INTRODUCCIÓN A BATCH

Un archivo Batch es un archivo de procesamiento por lotes. Se trata de archivos de texto sin formato, guardados por la extensión .bat los cuales contienen un conjunto de instrucciones.

Así cuando se ejecuta un archivo .bat las ordenes contenidas son ejecutadas en grupo, de forma secuencial, permitiendo automatizar diversas tareas.

Es una forma de automatizar procesos como por ejemplo copiar, pegar, renombrar datos y enviar datos. Tiene la funcionalidad de conectarse con otras interfaces por linea de comandos.

Cuando es iniciado un archivo .bat un programa shell o terminal lo lee y lo ejecuta generalmente linea a linea. De este modo se emplea para ejecutar series de comandos automáticamente.

Ordenes de procesamiento por lotes:

-Call: Esta orden realiza, desde un archivo .bat una llamada a la ejecución de otro archivo del mismo tipo. El primer archivo no finaliza su ejecución al realizar la llamada.

Sintaxis: CALL [unidad_disco:][path]nombre_archivo [parámetros]

Donde: [unidad_disco:][path]nombre_archivo: es la ruta al archivo batch llamado (debe ser un archivo *.bat)

[parámetros] puede ser cualquier tipo de información que necesite el programa llamado que pueda pasarse en línea de comandos.

-Choise: ofrece a los usuario una entrada de datos para que pueda escoger una opción y espera hasta que este tenga lugar (análogo al cin en c++ o input de python)

Donde: /C[:]letra, especifica las letras que indicarán las opciones del usuario. Si las específicas separadas por comas, aparecerán entre corchetes seguidas de un interrogante. Si no se especifican, se usara YN (si, no) por defecto.

/N: evita que se muestre el prompt de usuario.

/S: hace que discrimine entre entrada de letras mayúsculas o minúsculas.

/T[:]c, nn: introduce un tiempo de espera antes de ejecutar la acción por defecto. Con los siguientes posibles valores: c: señala que carácter será usado como opción por defecto después de nn segundos. Solo puedes indicar uno de los caracteres especificados con el modificador /C. nn: indica el número de segundos de pausa. Valores entre 0 y 99.

Sintaxis: CHOICE [/C[:]letra] [/N] [/S] [/T[:]c, nn] [texto], puedes especificar un texto que aparecerá antes de la entrada de datos. No hace falta que se entre comillas, salvo que dicha cadena de texto incluya una barra /.

-Echo: esta orden imprime en la consola (análogo al print en python o cout en c++)

Sintaxis: ECHO [ON|OFF] echo [mensaje] Parámetros

Donde: ON|OFF: activa o desactiva el volcado de texto. Para conocer el estado actual, usar ECHO sin parámetros.
Mensaje: el texto a mostrar. echo.

-***Set:*** Da valor a una variable general o de entorno

Sintaxis: set nombre=jake

Donde: set /a: Da valor a una variable utilizando operaciones aritméticas
Ejemplo: set /a número=2+2 Lo que devolvería el valor 4

Ademas: set /p permite la entrada de datos por parte del usuario (prompt), el valor introducido por el mismo define la variable.
Ejemplo: set /p nombre=Introduce tu nombre Esto definiría la variable %nombre% con el valor introducido por el usuario.
Nota: Todo lo que vaya después del signo de = en un set /p se verá en pantalla.

-For: Ejecuta una instrucción sobre un grupo de archivos. Puede utilizarse en la linea de comandos o en un archivo bat o batch:
a) en archivos BAT: FOR %%variable IN (set) DO command [command-parameters]
b) en línea de comandos: FOR %variable IN (set) DO command [command-parameters]

Parámetros:

%%variable o %variable: representa una variable que será reemplazada por su valor. FOR reemplazará %%variable o %variable% con la cadena de caracteres especificados en SET hasta que la instrucción especificada se haya ejecutado sobre todos los archivos. %%variable se emplea con FOR dentro de archivos batch, y %variable desde la línea de comandos.
(set), especifica uno o más archivos de texto (o cadenas) que se procesaran con el comando. Necesita paréntesis.
command, la orden que debe ejecutarse sobre cada archivo especificado en SET.
[command-parameters], parámetros de la instrucción. Podemos emplear la orden son cualquiera de sus parámetros habituales.

-Goto: Salta la ejecución del programa hacia la línea indicada

Sintaxis: GOTO :label

Donde: Label (advierte los dos puntos) es una etiqueta que identifica la línea. 8 caracteres máximo.

-If: Permite la ejecución condicional, es decir, sólo se ejecuta la orden si cumple con la condición introducida.

Sintaxis: IF [NOT] ERRORLEVEL número comando

IF [NOT] string1==string2 comando

IF [NOT] EXIST nombre_archivo comando

Parámetros:

NOT: La instrucción sujeta a condición se ejecuta solo si esta es falsa.
ERRORLEVEL número: la condición es verdadera sólo si la orden anterior devuelve un codigo de salida igual o mayor que el especificado.
command: especifica la orden a ejecutar si la condición se cumple.
string1==string2: La condición se cumple solo si cadena1 es igual a cadena2. Cadena1 y cadena2 pueden ser cadenas de texto o variables.
EXIST nombre_archivo: la condición se cumple si existe nombre_archivo.

-Pause: Suspende la ejecución de instrucciones y presenta un mensaje para que el usuario presione una tecla para continuar.

pause > nul no muestra mensaje al pausar la ejecución, pero sigue requiriendo que el usuario presione cualquier tecla para poder continuar.

-Rem: Línea de comentario.

Sintaxis: REM , :: y % comentario %

-Shift: Alterna la posición de los parámetros en el archivo bat.

Parámetros pasados al script batch

El signo % seguido de un número (del 1 al 8) son los parámetros que se pasaron al invocar nuestro archivo bat. Por ejemplo si tenemos un archivo saludo.bat con la línea echo Hola %1, si lo invocamos tecleando saludo.bat pepe presentará en pantalla Hola Pepe. Se usa por ejemplo para pasar nombres de archivos a un bat que se encarga de borrarlos, moverlos, etc.

El signo %0 representa el nombre del archivo bat en ejecución. El signo %* representa todos los parámetros que se le pasaron.

Para crear nuestro propio batch, es tan fácil como ir al a un editor de texto plano, escribir los comandos/códigos deseados, y al terminar hay que guardar el archivo con el nombre que queramos acabado en .bat.

