecera
- Descripción
- Sentencias
- Devolver código de finalización

```
#!/bin/sh       °# cabecera
# descripción

sentencias

exit codigo-finalización
```

Cabecera

- Concepto semejante al de *magic number*
- #!/bin/sh (*sha-bang* ó *shebang*: sharp(#) + bang (!))
- Otras opciones:

```
#!/bin/sh
#!/bin/bash
#!/usr/bin/perl
#!/usr/bin/tcl
#!/bin/sed -f
#!/usr/awk -f
```

Descripción

- Documentación
- Objetivos, parámetros necesarios, salida…

Ejemplo:

```
# Comentario (explicación del script)
```

Sentencias

- Comandos
- *script*s
- Asignación de variables
- Estructuras de control de flujo: *if*, *for*...
- ...

Código de retorno

- Resultado de la ejecución
  - Éxito: *exit 0* (o *exit*)
  - Error: *exit codigo_error*
- echo $? ⇒ código de retorno de la última ejecución

Ejemplo:

```
$ ls no-existe↲
ls: no se puede acceder a no-existe: No existe el fichero ó directorio
$ echo $?↲
2         # codigo de retorno de ls
$ echo $?↲
0         # codigo de retorno de echo
```

Script de ejemplo

```
#!/bin/sh

# script de prueba.
# No recibe parametros y solo muestra en pantalla
# un texto

echo "Texto"

exit 0
```

3.2. Ejecución

Introducción

- Fichero de texto
- Contiene sentencias
- Interpretado

Ejecución de *scripts*

a) Nombre del *script* como parámetro al intérprete

```
sh mi-script.sh
perl fichero.pl
```

b) Autocontenido

- Al comienzo del *script* indicar el intérprete (*sha-bang*)
- Asignar permiso de ejecución (comando *chmod*)

```
chmod +rx mi-script.sh
chmod u+rx mi-script.sh
```

- Para ejecutar

```
/home/user/bin/mi-script.sh # path absoluto
./mi-script.sh              # desde el catalogo
mi-script.sh                # configurando la variable PATH para que incluya /home/user/bin/
```

3.3. Redireccionando la entrada/salida

Modificar la entrada/salida desde el propio *script*

exec Este comando permite redireccionar la entrada y/o salida del proceso (*script*) en ejecución.

```
#!/bin/sh
exec  1>'file1.log'
...
```

Ejercicios

1. Programar un *script* que muestre el nombre del *script*.
2. ¿Dónde debería ser copiado el *script* para que se pueda ejecutar como cualquier comando del sistema (sin hacer referencia al directorio en el que está), como por ejemplo, *$ ls*?
3. Programar un *script* que obtenga un texto (cualquiera) por la salida estándar y otro por la salida estándar de errores.
4. Modificar el *script* del punto anterior para que desde el interior del propio *script* redirija la salida estándar al fichero "*nombre-del-script*-output.txt" y la de errores a "*nombre-del-script*-errors.txt".

3.4. Variables

Introducción

- Asociación dato ⇔ etiqueta
- Utilización
  - Asignación: *VARIABLE=*
  - Acceder a valor: *$VARIABLE* o *${VARIABLE}*
- Por defecto tipo texto
- Ejemplos:

```
A="Texto"
A = "Texto"    # Mal. Cuidado con los espacios
echo $A
echo "$A"
echo "${A}"
echo ${A}001    # no $A001
B="Esto es un $A"
```

Variables (II)

- Para borrar el valor de una variable
  - *unset variable*
  - Al terminar el proceso (terminal), la variable desaparece (entorno de trabajo)
- Leer del teclado: comando *read*

```
read VAR
read VAR1 VAR2 VAR3    # valores separados por valor una variable denominada IFS
```

Tipos de variables

- Implícitas, explícitas
  - Variables locales
  - Variables de entorno
  - Variables posicionales (parámetros)
- Ámbito (*scope*)

Variables posicionales

- $0

```
# script que muestra el nombre del script
echo $0
```

- Lista de argumentos: $* ó $@
- Número de argumentos: $#
- Argumentos $1-$n (n==$#)

```
echo "El número de argumentos es $#"
for i in $* # o "$@", si se desea entrecomillar las variables
  do
   echo $i
  done
```

- ¡*Pipe*! ⇒ comando *xarg*

Operaciones aritméticas

- Por defecto tipo texto
- $(())

```
a=1
b=$(($a+1))     # b==2
```

- Comando *expr*

```
a=1
b=`expr $a+1`     # b==2
```

Variables: acceso avanzado al contenido

```
$ FILE="/home/user1/file1.txt"↲
$ echo ${FILE#/home/}↲     # eliminar la primera ocurrencia desde el principio
user1/file1.txt
$ echo ${FILE#/*/}↲        # también se pueden utilizar metacaracteres
user1/file1.txt
$ echo ${FILE##*/}↲        # eliminar la ocurrencia mas extensa desde el principio
file1.txt
$ echo ${FILE##*.}↲
txt
$ echo ${FILE%.*}↲         # eliminar la primera ocurrencia comenzando por el final
/home/user1/file1
$ echo ${FILE%/*}↲         # eliminar la primera ocurrencia por el final
/home/user1
$ echo ${#FILE}↲           # número de caracteres
21
```

Variables: procesando el contenido

- En las variables también se pueden almacenar comandos
- Para ejecutarlos ⇒ *eval*

```
$ options="la > test"↲
$ command="ls -$options"↲
$ $command↲ # ERROREA
$ eval "$command"↲
```