Ojo. En Linux el archivo se guarda como .sh

Es fundamental e indispensable respetar siempre los espacios, caracteres, comandos, etc, a la hora de escribir cualquier línea de comando/código por simple o básica que sea.

El compilador del archivo .bat es la terminal o shell.

Cuando hayas abierto tu editor escribe el siguiente código para el caso de Linux:

#!/bin/bash

Esta línea avisa a Linux de que es un programa de tipo Bash y por tanto deberá interpretar los comandos de dicho programa. Escrito el código, guárdalo pulsando en Archivo, Guardar y con el nombre holamundo.sh. La extensión sh es reconocida por muchos editores y marcará con colores las palabras clave.

Automatiza tu programa

Ahora que has guardado el 'script' es hora de ejecutarlo cada vez que quieras realizar todas las tareas programadas en dicho archivo. Esto te permitirá automatizar tus tareas evitándote la tediosa tarea de repetir una y otra vez la serie de comandos que has determinado.

El primer paso que tienes que hacer es cambiar los permisos de tu archivo y hacerlos ejecutables. Para ello escribe el comando chmod u+x holamundo.sh.

Este comando transforma el fichero de tal manera que puedas ejecutar el 'script' fácilmente. Para iniciar el 'script' simplemente escribe el nombre del fichero precedido de ./ para el ejemplo anterior sería ./holamundo.sh.

Como puedes observar, se reproduce el mismo programa que has iniciado antes. Esto te va a permitir escribir en el archivo 'holamundo.sh' todos los comandos que necesites evitando que lo ejecutes uno por uno.

Características de Bash

Cuando has aprendido cómo ejecutar y crear 'scripts' tienes que conocer las características de Bash. Esto te permitirá tener un control sobre los comandos y poder personalizarlos como desees para poder automatizar tareas en Linux.

Comentarios

Los comentarios en Bash se hacen línea a línea y con el símbolo # al principio de ella. Esto puede ser útil cuando tu programa es muy grande y quieras poner explicaciones sobre qué trata cada parte.

Un ejemplo sería el siguiente código:

#!/bin/bash

#

Hola Mundo comentado

echo "Hola mundo"

Variables

En programación, las variables son estructuras de datos que, como su nombre indica, pueden cambiar de contenido a lo largo de la ejecución de un programa. Las variables en Bash no tienen tipo, es decir, una variable puede contener una cadena o un número sin necesidad de que lo definas.

La sintaxis es:

nombre_variable=valor_variable

Es obligatorio que no dejes espacios antes o después del símbolo ‘=’ ya que sino Bash interpretaría la variable como un comando Linux.

Para acceder a una variable simplemente escribe como prefijo $ en el nombre de la variable. echo $varname

Prueba con este ejemplo sencillo:

#!/bin/bash

Asignación y salida de variables

mivariable=”Me llamo Nacho”

echo $mivariable

Como puedes observar la línea echo $mivariable ejecutará todo lo contenido entre comillas de la línea anterior.

Paso de variables

Al ejecutar el ‘script’ desde tu terminal o consola puedes pasarle más argumentos. Por ejemplo:

./miScript.sh hola 4

Para recoger estos valores, hola y 4, escribe $ y a continuación el número de posición del argumento pasado. El primer argumento tiene valor $1, que sería ‘hola’, y el segundo sería $2, en el ejemplo sería el número ‘4’. La variable $0 es el nombre del archivo.

Por ejemplo:

#!/bin/bash

#

Paso de variables

#

echo “Tu primer argumento es” $1

echo “Tu segundo argumento es” $2

Hay que destacar que $? guarda el valor de salida del último comando ejecutado, $* almacena todos los argumentos y $# es el número de argumentos pasados.

Comandos Linux

Existe una gran cantidad de comandos ya sean para ayuda, para manejo de archivos y directorios, para manejo de usuarios, de procesos, de disco, de sistema, de red, de impresoras, etc. Esta es una lista de algunos de ellos.

Comandos de ayuda

Los siguientes comandos te proporcionan información y ayuda sobre los diferentes comandos.

Man: muestra el manual del comando que le indiquemos.

El comando—help: muestra una ayuda de los comandos.

Comandos de archivos y directorios

Hay otro tipo de órdenes que te permiten moverte y administrar los archivos y directorios de tu sistema Linux. Aquí te mostramos algunos de ellos:

-ls: lista los archivos y directorios.

-cd:cambia de directorio.

-pwd: muestra la ruta al directorio actual.

-mkdir: crea un directorio.

-rmdir: borra un directorio.

-rm -r: borra directorios no vacíos.

-cp: copia archivos.

-rm: borra archivos.

-mv: mueve o renombra archivos y directorios.

Algo mas…

Variables

Como cualquier otro lenguaje de programación, necesitamos variables que nos servirán para guardar datos en la memoria del ordenador hasta el momento que los necesitemos. Podemos pensar en una variable como una caja en la que podemos guardar un elemento (e.g, un número, una cadena de texto, la dirección de un archivo…) y, siguiendo con el símil, la memoria del ordenador no sería más que el conjunto de esas cajas.

Para asignar el valor a una variable simplemente simplemente debemos usar el signo `=` :

nombre_variable=valor_variable

Es importante no dejar espacios ni antes ni después del `=` .

Para recuperar el valor de dicha variable sólo hay que anteponer el símbolo de dolar `$` antes del nombre de la variable:

$nombre_variable

Nombre de las variables

Las variables pueden tomar prácticamente cualquier nombre, sin embargo, existen algunas restricciones:

•Sólo puede contener caracteres alfanuméricos y guiones bajos

•El primer carácter debe ser una letra del alfabeto o "_" (este último caso se suele reservar para casos especiales).

•No pueden contener espacios.

•Las mayúsculas y las minúsculas importan, "a" es distinto de "A".

•Algunos nombres son usado como variables de entorno y no los debemos utilizar para evitar sobrescribirlas (e.g.,PATH).

De manera general, y para evitar problemas con las variables de entorno que siempre están escritas en mayúscula, deberemos escribir el nombre de las variables en minúscula.