Ejercicios

1. Programa un *script* que calcula y muestra la suma de tres números que recibe como parámetro.
2. Programa un *script* que calcula y muestra la suma de los números que recibe como parámetro. El número de parámetros no se conoce a priori.
3. Como el anterior, pero en lugar de recibir los números como parámetros, los recibe a través del teclado.
4. Programa un *script* que muestre sólo los nombres de los ficheros del directorio principal, es decir, sin la parte correspondiente al directorio en el que están ni la extensión. Ejemplo: */home/user/file1.txt* ⇒ *file1*

3.5. Comillas (quoting)

Comillas

Tres tipos:

- Simples: '
- Dobles: "
- Invertidas (acento grave): `(*command substitution*)

Comillas simples y dobles

Asignación de texto:

```
A="Text"
B='$A'     # B=="$A"
C="$A"     # B=="Text"
```

Command substitution

Para sustituir el resultado de la ejecución del comando donde se desee:

- Comilla invertida `` `comando` ``
- POSIX: $(*comando*)

```
$ echo "El número de procesos es `ps -aux | wc -l`"↲
El número de procesos es 98
$ echo "El número de procesos es  $(ps -aux | wc -l)"↲
El número de procesos es 98
```

3.6. Estructuras de control

Estructuras

- Lógica, condiciones (*test*)
- *if*
- *case*
- Bucles (*looping*)
  - *for*
  - *while*
- Funciones

Establecer condiciones

- Útil para *if*, *while*, *for*...
- Establece una condición:
  - Se cumple ⇒ devuelve 0
  - No se cumple ⇒ devuelve valor $\neq 0$
- Comando *test*

Comando *test*

- Uso general
  - *test condición* o *[condición]*
  - Negar condición: *test ! condición* o *[! condición]*
  - AND: *test condición1 -a condición2*
  - OR: *[condición1 -o condición2]*
- ¡Cuidado con los espacios!
- Consultar resultado: *echo $?*

Comando *test*: condiciones

- Texto:

```
test TEXTO  # FALSE si vacío
test TEXTO1 = TEXTO2  # TRUE si iguales. También se puede utilizar !=
```

- Números:

```
test NUM1 -eq NUM2  # TRUE si iguales. Otras:−ne,−ge,−gt,−le,−lt
```

- Ficheros

```
test -f FILE # TRUE si fichero regular. Otras: −d, −L ...
test -s FILE # TRUE si fichero no vacío
test -r FILE # TRUE si fichero permiso lectura. Otras: −w, −x, −u, −g
```

if

Estructura tradicional: if *condición* then *sentencias*. Parte *else* opcional.

```
if grep "texto" fichero > /dev/null
  then
    echo "Se ha encontrado la cadena texto en el fichero"
  else
    echo "No se ha encontrado la cadena texto en el fichero"
  fi
```

Otro modo:

```
$ comando1 && comando2                  # if a) then b)
$ mkdir nuevo-dir && cd nuevo-dir
$ apt-get update && apt-get upgrade     # if a) then b)
$ comando1 || comando2                  # if ! a) then b)
ls no-existe >/dev/null 2>&1 && echo "SI" || echo "NO"
```

if: ejemplos

Recoger el identificador de usuario que se pasa por parámetro e indicar si está conectado.

```
id="$1"
if who | grep "$id" >/dev/null
  then
    echo "$id conectado"
  else
    echo "$id no conectado"
  fi
```

Comprobación de parámetros:

```
# Comprobar que al menos hay un parámetro
if [ $# -lt 1 ]
  then
    echo "Error: no hay parámetros"
    exit 1
  fi
```

case

```
case "$1" in
  start)
    sentencias
    ;;

  stop)
    sentencias
    ;;

  restart)
    $0 stop
    $0 start
    ;;

  *)
    echo "Utilización: $0 {start|stop|restat}"
    exit 1
esac
```

for

```
A="1 2 3 4 5"
for I in $A # for I in $(seq 5)
  do
    echo "${I}"
  done
```

Mostrar las 10 primeras líneas de los ficheros regulares de extensión ".h"

```
for i in *.h
  do
    if [ -f $i ]    # fichero regular
      then
        echo "======== $i fichero ========="
        head $i
    fi
  done
```

while

```
I=1
while [ $I -le 5 ]
  do
    echo "$I"
    I=$(($I+1))
  done
```

```
ls -l | tail --lines=+2 |
while read MODE LNK USER GROUP SIZE DATE HOUR NAME
  do
    echo "____________________"
    echo "Name : $NAME"
    echo "User : $USER"
    echo "Group: $GROUP"
    echo "Date : $DATE"
    echo "Hour : $HOUR"
    echo "Link : $LNK"
    echo "Mode : $MODE"
    echo "Size : $SIZE byte"
  done
```

Ejemplo

Listar las diez primeras líneas de los ficheros de cualquiera de los tipos que se pasan como parámetro:

```
if [ $# -lt 1 ] # Comprobar que al menos hay un parámetro
  then
    echo "Error: no hay parámetros"
    exit 1
  fi

for tipo in $* # por cada parámetro
  do
    for i in *.$tipo
      do
        if [ -f $i ]
          then
            echo "======== $i fichero ========="
            head $i
          fi
      done
  done
```

Ejercicios

1. Programa un *script* que compruebe que recibe dos parámetros.
2. *script* que recibe como parámetro dos números e indica cuál es el de mayor valor.
3. *script* que comprueba que el fichero que recibe como parámetro existe.
4. *script* que comprueba que el directorio que recibe como parámetro existe y después cuenta cuántos subdirectorios y cuántos ficheros contiene.
5. Como en el caso anterior, pero contando el número de ficheros con extensión *txt*.
6. *script* que cuenta el número de ficheros de tipo *txt* en el directorio de trabajo del usuario.
7. *script* que muestra cuántas extensiones diferentes hay en un directorio, junto con el número de ocurrencias de cada una. Ejemplo:

   'txt' $\Rightarrow$ 4

   'pdf' $\Rightarrow$ 2

   ...