Funciones en batch

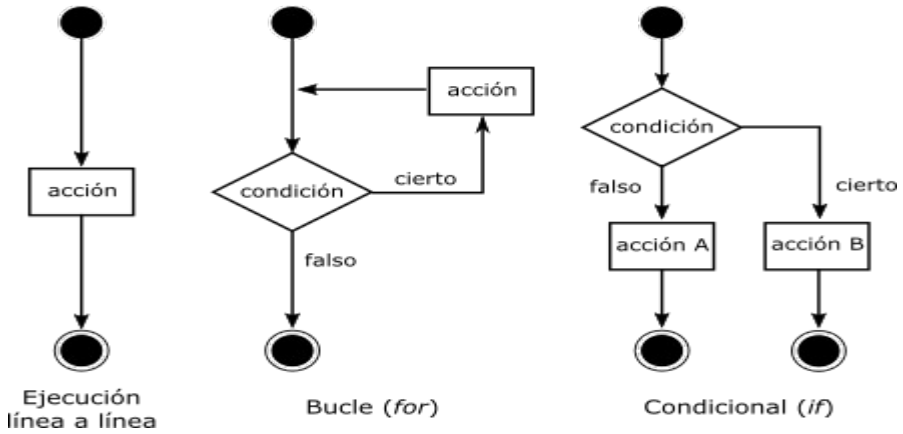
```
function functionName
{
        first command
        second command
return
}
```

Las funciones bash siempre devuelven un único valor.

La entrada interactiva se toma usando **read** para variables.

Control de flujo

Como hemos visto los scripts se ejecutan línea a línea hasta llegar al final, sin embargo, muchas veces nos interesará modificar ese comportamiento de manera que el programa pueda responder de un modo u otro dependiendo de las circunstancias o pueda repetir trozos de código.



Bucles (for)

La sintaxis general de los bucles es la siguiente:

```
for VARIABLE in LISTA_VALORES;
do #Ojo siempre iniciar instrucciones con do
COMANDO 1
COMANDO 2
...
COMANDO N
done #Ojo siempre cerrar for con done
```

Donde la lista de valores puede ser un rango numérico:

```
for VARIABLE in 1 2 3 4 5 6 7 8 9 10;
for VARIABLE in {1..10};
```

una serie de valores:

```
for VARIABLE in file1 file2 file3;
```

o el resultado de la ejecución de un comando:

```
for VARIABLE in $(ls /bin | grep -E 'c.[aeiou]');
```

Hay que tener en cuenta que si pasamos un listado de valores pero lo ponemos entrecomillado, el ordenador lo enterá como un única línea:

```
for VARIABLE in "file1 file2 file3";
```

Bucles (while)

```
while [condicion]
do
        instrucciones
done
```

Tambien se puede utilizar break, continue.

En Bash **&&** representa el elemento lógico AND, mientras que **||** representa a OR.

Condicionales (if)

La sintaxis básica de un condicional es la siguiente

```
if [[ CONDICIÓN ]];
then
  COMANDO 1 si se cumple la condición
fi
```

También se puede especificar qué hacer si la condición no se cumple:

```
  if [[ CONDICIÓN ]];
then #Ojo siempre iniciar instrucciones con then
  COMANDO 1 si se cumple la condición
else
  COMANDO 2 si no se cumple la condición
fi #Ojo siempre cerrar con fi
```

Incluso se pueden añadir más condiciones concatenando más if:

```
 if [[ CONDICIÓN 1 ]];
 then #Ojo siempre iniciar instrucciones con then
  COMANDO 1 si se cumple la condición 1
 elif [[ CONDICIÓN 2 ]];
 then #Ojo siempre iniciar instrucciones con then
  COMANDO 2 si se cumple la condición 2
 else
  COMANDO 3 si no se cumple la condición 2
 fi #Ojo siempre cerrar if con fi
```

Condicionales con números

Al comparar números podemos realizar las siguientes operaciones:

operador	significado
-lt	menor que (<)
-gt	mayor que (>)
-le	menor o igual que (<=)
-ge	mayor o igual que (>=)
-eq	igual (==)
-ne	no igual (!=)

Condicionales con cadenas de texto

A la hora de comparar cadenas de texto:

operador	significado
=	igual, las dos cadenas de texto son exactamente idénticas
!=	no igual, las cadenas de texto no son exactamente idénticas
<	es menor que (en orden alfabético ASCII)
>	es mayor que (en orden alfabético ASCII)
-n	la cadena no está vacía
-z	la cadena está vacía

También podemos hacer comparaciones haciendo uso de wildcards:

Condicionales con archivos

operador	Devuelve true si
-e name	name existe
-f name	name es un archivo normal (no es un directorio)
-s name	name NO tiene tamaño cero
-d name	name es un directorio
-r name	name tiene permiso de lectura para el user que corre el script
-w name	name tiene permiso de escritura para el user que corre el script
-x name	name tiene permiso de ejecución para el user que corre el script

Por ejemplo, podemos hacer un script que nos informe sobre el contenido de un directorio:

Manipulación de cadenas de texto

Extraer subcadena

Mediante `${cadena:posición:longitud}` podemos extraer una subcadena de otra cadena. Si omitimos :longitud, entonces extraerá todos los caracteres hasta el final de cadena.

Por ejemplo en la cadena `string=abcABC123ABCabc` :

- `echo ${string:0}` : abcABC123ABCabc

- `echo ${string:0:1}` : a (primer caracter)
- `echo ${string:7}` : 23ABCabc
- `echo ${string:7:3}` : 23A (3 caracteres desde posición 7)
- `echo ${string:7:-3}` : 23ABCabc (desde posición 7 hasta el final)
- `echo ${string: -4}` : Cabc (atención al espacio antes del menos)
- `echo ${string: -4:2}` : Ca (atención al espacio antes del menos)

Borrar subcadena

Hay diferentes formas de borrar subcadenas de una cadena:

- `${cadena#subcadena}` : borra la coincidencia más corta de subcadena desde el principio de cadena
- `${cadena##subcadena}` : borra la coincidencia más larga de subcadena desde el principio de cadena

Por ejemplo, en la cadena `string=abcABC123ABCabc` :

- `echo ${string#a*C}` : 123ABCabc
- `echo ${string##a*C}` : abc

Reemplazar subcadena

También existen diferentes formas de reemplazar subcadenas de una cadena:

- `${cadena/buscar/reemplazar}` : Sustituye la primera coincidencia de buscar con reemplazar
- `${cadena//buscar/reemplazar}` : Sustituye todas las coincidencias de buscar con reemplazar

Por ejemplo, en la cadena `string=abcABC123ABCabc` :

- `echo ${string/abc/xyz}` : xyzABC123ABCabc.
- `echo ${string//abc/xyz}` : xyzABC123ABCxyz.