4. *Wildcarding* y expresiones regulares

4.1. *Wildcarding/metacaracteres*

Wildcards

- Para indicar grupos de ficheros ⇒ *file globbing*
- El *Shell* lo interpreta, obtiene la lista y le pasa esta última al comando

Caracteres *Wildcard*

*	cualquier cadena de texto (vacía o de longitud n)
?	cualquier carácter (uno)
[xy]	el carácter x o y
[x-y]	cualquier carácter del rango indicado (de x a y)
[xy,zu]	caracteres xy o zu
~	directorio $HOME$ del usuario
~ user	directorio $HOME$ del usuario *user*
!	para negar los anteriores

Ejemplos

```
$ touch file file1 file2 file3 file4↲
$ ls file*↲
file  file1  file2  file3  file4
$ ls file?↲
file1  file2  file3  file4
$ ls file[1-2]↲
file1  file2
$ ls file[!1-2]↲
file3  file4
$ ls file[1,2,3]↲
file1  file2  file3
$ touch .hidden-file1↲
$ ls *file*↲
file  file1  file2  file3  file4
$ ls .*file*↲
.hidden-file1
```

4.2. *Expresiones regulares*

Expresiones regulares (*regexp*)

- Para expresar cadenas de texto
  - Líneas que comienzan con el texto “xxxx”
  - Palabras que terminan con el texto “yyy”

  - ...

- Se utiliza para: búsquedas, filtros, sustituciones...
- Gran capacidad
- Procesamiento por línea[4]
- Programas: *grep,sed,awk,Perl,Python,Ruby,less,vim,emacs,...*

Patrones para expresar un carácter

x	carácter *x*
[xy]	caracteres *x* o *y*
[x-y]	cualquier carácter en un rango (de *x* a *y*)
[^xy]	cualquier carácter que no sea ni *x* ni *y*
[^x-y]	cualquier carácter que no esté en el rango indicado
.	cualquier carácter [5]

Ejemplos

Patrón	Resultado[6]	Patrón	Resultado
a	"a", "xxayy"	ab	"ab", "xxabyy"
[aA]	"a", "A"	ab	"ab", "xxabyy"
[a-c]	"a", "b" o "c"	[0-9]	cualquier dígito, "x3y"
..	al menos dos caracteres	[^0-9]	ningún dígito

Conjuntos predefinidos

- `[:alpha:]` ⇒ alfabeto == [a-zA-Z]
- `[:lower:], [:upper:]`
- `[:digit:]` ⇒ números == [0-9]
- `[:alnum:]` ⇒ [:alpha:] + [:digit:]
- `[:blank:]` ⇒ espacio + TAB

[4] También multilinea

Ejemplos

Patrón	Resultado[7]
[[:lower:]]	"a"
[[:lower:][:upper:]]	"aA", "bC"
[[:digit:][:digit:]]	dos dígitos
[0-9]	cualquier dígito, "x3y"
..	al menos dos caracteres
[^0-9]	ningún dígito

Posición dentro de la linea

^/ $	Al comienzo / final de la línea
\< / \>	Al comienzo / final de una palabra

Número de ocurrencias

x	el carácter *x* sólo una vez
x+	una o más veces
x*	cero o más veces
x{n}	n veces
x{n,} / x{,n}	al menos / como máximo n veces
x?	opcional. Como mucho una vez

Básico (*basic*) vs extendido (*extended*). Ej: *grep* vs *egrep*

- ? ⇒ \?
- {,} ⇒ \{,\}

Ejemplos

Mediante el comando *expr*

- *expr string : string* : compara dos strings
  - Si iguales, devuelve número de caracteres, si no 0
- *expr string : expr-regular*

```
$ A="Text 45"↲
$ expr "$A" : 'T*'↲
7
$ expr "$A" : 'aa'↲
0
$ expr "$A" : '.*[0-9]'↲
7
```

Otra posibilidad:

```
echo "$A" | egrep "regexp"
```

4.2.1. sed

```
var="      output.txt"
var=$(sed -e 's/^[[:space:]]*//' <<<"$var")
echo "|${var}|"
```

5. awk

awk (*A*ho, *W*einberger y *K*ernighan)

- Lenguaje de programación: sintaxis *C*
  - Entre *sh* y *Perl*
- Procesado de texto:
  - Expresiones regulares
  - Tablas *hash*
- Interpretado: disponible en casi todos los SOs
- GNU ⇒ *gawk*

Modo de trabajo

- Unidad de información: registro (= línea normalmente)
- Entrada (*pipe* o fichero): analizada línea a línea
- Por cada línea aplica */patrón/ acción*
  - Patrón: condición a cumplir para poder aplicar la acción.
    - No *patrón* ⇒ la acción se aplica a todas las líneas
    - Puede ser una expresión regular
  - Instrucción(es): en formato/sintaxis *awk*

```
awk '/patron/ {acción}' fichero
comando | awk '/patrón/ {acción}'
```

Estructura básica de un programa *awk*

- Antes de procesar el fichero: BEGIN {acción}
- Después de procesar el fichero: END {acción}

```
BEGIN { instrucciones }
{aginduak} // erregistro bakoitzeko
END { instrucciones }
```