Operaciones aritméticas

Por último, Bash también permite la operaciones aritméticas con número enteros:

- + - : suma, resta

```
~$ $num=10
~$ echo $((num + 2))
```

- ** : potencia

~$ echo $((num ** 2))

- * / % : multiplicación, división, resto (módulo)

~$ echo $((num * 2))

~$ echo $((num / 2))

~$ echo $((num % 2))

- VAR++ VAR– : post-incrementa, post-decrementa

~$ echo $((num++))

~$ echo $num

- ++VAR –VAR : pre-incrementa, pre-decrementa

~$ echo $((++num))

~$ echo $num

Ejercicios

1. Haz un script que cree 40 archivos .txt en una carpeta de tu escritorio (usa touch para crearlos)
2. Haz un script que comprima con gzip sólo los archivos 25 y 29.
3. Escribe un script que cambie la extensión de los ficheros que contengan un 3 en su nombre de .txt a .md.
4. Crea un script que copie todos los archivos (no directorios) /etc a una carpeta de tu escritorio.
5. Prepara un script que cuenta el número de directorios y archivos que hay en /etc
6. Haz un script que devuelva el número de archivos que has guardado

Arreglos en batch

Un arreglo en bash se define entre paréntesis. No hay comas entre los elementos del arreglo.

beatles=('John' 'Paul' 'George' 'Ringo')

Para acceder a un elemento desde un arreglo, usará corchetes ([]).Los arreglos están indexados en 0 en bash.También es necesario utilizar la sintaxis de expansión de parámetros.

Redireccionamiento y Tuberias:

Dispositivos de entrada/salida

Para que un usuario pueda comunicarse con el ordenador es necesario disponer de algún dispositivo de entrada que permita introducir órdenes. Asimismo, para que el usuario pueda aprovechar los resultados de las órdenes que ha introducido, necesita que dichos resultados se le muestren de alguna manera.

Estos canales de comunicación se denominan dispositivos de entrada/salida. En Linux los dispositivos por defecto son los denominados entrada estándar y salida estándar. Si un programa lee de la entrada estándar estará obteniendo datos desde el teclado. Cuando escribe algún resultado a la salida estándar, lo estará haciendo a la pantalla.

Ciertos comandos no utilizan la entrada estándar (teclado) para recibir la información que necesitan para ejecutarse, sino que la reciben como argumento directamente.

El interés del concepto de entrada/salida estándar estriba en que todos los dispositivos son tratados como si fueran ficheros. Por esta razón, a los programas no les importa si están leyendo de un teclado o de un fichero ni tampoco si están escribiendo en la pantalla o, de nuevo, en un fichero.

El comando cat

Para ilustrar la noción de entrada/salida estándar, vamos a introducir un comando denominado cat. Este comando toma la información que se le proporciona de entrada, la une y la vuelve a dar como salida.

Por ejemplo, si tecleamos lo siguiente

cat

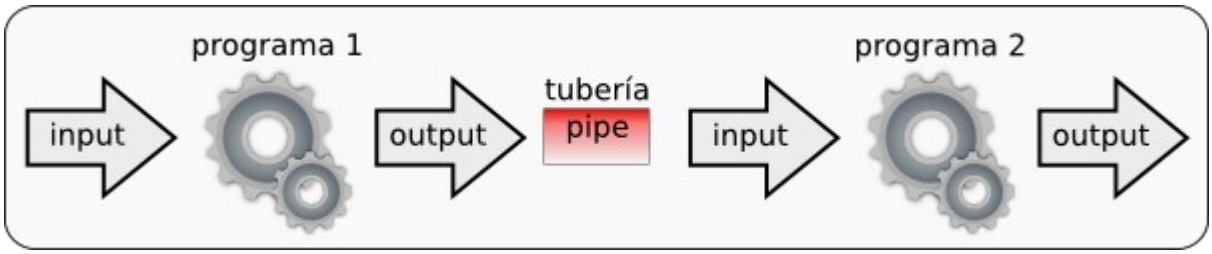
el sistema quedará aparentemente en espera mostrando una línea en blanco. Si tecleamos una frase y pulsamos Intro, vemos que cat muestra la misma frase por pantalla. Pulsando CTRL-D podemos finalizar la ejecución.

cat

Muestra esta frase

Muestra esta frase

Este ejemplo constituye una demostración del uso de la entrada/salida estándar. La línea en blanco es la forma de indicar que cat está esperando algo del teclado (entrada estándar). Cuando lo tiene, realiza su función que es mostrarlo por pantalla (salida estándar).



Conceptos básicos

Antes de pasar explicarles en que consisten las tuberías y su importancia (la parte divertida), debemos tener claros tres conceptos fundamentales en Linux: entrada estándar, salida estándar y error estándar.

La entrada estándar: representa los datos que son necesarios para el correcto funcionamiento de una aplicación. Un ejemplo de ellos puede ser un archivo con datos

estructurados o información ingresada desde la terminal. En la terminal se representa como el tipo 0.

La salida estándar: es el medio que utiliza un aplicación para mostrar información de sus procesos y/o resultados, estos pueden ser simples mensajes, avisos referentes al progreso o archivos con datos estructurados como resolución de un proceso (un reporte, por ejemplo). En la terminal se representa como el tipo 1.

El error estándar: es la manera en que las aplicaciones nos informan sobre los problemas que pueden ocurrir al momento de su ejecución. Se representa como el tipo 2 en la terminal.

Redirecciones

Ahora bien, ¿qué es una redirección?

Las redirecciones consisten en trasladar información de un tipo a otro (los tipos mencionados anteriormente), por ejemplo, de error estándar a la salida estándar o de la salida estándar a la entrada estándar. A través de la terminal, logramos eso haciendo uso del símbolo >.

Redireccionar salida y error estándar

Por ejemplo, para hacer la redirección de la salida de un comando y enviarla a un archivo; nos basta con ejecutar:

ls -la ~ > (nombre del archivo)

Sin embargo, si realizamos la ejecución de esa forma, el contenido de nuestro archivo sera remplazado, cada vez, por la salida del comando. Si lo que quisiéramos es que se adicione dicha salida en el archivo, entonces la ejecución seria de la siguiente manera:

ls -la ~ >> (nombre del archivo)

Lo que resulta interesante es que, podemos redireccionar las salidas, los errores y las entradas estándar. Es aquí donde toman sentido los números que les mencionaba al inicio. Por ejemplo, para forzar que un programa nos muestre en pantalla los errores que se generen durante una ejecución, redireccionamos el error estándar hacia la salida estándar durante su ejecución:

aplicación 2 >> &1

Donde 2, representa al error estándar y &1 representa la salida estándar.