Formas de ejecución

- Línea de comandos

```
awk 'programa' file-input
some-command | awk 'programa'
```

- Guardar en un fichero, por ejemplo de nombre *script1.awk*:

```
#!/usr/bin/awk -f
programa
```

  Y después ejecutarlo

```
awk -f script1.awk file-input # ./script1.awk file-input
some-command | awk -f script1.awk
```

Ejemplos

Número de ficheros en el directorio

```
ls -l | awk 'BEGIN {count=0} {count+= 1} END {print count}'
```

Número de ficheros modificados en 2010 (en realidad, que contienen el texto *2010* en el resultado de *ls -l*)

```
ls -l | awk 'BEGIN {count=0} /2010/ {count+=1} END {print count}'
```

Estructurando la entrada

La entrada principalmente se estructura en los siguientes campos:

$0 uneko erregistro osoa (lerroa)

```
# Imprimir datos de todos los ficheros 'de 2010'
ls -l | awk '/2010/ { print $0 }' # ls -l | awk '$0 ~ /2010/ { print $0 }'
```

NR Número de registro (*Number of Record*)

```
# Imprimir datos de todos los ficheros 'de 2010', junto a sus posiciones
ls -l | awk '/2010/ { print NR,$0 }'
```

NF Número de campos en la línea actual (*Number of Field*)

```
# Imprimir número de palabras encontradas por cada línea
ls -l | awk ' { print NF }' fichero-texto.txt
```

$n Campo n: $1,... ${NF-1}, $NF

```
# Imprimir nombre de todos los ficheros 'de 2010'
ls -l | awk '/2010/ { print $8 }'
# Imprimir nombre de todos los ficheros modificados el 1/2/2010
ls -l | awk '$6=="2010-02-01" { print $8 }'
# Imprimir nombre de todos los ficheros modificados en febrero de 2010
ls -l | awk '$6~"2010-02" { print $8 }'
```

FS Separador de campos (*Field Separator*)

```
# Imprimir la información de todos los usuarios del sistema
# En el fichero /etc/passwd los campos están separados por ':'
awk -F":" '{print $1}'  /etc/passwd
awk 'BEGIN{FS=":";} {print $1}' /etc/passwd
```

Ejemplos

- Tamaño total de los ficheros del directorio de trabajo que han sido modificados en febrero de 2010:

```
ls -l | awk '/2010\-02\-/ { tot += $5 } END { print tot }'
```

- Frecuencia de palabras en un fichero de texto

```
awk '
# Por cada palabra que se encuentre en cada línea acumular la frecuencia
{
    for (i = 1; i <= NF; i++)
    freq[$i]++
}
# imprimir las frecuencias
END {
    for (word in freq)
        printf "%s\t%d\n", word, freq[word]
    }'
```

Registros vs campos

awk estructura la entrada en registros (líneas por lo general) y cada registro en campos (cada registro puede tener un número distinto de campos).

Un registro puede tener sus campos en varias líneas

```
BEGIN {
    FS="\n";
    RS="\n\n";
}
{
    print "$2\t$3"
}
```

Utilizar *awk* desde un *script* de *sh*

- Como en la línea de comandos
- En ocasiones pueden surgir problemas al querer utilizar variables de *sh* en el programa de *awk*.
  - Ejemplo: en el fichero que se pasa como segundo parámetro, mostrar las líneas que comienzan con el texto que se pasa como primer parámetro:

```
#!/bin/sh
awk '/^$1/ {print $1}' $2
```

- $1
  - En *sh* ⇒ primer parámetro
  - En *awk* ⇒ primer campo del registro

awk desde *sh*: opciones

a) Utilizar una variable adicional de *sh*

```
#!/bin/sh
SEARCH=$1
awk '/^"$SEARCH"/ {print $1}' $2
```

b) Utilizar una variable adicional de *awk*

```
#!/bin/sh
awk -v param="$1" 'BEGIN {search="^"param} $1 ~ search {print $1}' $2
```

Ejercicios

1. Imprimir el nombre del usuario con el identificador más alto (identificadores almacenados en el fichero */etc/passwd*).
2. Imprimir el nombre de los usuarios cuyo identificador e identificador de grupo sean distintos (en el fichero */etc/passwd*, el tercer y cuarto campo).
3. Mostrar los usuarios con identificador mayor que 100.
4. Contar las líneas vacías en un fichero.
5. Contar líneas que sólo contengan espacios o tabuladores.
6. Eliminar de un fichero las líneas vacías o que sólo tengan espacios y/o tabuladores.
7. Exportar el contenido del fichero */etc/passwd* a un fichero *html*. Los datos deberán aparecer en formato tabla:
   - a) Una fila por cada usuario
   - b) Una columna por cada usuario
8. Buscar en el fichero */etc/shadow* aquellos usuarios que tengan contraseña establecida.