También podemos hacer un descarte del error estándar en cierto proceso, algo común en la administración de sistemas. Para esto ejecutamos:

aplicación 2 > /dev/null

Inclusive descartar la salida estándar:

aplicación > /dev/null

Ya que en Linux, el archivo /dev/null, es un archivo especial a donde se envía la información para que sea descartada.

Redireccionar la entrada

De la misma manera en que redireccionamos salidas y errores estándar, podemos hacerlo con las entradas estándar desde un archivo y para ello utilizamos el operador <.

Esto resulta útil en comandos o programas donde se introducen los argumentos por teclado, del tal forma que podemos sustituirlos por un archivo, por ejemplo:

echo "Hola mundo" > saludo

cat < saludo

Hola mundo

Tuberías

Luego de comprender el funcionamiento de las redirecciones, el concepto de tuberías sera bastante sencillo. Entre los principios dela filosofía de Unix, tenemos el hecho de tener aplicaciones pequeñas que se encarguen de realizar tareas muy puntuales y que en conjunto realicen tareas complejas. Siguiendo este principio, tiene que existir una manera para que un conjunto de aplicaciones pueda interactuar entre si. Es aquí donde surgen las llamadas tuberías.

Las tuberías son un tipo especial de redirección que permiten enviar la salida estándar de un comando como entrada estándar de otro. La forma de representarlo es con el símbolo | (pipe). Su principal utilidad es que nos ofrece la posibilidad de concatenar comandos, enriqueciendo la programación.

Un ejemplo sencillo y muy útil es ver los procesos que están ejecutando en el sistema con ps y redireccionar su salida a sort para ordenarlos por PID:

ps -a | sort

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

Como pueden ver, las redirecciones y las tuberías son conceptos de Linux fundamentales y que sin duda debemos manejar. De este modo te sentirás cada vez mas cómodo en la terminal.

Direcciones:

El sistema de archivos controla como se almacenan los archivos en el ordenador. Sus dos tareas principales son guardar y leer archivos previamente guardados.

Sistemas jerárquicos

Los sistemas de archivos suelen tener directorios en los que organizar los archivos y estos directorios suelen estar organizados jerárquicamente. La jerarquía implica que un directorio puede contener subdirectorios. El directorio más alto en la jerarquía del que cuelgan todos los demás se denomina raíz (root). En los sistemas Unix el directorio raíz se representa con una barra “/” y sólo existe una jerarquía, es decir, sólo existe un directorio raíz, incluso aunque haya distintos discos duros en el ordenador.

Dentro del directorio raíz podemos encontrar diversos subdirectorios, por ejemplo en Linux existe el directorio home. Home es por tanto un subdirectorio del directorio raíz. Esta relación se representa como:

/home

home es el directorio dónde se encuentran los directorios de los usuarios en un sistema Linux. Imaginemos que tiene los subdirectorios alicia y juan. Se representaría como:

/home/alicia

/home/juan

Existe un estándar, denominado Filesystem Hierarchy Standard que define la estructura de directorios de los sistemas Unix. Los sistemas Unix suelen seguir este estándar, aunque a veces lo violan en algunos aspectos. Por ejemplo en MacOS X el directorio donde se encuentran los directorios de los usuarios se denomina Users y no home

En algunos sistemas operativos no UNIX la barra se escribe al revés “", a pesar de que la convención siempre fue la contraria.

En el directorio raíz hay diversos directorios que, en la mayoría de los casos, sólo deberían interesarnos si estamos administrando el ordenador. Los usuarios normalmente sólo escriben dentro de un directorio de su propiedad localizado dentro de /home y denominado como su nombre de usuario.

Los usuarios también pueden escribir en /tmp aunque normalmente son los procesos lanzados por estos lo que hacen esta escritura. Es importante revisar el espacio libre en la partición en la que se encuentra /tmp para que no se colapse el sistema. Recuerda que /tmp es borrado habitualmente por el sistema. Normalmente con cada nuevo arranque.

Rutas absolutas, relativas y directorio de trabajo

Para referirnos a un archivo o a un directorio debemos indicar su ruta (path. Un ejemplo de ruta podría ser:

/home/alicia/documentos/tesis.md

Este tipo de rutas en las que se especifican todos los subdirectorios empezando desde el directorio raíz se denominan rutas absolutas.

Para no tener que escribir la ruta absoluta completa cada vez que queremos referirnos a un archivo o a un directorio se crearon los conceptos de directorio de trabajo y de ruta relativa.

El directorio de trabajo es una propiedad del terminal (del shell) en la que estamos trabajando. Siempre que estemos trabajando en una terminal tendremos asignado un directorio de trabajo. Por ejemplo, si nuestro usuario es alicia sería normal que al abrir un terminal nuestro directorio de trabajo fuese:

/home/alicia

El directorio de trabajo se utiliza para escribir rutas a archivos relativas al mismo. De este modo nos ahorramos escribir bastante. Imaginemos que Alicia tiene en su directorio un documento llamado peliculas.txt. La ruta absoluta sería.

/home/alicia/peliculas.txt

Mientras su directorio de trabajo sea /home/alicia la ruta relativa sería simplemente:

peliculas.txt

Es decir, podemos escribir rutas relativas al directorio de trabajo, rutas que en vez de partir del directorio raíz parten desde el directorio de trabajo. Las rutas relativas se diferencian de las absolutas en los sistemas Unix porque las absolutas empiezan por “y” las relativas no.

Es común referirse al directorio de trabajo de una terminal como a un lugar en el que nos encontramos mientras estamos trabajando en la terminal. Siempre que estemos en una terminal estaremos dentro de un directorio de trabajo.

Por ejemplo, cuando abrimos un nuevo terminal el directorio de trabajo se sitúa en /home/nombre_de_usuario. Si ejecutamos el comando ls, el programa asumirá que queremos listar los archivos presentes en ese directorio y no en otro cualquiera. Existe un comando que nos informa sobre el directorio de trabajo actual, pwd (Print Working Directory):

$ pwd

/home/alicia

Si deseamos podemos modificar el directorio de trabajo “moviéndonos” a otro directorio. Para lograrlo hay que utilizar el comando cd (Change Directory):

$ cd documentos

$ pwd

/home/alicia/documentos

A partir de ese momento los comandos asumirán que si no se les indica lo contrario el directorio desde el que deben trabajar es /home/alicia/documentos.

Cd además tiene algunos parámetros especiales:

cd Ir al directorio $HOME del usuario.

cd - Ir al directorio de trabajo previo

Directorio $HOME

El directorio $HOME en los sistemas Unix, que son sistemas multiusuario, es el directorio en el que el usuario debe mantener sus ficheros y directorios. Fuera de este directorio el usuario tendrá unos permisos restringidos puesto que sus acciones podrían afectar a otros usuarios.

En Linux los directorios $HOME de los usuarios son subdirectorios del directorio /home.

El directorio $HOME de un usuario es además el directorio de trabajo por defecto, es decir, el directorio de trabajo que se establece cuando se abre una terminal.

Moviendo, renombrando y copiando ficheros

En primer lugar vamos a crear un fichero de prueba:

~$ touch data.txt

~$ ls

data.txt

El comando touch, en este caso, ha creado un fichero vacío.

Los ficheros se copian con el comando cp(CoPy):

~$ cp data.txt data.bak.txt

~$ ls

data.bak.txt data.txt

Se mueven y renombran con el mv (MoVe):

~$ mv data.txt experimento_1.txt

~$ ls

data.bak.txt experimento_1.txt

Para crear un nuevo directorio podemos utilizar la orden mkdir (MaKeDIRectory):

~$ mkdir exp_1

~$ ls

data.bak.txt exp_1 experimento_1.txt

mv también sirve para mover ficheros entre directorios:

~$ mv experimento_1.txt exp_1/

~$ ls

data.bak.txt exp_1

~$ ls exp_1/

experimento_1.txt

Los ficheros se eliminan con la orden rm ReMove):

~$ rm data.bak.txt

~$ ls

exp_1

En la línea de comandos de los sistemas Unix cuando se borra un fichero se borra definitivamente, no hay papelera. Una vez ejecutado el rm no podremos recuperar el archivo.

Los comandos cp y rm no funcionarán bien con los directorios a no ser que modifiquemos el comportamiento que muestran por defecto:

~$ rm exp_1/

rm: cannot remove exp_1/ Is a directory

~$ cp exp_1/ exp_1_bak/

cp: omitting directory exp_1/

Esto sucede porque para copiar o borrar un directorio hay que copiar o borrar todos sus contenidos recursiva mente y esto podría alterar muchos datos con un sólo comando. Por esta razón se exige que estos dos comandos incluyan un modificador que les indique que sí deben funcionar recursivamente cuando tratan con directorios:

~$ cp -r exp_1/ exp_1_bak/

~$ ls

exp_1 exp_1_bak

~$ rm -r exp_1_bak/

~$ ls

exp_1

Nombres de directorios y archivos

En Unix los archivos pueden tener prácticamente cualquier nombre. Existe la convención de acabar los nombres con un punto y una pequeña extensión que indica el tipo de archivo. Pero esto es sólo una convención, en realidad podríamos no utilizar este tipo de nomenclatura.

Si deseamos utilizar nombres de archivos que no vayan a causar extraños comportamientos en el futuro lo mejor sería seguir unas cuantas reglas al nombrar un archivo:

- •Añadir una extensión para recordarnos el tipo de archivo, por ejemplo .txt para los archivos de texto.
- •No utilizar en los nombres:
  - •espacios,
  - •caracteres no alfanuméricos,
  - •ni caracteres no ingleses como letras acentuadas o eñes.

Por supuesto, podríamos crear un archivo denominado “$ñ 1.txt” para referirnos a un archivo de sonido, pero esto conllevaría una sería de problemas que aunque son solventables nos dificultarán el trabajo.

Además es importante recordar que en Unix las mayúsculas y las minúsculas no son lo mismo. Los ficheros “documento.txt”, “Documento.txt” y “DOCUMENTO.TXT” son tres ficheros distintos.

Otra convención utilizada en los sistema Unix es la de ocultar los archivos cuyos nombres comienzan por punto “.”. Por ejemplo el archivo “.oculto” no aparecerá normalmente cuando pedimos el listado de un directorio. Esto se utiliza normalmente para guardar archivos de configuración que no suelen ser utilizados directamente por los usuarios. Para listar todos los archivos (All), ya sean éstos ocultos o no se puede ejecutar:

$ ls -a

. .fontconfig .HyperTree .pki

.. fsm.jpg .ICEauthority .recently-used

Esta convención de ocultar los ficheros cuyo nombre comienza por un punto se mantiene también en el navegador gráfico de ficheros. En este caso podemos pedir que se muestren estos archivos en el menú Ver -> Mostrar los archivos ocultos.

Para acelerar el acceso a ciertos directorios existen algunos nombres especiales que son bastante útiles:

* ".." indica el directorio padre del directorio actual

* "." indica el directorio actual

* "~" representa la $HOME del usuario

WildCards

En muchas ocasiones resulta útil tratar los ficheros de un modo conjunto. Por ejemplo, imaginemos que queremos mover todos los ficheros de texto a un directorio y la imágenes a otro. Creemos una pequeña demostración::

~$ touch exp_1a.txt

~$ touch exp_1b.txt

~$ touch exp_1b.jpg

~$ touch exp_1a.jpg

~$ ls

exp_1 exp_1a.jpg exp_1a.txt exp_1b.jpg exp_1b.txt

Podemos referirnos a todos los archivos que acaban en txt utilizando un asterisco:

~$ mv *txt exp_1

~$ ls

exp_1 exp_1a.jpg exp_1b.jpg

El asterisco sustituye a cualquier texto, por lo que al escribir *txt incluimos a cualquier fichero que tenga un nombre cualquiera, pero que termine con las letras txt. Podríamos por ejemplo referirnos a los ficheros del experimento 1a:

~$ ls *1a*

exp_1a.jpg

Esta herramienta es muy potente y útil, pero tenemos que tener cuidado con ella, sobre todo cuando la combinamos con rm. Por ejemplo la orden:

$ rm -r *

Borraría todos los ficheros y directorios que se encuentren bajo el directorio de trabajo actual, si lo hacemos perderemos todos los ficheros y directorios que cuelgan del actual directorio de trabajo, puede que esto sea lo que queramos, pero hemos de andar con cuidado.