6. Comandos de referencia

6.1. Fundamentos

Comandos iniciales

man : Obtener información sobre comandos y conceptos

Otras opciones: *apropos, info*

echo : Escribir un string en la salida estándar

printf : Escribir un string en la salida estándar

Como el anterior, pero con más opciones (formato . . .)

```
printf "This is $0\n"
printf "This is %s\n"
```

6.2. Gestión de ficheros y directorios

Listar ficheros y directorios

ls (*list*): Contenido del directorio / mostrar las características de ficherosj

Opciones:

- *a (all)*
- *l long list* con detalle (tamaño, permisos . . .)
- *d (directory)* sólo directorios
- *t (time)* ordenar por fecha
- *r (reverse)* orden inverso
- *S (size)* ordenar por tamaño

Ejemplos:

```
ls /home    # mostrar el contenido del directorio '/home'
ls          # mostrar el contenido del directorio de trabajo
ls -lt      # como el anterior, pero ordenado por fecha
ls -lSr     # por tamaño, de menor a mayor
```

Cambiar/consultar el directorio de trabajo

cd (*change directory*):

```
cd /home     # cambiar el dir. de trabajo a "/home"
cd ..        # cambiar el dir. de trabajo al dir. padre
cd -         # volver al directorio previo
cd           # cambiar el dir. de trabajo al dir. principal del usuario
cd ~         # como el anterior
```

pwd (*print working directory*):

```
pwd
```

Gestión de directorios

mkdir (*make directory*): Crear directorios

```
mkdir /tmp/prueba     # crear un dir. de nombre "prueba" en "tmp"
mkdir prueba          # crear "prueba" en el dir. de trabajo"
mkdir -p /home/usu/dir1/dir2/dir3 # Crear directorio, incluyendo directorios intermedios
```

rmdir (*remove directory*): Borrar directorios

Los directorios deberán deberán estar vacíos

```
rmdir /tmp/prueba    # borrar el directorio "/tmp/prueba"
rmdir prueba         # borrar el directorio "prueba" del dir. de trabajo
```

Copiar ficheros y directorios

cp (*copy*): Copiar ficheros o directorios

```
cp fich1 fich2   # en el dir de trabajo, copiar "fich1" con el nombre "fich2"
cp fich1 /tmp/. # copiar "fich1" al directorio "/tmp" (con el mismo nombre)
```

Al copiar directorios, también se debe copiar su contenido de modo recursivo.

```
cp -r /home /home_backup   # copiar el dir. "/home" y su contenido con el nombre "/home_backup"
cp -r /home /tmp/     # copiar el dir. "/home" y su contenido
                      # dentro del directorio "/tmp/" (con el mismo nombre)
```

Moviendo o renombrando ficheros

mv (*move*): Renombrar o mover ficheros (o directorios)

```
mv file1 /home/  # mover el fichero "file1" del dir. actual a "/home"
mv file1 /home/file2      # lo anterior, renombrando como "file2"
mv file1 file2   # renombrar "file1" a "file2"
```

Borrado de ficheros

rm (*remove*): Borrado de ficheros (o directorios)

```
rm file1     # borrar el fichero "file1" del directorio actual
```

En el caso de los directorios, antes de borrarlos es necesario borrar su contenido, para lo que se puede utilizar la opción (*recursive*).

Ejemplo: borrar el directorio *dir1* y su contenido.

```
rm -r dir1
```

Enlaces

ln (*link*): Crear un enlace a un fichero o directorio

```
ln -s /home/user /home/user-link # Enlace tipo soft
```

- Dos tipos: *symbolic* y *hard*
- Cuidado con los bucles
- A tener en cuenta al realizar búsquedas/operaciones generales

Listado de directorios

tree : Mostrar el subarbol asociado al contenido de un directorio

Estructura:[8]

```
$ tree↵
.
|-- folder1
|   |-- file1_1
|   `-- file1_2
`-- folder2
    `-- file2_1
```

6.3. *Examinar el contenido de ficheros*

Mostrar en pantalla

cat , more, less, tac (*conCATenate*): Visualizar el contenido de uno o más ficheros

[8] Es necesario tener instalado el paquete *tree*.

- *cat*

```
$ cat lines.txt↲
line l1
line l2
line l3
line l4
$ cat lines.txt lines.txt > double-lines.txt↲
```

- *more*: salida paginada (ajustada al tamaño del terminal)
- *less*: como *more*, pero con la opción de moverse hacia atrás/adelante, buscar ...
- *tac*: lo contrario a *cat*

Mostrar el comienzo (cabecera) del fichero

head :

```
$ head --lines=2 lines.txt↲ # o head −2 lines.txt. Mostrar las 2 primeras líneas
line l1
line l2
$ head --lines=-2 lines.txt↲ # Mostrar todas las líneas, salvo las dos últimas
line l1
line l2
line l3
```

- Por línea o byte
- Por defecto, las 10 primeras líneas

Mostrar el final de un fichero

tail :

```
$ tail --lines=2 lines.txt↲ # o tail −2 lines.txt. Mostrar las dos últimas lineas
line l4
line l5
$ tail -f /var/log/messages↲ # Mostrar las ultimas lineas y actualizar
                             # a medida que se añade información al fichero
```

Buscar texto en el contenido

grep, egrep, fgrep : Buscar texto en el contenido del fichero. Se busca línea a línea: si el texto buscado es encontrado en una línea, ésta es mostrada

Búsqueda por línea; si encuentra el texto en una línea, muestra toda la línea

```
# Buscar el usuario "user1" entre los usuarios
grep "user1" /etc/passwd --color
```

Opciones interesantes:

- *–color*: muestra en color el texto encontrado
- *-i*: no distingue entre mayúsculas y minúsculas
- *-v*: mostrar las líneas que NO contienen el texto
- *-c*: contar el número de ocurrencias
- *-l*: mostrar sólo el nombre del fichero cuyas lineas contienen el texto

- *grep*: expresiones regulares
- *egrep* o *grep -E*: expresiones regulares extendidas
- *fgrep* o *grep -F*: cadenas de texto

Numerar/cuantificar el contenido

seq (*sequence*):

Generar una secuencia de números

```
$ seq 1 3↲ # seq 3 (por defecto desde el 1)
1
2
3
```

Opciones interesantes:

- *-w*: todos los números de igual longitud (ceros a la izquierda)
- *-s*: carácter de separación (salto de línea por defecto)

nl (*number lines*):

Mostrar las líneas de un fichero numeradas

```
$ nl lines.txt↲
1  line l1
2  line l2
...
```

wc (*word count*): Mostrar el número de líneas, palabras y caracteres

```
$ wc lines.txt↲ # número de líneas, palabras y caracteres
4 8 40 lines.txt
$ wc -l lines.txt↲ # solo número de lineas
4 lines.txt
$ cat lines.txt | wc -l↲ # mediante pipe
4
```

Comparar ficheros

diff : Comparar ficheros y mostrar las diferencias

sdiff : Comparar dos ficheros y mostrar las diferencias en paralelo y línea por línea

Edición

- *nano*, *pico*
- *vim*, *emacs*
- *gedit*, *kate* ...

Dividiendo y uniendo ficheros

split : División de un fichero en varias partes

Ejemplo: dividir el fichero *file1* en ficheros de 1 KB de tamaño (con el nombre *file1.partX*, donde X es el número de fragmento)

```
split -b 1K -d file1 file1.part     # division del fichero
cat file1.part* > file1-again       # unir las diferentes partes
```

Para comprobar que el fichero original (*file1*) y el obtenido (*file1-again*) son el mismo, se puede utilizar *md5sum*

md5sum : Cálculo del *checksum* de un fichero para más adelante comprobarlo

```
md5sum -b file1 > file1.md5 # creación del checksum
md5sum -c file1.md5sum    # comprobación del checksum
```

6.4. *Propietarios y permisos de los ficheros*

Propiedad del fichero

chown (change owner) : Cambiar el propietario del fichero

```
chown user1 lines.txt                # el nuevo propietario del "lines1.txt" es "user1"
chown user1:group_user1 lines.txt    # lo anterior + cambiar también el grupo propietario
chown -R user1 dir1 # cambiar el propietario del directorio y de su contenido ("-R")
```

chgrp (change group) : Cambiar el grupo propietario del fichero

Permisos: ACL

Lista ACL simplificada

- Usuarios:
  - u: *user* (propietario del fichero)
  - g: *group* (grupo propietario del fichero)
  - o: *others* (resto del mundo)
- Acciones:
  - r: *read*
  - w: *write*
  - x: *execute*

Permisos: bits especiales

Significado especial, en función del tipo de archivo

- *setuid*
- *setgid*
- *sticky*

Gestión de permisos

chmod (change mode) Modificar los permisos de un fichero:

```
chmod u=rw,g=r,o= lines.txt # asignación de permiso por tipo de usuario
chmod a=rwx lines.txt             # o "ugo=rwx"
chmod u+w,a-x lines.txt      # modificar permisos respecto a los anteriores

                             # utilizando sistema octal: r=4, w=2, x=1
chmod 640 lines.txt      # propietario "r" y "w", grupo "r" y el resto nada

chmod u+s,g+s,o+t lines.txt # activar todos los bits especiales
```

Permisos por defecto

umask : Para visualizar/configurar los permisos por defecto de los ficheros que se crean

```
$ umask↲ # consultar los permisos por defecto
0022          # bits especiales – user – group – others
              # los ficheros creados tendrán los siguientes permisos
              # 0022 . Mascara => 7755 => rwx/rwx/r–x/r–x
              # Los bits especiales y los de ejecución no se activan nunca por defecto:
              # rwx/rwx/r–x/r–x => ———/rw–/r––/r––
$ umask 0077↲      # modificar los permisos por defecto. Ejemplo: solo user tendrá permisos
```

6.5. Propiedades de los ficheros

Datos asociados a un fichero

stat : Datos asociados al estado y características de un fichero (tamaño, fechas . . .)

```
$ stat lines.txt↲
File: <lines.txt>
Size: 50              Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d      Inode: 1056768     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/   user1)   Gid: ( 1000/   user1)
Access: 2007-07-11 18:18:28.000000000 +0200
Modify: 2007-07-11 19:27:32.000000000 +0200
Change: 2007-07-11 19:27:32.000000000 +0200
$ stat lines.txt -c %s↲ # mostrar datos individuales
0       # Para más opciones, consultar man
```

Datos asociados a un fichero

file : Mostrar el tipo de fichero

```
$ file lines.txt↲
lines.txt: ASCII text
```

Fechas de un fichero

touch : Modificar la información relativa al tiempo asociada a un archivo

- Si el fichero no existe previamente, crea uno vacío
- Modificar las fechas asociadas a un fichero (*created*, *modified*, *accessed*)
  - *ctime* (*change time*): última vez que se modificó el i-nodo asociado al fichero
  - *mtime* (*modification time*)
  - *atime* (*access time*)

```
touch new.txt    # actualizar la fecha y hora actual en los 3 campos de fecha
                                # o indicar '−m' (modified), '−a− (\emph{accessed})
touch -t 200701011200 lines.txt # utilizar 01/01/2007 12:00
```

Propiedades especiales

chattr (change attribute) : Modificar propiedades avanzadas del fichero (sistema de ficheros *ext2fs*)

lsattr (list attributes) : Mostrar la configuración de las propiedades avanzadas

```
$ lsattr lines.txt↲
------------------- links.txt
# chattr +i lines.txt↲
$ lsattr lines.txt↲
----i-------------- links.txt
# chattr -i lines.txt↲
```

Duplicar la salida

tee : duplicar la salida *stdout*

Se puede enviar a un fichero y a la salida estándar al mismo tiempo

```
who | tee -a log-file | wc -l # −a== append
```

Búsquedas

find I : Búsqueda de ficheros por criterios

- Utilización: *find /path -criterios [acción]*
- Path: ubicación en el sistema de ficheros virtual a partir de la cuál comenzar la búsqueda (también subdirectorios)
- Criterios: según características de los ficheros (nombre, tamaño, tipo, permisos . . .)
- Acción: por defecto listar archivos

find: Criterios de búsqueda

- Nombre: `-name "nombre.txt"`, (*case sensitive*⇒*-iname*)
- Tamaño: *-size*
- Fecha: *-mtime*
- Tipo: *-type*
- Permisos: *-perm*
- . . .