Ejercicios

1. ¿Cuáles son los ficheros y directorios presentes en el directorio raíz?
2. ¿Cuáles son todos los archivos presentes en nuestro directorio de usuario?
3. Crea un directorio llamado experimento.
4. Crea con touch los archivos datos1.txt y datos2.txt dentro del directorio experimento.
5. Vuelve al directorio principal de tu usuario y desde allí lista los archivos presentes en el directorio experimento usando rutas absolutas y relativas
6. Haz del directorio ~/Documentos tu directorio de trabajo y repite el ejercicio anterior
7. Borra todos los archivos que contengan un 2 en el directorio experimento.
8. Copia el directorio experimento a un nuevo directorio llamado exp_seguridad.
9. Borra el directorio experimento.
10. Renombra el directorio exp_seguridad a experimento.
11. Copia el fichero /etc/passwd al directorio ~/Documentos
12. Copia el fichero /etc/passwd al directorio ~/Documentos llamándolo usuarios.txt

Obteniendo información sobre archivos y directorios

ls es un comando capaz de mostrarnos información extra sobre los archivos y directorios que lista. Por ejemplo podemos pedirle, usando la opción -l (Long), que nos muestre quién es el dueño del archivo y cuanto ocupa y qué permisos tiene además de otras cosas::

~$ ls

exp_1

~$ ls -l

total 4

drwxr-xr-x 2 usuario usuario 4096 Oct 13 09:48 exp_1

La información sobre la cantidad de disco ocupada la da por defecto en bytes, si la queremos en un formato más inteligible podemos utilizar la opción -h (Human):

~$ ls -lh

total 4.0K

drwxr-xr-x 2 usuario usuario 4.0K Oct 13 09:48 exp_1

Podemos consultar el tipo de un archivo mediante el comando file.

~$ file imagen.png

imagen.png: PNG image data, 1920 x 1080, 8-bit/color RGB, non-interlaced

En principio, el tipo de un archivo no está determinado por la extensión, la extensión es sólo parte del nombre, aunque hay software que viola o complementa este principio. El tipo de archivo está determinado por su magic number. El magic number está compuesto por una corta serie de bytes que indican el tipo de archivo.

Permisos

Unix desde su origen ha sido un sistema multiusuario. Para conseguir que cada usuario pueda trabajar en sus archivos, pero que no pueda interferir accidental o deliberadamente con los archivos de otros usuarios se estableció desde el principio un sistema de permisos. Por defecto un usuario tiene permiso para leer y modificar sus propios archivos y directorios, pero no los de los demás. En los sistemas Unix los ficheros pertenecen a un usuario concreto y existen unos permisos diferenciados para este usuario y para el resto. Además el usuario pertenece a un grupo de trabajo. Por ejemplo, imaginemos que la usuaria alicia puede pertenecer al grupo de trabajo “diagnostico”. Si alicia crea un fichero este tendrá unos permisos diferentes para alicia, para el resto de miembros de su grupo y para el resto de usuarios del ordenador. Podemos ver los permisos asociados a los ficheros utilizando el comando ls con la opción -l (Long)::

~$ ls -l

total 7324

-rw-r--r-- 1 alicia diagnostico 1059 Oct 20 12:42 busqueda_leukemia_100.txt

-rw-r--r-- 1 alicia diagnostico 0 Oct 13 10:53 datos_1.txt

drwxr-xr-x 2 alicia diagnostico 4096 Oct 13 10:29 experimento

En este caso, los ficheros listados pertenecen Alicia y al grupo diagnostico. Los permisos asignados al usuario, a los miembros del grupo y al resto de usuarios están resumidos en la primeras letras de cada línea::

drwxr-x---

La primera letra indica el tipo de fichero listado: (d) directorio, (-) fichero u otro tipo especial. Las siguientes nueve letras muestran, en grupos de tres, los permisos para el usuario, para el grupo y para el resto de usuarios del ordenador. Cada grupo de tres letras indica los permisos de lectura (Read), escritura (Write) y ejecución (eXecute). En el caso anterior el usuario tiene permiso de lectura, escritura y ejecución (rwx), el grupo tiene permiso de lectura y ejecución (r-x), es decir no puede modificar el fichero o el directorio, y el resto de usuarios no tienen ningún permiso (—).

En los ficheros normales el permiso de lectura indica si el fichero puede ser leído, el de escritura si puede ser modificado y el de ejecución si puede ser ejecutado. En el caso de los directorios el de escritura indica si podemos añadir o borrar ficheros del directorio y el de ejecución si podemos listar los contenidos del directorio.

Estos permisos pueden ser modificados con la orden chmod. En chmod cada grupo de usuarios se representa por una letra:

- •u: usuario dueño del fichero
- •g: grupo de usuarios del dueño del fichero
- •o: todos los otros usuarios
- •a: todos los tipos de usuario (dueño, grupo y otros)

Los tipos de permisos también están abreviados por letras:

- •r: lectura
- •w: escritura
- •x: ejecución

Con estas abreviaturas podemos modificar los permisos existentes.

Hacer un fichero ejecutable:

$ chmod u+x

O:

$ chmod a+x

También podemos mediante chmod indicar los permisos para un tipo de usuario determinado.

$ chmod a=rwx

Un modo algo menos intuitivo, pero más útil de utilizar chmod es mediante los números octales que representan los permisos.

- lectura: 4

- escritura: 2

- ejecución: 1

Para modificar los permisos de este modo debemos indicar el número octal que queremos que represente los permisos del fichero. La primera cifra representará al dueño, la segunda al grupo y la tercera al resto de usuarios. Por ejemplo si queremos que único permiso para el dueño y su grupo sea la lectura y que no haya ningún permiso para el resto de usuarios:

$ chmod 110 fichero.txt

También podemos combinar permisos sumando los números anteriores. Por ejemplo, permiso para leer y escribir para el dueño y ningún permiso para el resto.

$ chmod 300 fichero.txt

Permisos de lectura, escritura y ejecución para el dueño y su grupo y ninguno para el resto.

$ chmod 770 fichero.txt

Las restricciones para los permisos no afectan al usuario root, al administrador del sistema. root también puede modificar quien el dueño y el grupo al que pertenecen los ficheros mediante los comando chown y chgrp.

$ chown alicia fichero.txt

$ chown diagnostico fichero.txt

Obteniendo información sobre el sistema de archivos

El sistema de archivos puede abarcar una o más particiones. Una partición es una región de un disco o de cualquier otro medio de almacenamiento. Las instalaciones de Windows tienen normalmente una partición por disco, pero en Linux esto no es tan habitual. Cada partición tiene un sistema de archivos propio, pero en Unix estos sistemas deben estar montados en algún lugar dentro de la jerarquía que cuelga de la raíz. En Windows cada partición tiene por defecto una jerarquía independiente.