Operadores entre criterios: *!*, *-a*, *-o* (como en *test*)

Ejemplo: buscar ficheros mayores de 100K y de tipo regular:

```
find /home/user1 -size +100K -type f
```

find: operaciones

- Por defecto, listado simple de archivos encontrados
- Listado detallado: *-ls*, *-printf*
- Otras acciones: *-exec comando '{}' \;*

```
# Borrar todos los archivos de la cuenta con extensión 'bak'
 find $HOME -iname "*.bak" -exec rm '{}' \;
```

find: ejemplos

```
find $HOME -perm -002 -print
find $HOME \( -perm 600 -o -perm 700 \) -ls
find /bin \( -perm -4000 -o -perm -2000 -o -perm -1000\) -ls
find . -name "*.c" -ls      # listados extendido
find / -name core -exec rm {} \;
find $HOME -perm -002 -ls
find ~ -name "*.txt" -size -100k -ls     # combinación de criterios
find /tmp -mtime +7 -print >tmp      # guardar los resultados para posterior proceso
find kat -exec grep "Linux" {} \;
```

Búsquedas mediante índice

locate, slocate : Búsqueda de ficheros por nombre utilizando índice

Necesita indexación previa y periódica

updatedb : Indexar ficheros

Para después poder utilizar *locate*

Ubicación de programas

type : Para averiguar la ubicación de un programa

which : Visualizar la ubicación de un programa

whereis : Visualizar la ubicación de ficheros asociados a un programa

Varios

cut : Estructurar el contenido de un fichero en columnas

```
# mostrar las columnas 1 y 5 (fields 1 and 5). El separador es ':'
cut -f "1 5" -d ":" /etc/passwd
```

Nota: el comando *cut* para dividir en columnas espera una única ocurrencia del carácter que separa. Se recomienda un preprocesado para eliminar las posibles repeteciones consecutivas de dicho carácter.

```
df | tail --lines +2 | tr -s ' ' | cut -d' ' -f2
df | tail --lines +2 | sed 's/[[:blank:]]\{1,\}/ /g' | cut -d' ' -f2
```

paste : Unir columnas situadas en distintos ficheros

```
paste file1 file2 # por defecto los separadores son tabuladores
```

join : Realizar entre ficheros la operación *merge*

De manera similar a la misma operación en bases de datos

```
join file1 file2
```

sort : Ordenar alfabéticamente las líneas de un fichero

```
# Ordenar los usuarios por su id (columna 3 y separador de ':')
sort -t ":" -n -k 3 /etc/passwd
```

uniq : Eliminar líneas repetidas

- Eliminar las líneas repetidas
- Es necesario que sean consecutivas ⇒ combinar con *sort*

```
sort file1 | uniq -c # lineas + numero de ocurrencias
```

tr : Sustituir/borrar caracteres

- *-d* Borrar los caracteres indicados
- *-s* Eliminar los caracteres repetidos entre los indicados
- *-c* Convertir

```
echo 'Hola!' | tr '[:lower:]' '[:upper:]' # tr a-z A-Z
tr -d '\015' < file.dos > file.unix # borrar el carácter CR
echo $PATH | tr : '\n'
```

expand/unexpand : Sustituir tabuladores por espacios y viceversa

6.6. Sistema de ficheros y disco

Estructura

fdisk : Estructurar las particiones de un disco

Opciones del comando (se obtienen con la opción *m*)

Tipos de particiones:

- Primaria (*primary*)
- Extendida (*extended*)
- Lógica (*logical*)

```
fdisk
Orden  Acción
   p   Imprime la tabla de particiones
   n   Añade una nueva partición
   d   Suprime una partición
   w   Escribe la tabla en el disco y sale
   q   Sale sin guardar los cambios
```

Estructura

mkfs (make filesystem) : Dar el formato de un sistema de ficheros a un dispositivo o partición

```
mkfs -t vfat /dev/sdb5
```

fsck (file system check) : Comprobar que el sistema de ficheros no tiene errores o inconsistencias

El dispositivo deberá estar sin montar

```
fsck -t ext3 /dev/sdb5
```

Gestionando el disco

mount : Montar una unidad en el sistema de ficheros virtual

```
# Montar la partición 1 del disco 2 en /media/disk
# El sistema de ficheros es ext3
mount /dev/sdb1 /media/disk -t ext3
# Montar la partición 5 del disco 2.
# La información que falta se buscara en /etc/fstab
mount /dev/sdb5
# Como el anterior
mount /media/disk2
# Montar un archivo de tipo iso
mount -o loop  file.iso /media/disk-iso
```

umount : Desmontar una unidad

```
umount /dev/dispositivo # o
umount /path-mounted-drive
```

Gestionando el espacio

df (disk free) : Utilización del espacio de las unidades montadas

```
$ df -h↲
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             2.8G  1.8G  867M  68% /
tmpfs                 126M     0  126M   0% /lib/init/rw
udev                   10M   10M     0 100% /dev
tmpfs                 126M     0  126M   0% /dev/shm
```

du (disk usage) : Espacio estimado que ocupa el directorio y su contenido

```
$ du -sh /home/user/↲ # s: summarize
1.3M    /home/user/
```

Gestión: contenido

sync : Vaciar (sincronizar) la *caché* de disco

dd : Copiar ficheros a bajo nivel

Por ejemplo, para copiar el contenido de un dispositivo en modo *raw*. También se utiliza para conversiones (ASCII, EBCDIC . . .)

```
# Crear una imagen iso de un cd
dd if=/dev/cdrom of=file.iso
```

6.7. Backup

Compresión I

gzip : Comprimir ficheros uno a uno (*GNU Zip*)

No comprime los contenidos de un directorio, por lo que previamente hay que unificarlos en un archivo ⇒ comando *tar*

```
gzip file   # crea el fichero file.gz, borrando el anterior
gzip -9 file   # mas compresión (−−fast <=> −−best)
gunzip file.gz   # recuperar el fichero original gunzip = gzip −d
```

bzip2 : Como *gzip*, pero utilizando un mejor algoritmo de compresión

```
bzip2 file   # crea file.bz2, borrando el fichero anterior
bzip2 -9 file   # mayor compresión
bunzip2 file.bz2   # recuperar el fichero original bunzip2 = bzip2 −d
```

zip : Compresión de ficheros

- También disponible en otros sistemas operativos

- También directorios y su contenido

```
zip -r file-compr.zip *   # comprimir todos los ficheros (*) del dir. de trabajo
        # incluyendo subdir. (-r) y guardar en el fichero file-compr.zip
        # Los ficheros origen no se borran
unzip file-compr.zip
```

Compresión: comando *tar*

- Objetivos:
  - Copias de seguridad
  - Intercambio de información
- Formato:

```
tar opciones disp/file.tar directorio
```

- Tipos: *.tar.gz* y *.tgz*

Comando *tar* komandoa: utilización

- Opciones:
  - *c*: crear
  - *x*: extraer
  - *f*: del/al fichero
  - *z/j*: compresión
  - *t*: mostrar el contenido
- Operaciones:
  - Guardar: *tar czf file* o *tar cz /dev/xxx*
  - Extraer: *tar xzf* o *tar xz*

```
tar cvzf dir1.tar.gz dir1    # Unir y comprimir
tar xzvf dir1.tar.gz         # Descomprimir y extraer el contenido
```

- Incremental:

```
find / -newer /etc/ctrl ! -type d | tar czvfT file.tgz -
```

Sincronización

rsync :

- Imagen de un conjunto de ficheros (en remoto)
- Backup remoto
- Replicación de servicios

6.8. Gestión de procesos

Visualizar procesos

ps : Listar procesos

```
ps       # Procesos asociados al terminal
ps -aux  # Procesos en ejecución en el sistema
```

pidof : Obtener el *pid* de un proceso

```
$ pidof bash↵
10590
```

pstree : Listado de procesos en árbol (proceso padre/hijo)

```
$ pstree↵
init-+-NetworkManager---{NetworkManager}
     |-NetworkManagerD
     |-gdm---gdm-+-Xorg
     |           `-x-session-manag---ssh-agent
     |-gnome-terminal-+-bash---pstree
     |                |-gnome-pty-helpe
     |-gnome-settings----{gnome-settings-}
     |-gnome-terminal-+-bash---sh---bash---sh---pstree
     |                |-gnome-pty-helpe
     |                `-{gnome-terminal}
...
```

Procesos y su utilización del sistema

top : Mostrar información sobre el estado del sistema, procesos en ejecución y utilización de los recursos del sistema de modo interactivo

Teclear *q* (*quit*) para salir

free : Información sobre el consumo de memoria

vmstat : Información sobre la memoria virtual

Terminar procesos

kill : Terminar un proceso a partir de su *pid* (mediante envío de *señales*)

xkill : Terminar un proceso gráfico

killall : Terminar todos los procesos con el mismo nombre

Prioridad entre procesos

nice : Establecer la prioridad de un proceso

renice : Modificar la prioridad de un proceso

6.9. Programación de tareas

Schedulling jobs

sleep : Esperar un tiempo

watch : Ejecutar un programa a intervalos de tiempo

```
watch -n 1 'ps -aux'     # similar a top
```

at : Programar la ejecución de un programa

crontab : Programar la ejecución periódica de una tarea

6.10. Varios

Varios

time : Medir el tiempo de ejecución de un proceso

```
$ time du -sh /home/user > result.txt↵
real   0m12.322s
user   0m0.044s
sys    0m0.592s

$ time cat↵  # poner en marcha un cronometro. Ctrl+D para terminar
```

alias : Para poner alias a comandos (incluyendo opciones)

date : Obtener la fecha del sistema o de un fichero

```
date "+%Y-%m-%d %H:%M"
date "+%Y-%m-%d %H:%M" -r file1
```

Modificar el tamaño de un fichero

truncate : Reducir o aumentar el tamaño de un fichero

```
$ truncate -s 1K file.txt↵ # establecer el tamaño del fichero en 1K
```

Si el tamaño del fichero *file.txt* es:

- Menor: introduce ceros hasta alcanzar el tamaño indicado
- Mayor: elimina los datos del fichero a partir del tamaño indicado

Crear/vaciar un fichero:

```
touch file.txt
truncate -s0 file.txt
:> file.txt
> file.txt
cat /dev/null > file.txt
```

7. Bibliografía

Índice alfabético