Podemos pedir información sobre el espacio ocupado por las distintas particiones que tenemos actualmente montadas usando el comando df (Disk Free).

$ df -h

S.ficheros Tamaño Usados Disp Uso% Montado en

udev 7,8G 0 7,8G 0% /dev

tmpfs 1,6G 9,8M 1,6G 1% /run

/dev/nvme0n1p2 25G 8,1G 16G 35% /

tmpfs 7,8G 5,3M 7,8G 1% /dev/shm

tmpfs 5,0M 4,0K 5,0M 1% /run/lock

tmpfs 7,8G 0 7,8G 0% /sys/fs/cgroup

/dev/nvme0n1p4 206G 18G 178G 9% /home

/dev/nvme0n1p1 511M 3,6M 508M 1% /boot/efi

/dev/sda1 2,7T 117G 2,5T 5% /home/jose/magnet

tmpfs 1,6G 64K 1,6G 1% /run/user/1000

Algunos de los sistemas de archivos montados puede que no se correspondan con particiones en un disco físico sino con espacios de la memoria RAM que son utilizados como sistemas de archivos especiales.

El comando du (disk usage) informa sobre el espacio que ocupa un árbol de directorios. Este comando tiene equivalentes gráficos como Baobab o xdiskusage. Podemos pedir a du que nos muestre cuanto espacio ocupan los directorios bajo el directorio analysis:

$ du -h analyses/

36K analyses/alicia/cache

204K analyses/alicia/differential_snps/differential

252K analyses/alicia/differential_snps/non_differentia

919M analyses/

Si sólo queremos obtener el resultado para el directorio que le hemos dado y no para sus subdirectorios podemos utilizar el parámetro -s:

$ du -sh analyses/

919M analyses/

Si queremos información sobre todos los archivos y no sólo los directorios podemos usar -a:

$ du -ha analyses/

32K analyses/alicia/cache/min_called_rate_samples_cache.pickle

36K analyses/alicia/cache

8,0K analyses/alicia/look_for_matching_accessions.py

Ejercicios

1.¿Cuáles son los permisos de los directorios presentes en el directorio raíz y en nuestro directorio de usuario? ¿A quién pertenecen los ficheros y qué permisos tienen los distintos usuarios del ordenador?

2.Crea un directorio en tu home y muestra los permisos que tiene.

3.Cambia los permisos para que sólo tu usuario pueda acceder al nuevo directorio

4.Crea un fichero nuevo y dale permisos de ejecución para todos los usuarios

5.Último fichero modificado en el directorio /etc.

6.Lista los ficheros de /etc con su tamaño y ordenarlos por tamaño.

7.Copia todos los ficheros y directorios del directorio /etc cuyo nombre comience por s. ¿Has podido copiarlos todos?

8.¿Cuánto espacio libre queda en las distintas particiones del sistema?

9.¿Cuánto espacio ocupan todos los ficheros y subdirectorios de tu $HOME?

Compresión y descompresión de ficheros

Existen distintos formatos de compresión de ficheros como: gzip, bzip, zip o rar. Los formatos más utilizados en Unix son gzip y bzip.

Comprimir un fichero con gzip o bzip:

$ gzip informacion_snps.txt

$ ls

informacion_snps.txt.gz

$ bzip2 accs.txt

$ ls

accs.txt.bz2

bzip2 comprime más que gzip, pero es más lento. gzip también dispone de varios niveles de compresión, cuanto más comprime más lenta suele ser la compresión.

Podemos descomprimir cualquier fichero utilizando la línea de comandos:

$ gunzip informacion_snps.txt.gz

$ ls

informacion_snps.txt

$ bunzip2 accs.txt.bz2

$ ls

accs.txt

Muchos estamos acostumbrados al formato zip. Un fichero zip no se corresponde en realidad con un sólo fichero comprimido sino con varios. Un fichero zip hace dos cosas: unir varios ficheros en uno y comprimir el resultado. Los comandos que hemos visto (gzip y bzip2) son capaces de comprimir un sólo archivo, pero no pueden unir varios archivos en uno. Tar es el comando capaz de unir varios archivos en uno.

$ ls

seq1.fasta seq2.fasta

$ tar -cvf secuencias.tar seq*

seq1.fasta

seq2.fasta

$ ls

secuencias.tar seq1.fasta seq2.fasta

tar también es capaz de desempaquetar los archivos que habíamos unido.

$ ls

secuencias.tar

$ rm seq1.fasta seq2.fasta

$ tar -xvf secuencias.tar

seq1.fasta

seq2.fasta

$ ls

secuencias.tar seq1.fasta seq2.fasta

El problema es que utilizando el comando tar tal y como lo hemos hecho hemos conseguido unir y separar archivos, pero no hemos comprimido el fichero unido. Para hacerlo podríamos utilizar los comandos gzip o bzip2, pero este no es el modo habitual de hacerlo. Dado que casi siempre que unamos archivos en un archivo tar también querremos comprimir el resultado el comando tar tiene también la capacidad de comprimir y descomprimir utilizando los algoritmos gzip y bzip2. Unir y comprimir con gzip varios archivos:

$ tar -cvzf secuencias.tar.gz seq*

seq1.fasta

seq2.fasta

$ ls

secuencias.tar.gz seq1.fasta seq2.fasta

Descomprimir un archivo tar.gz:

$ tar -xvzf secuencias.tar.gz

seq1.fasta

seq2.fasta

$ ls

secuencias.tar.gz seq1.fasta seq2.fasta

También podemos descomprimir el contenido de un fichero de texto y enviar el resultado a la terminal con el comando zcat.

$ zcat fichero.txt.gz

Con bzip2.

$ tar -cvjf secuencias.tar.bz seq*

seq1.fasta

seq2.fasta

$ ls

secuencias.tar.bz seq1.fasta seq2.fasta

$ tar -xvjf secuencias.tar.bz

seq1.fasta

seq2.fasta

Ejercicios

1. Crea un fichero de texto en el directorio ~/Documentos y comprimelo con gzip
2. Muestra el contenido del fichero anterior en pantalla sin descomprimirlo previamente
3. Crea un archivo tar de todo el contenido del directorio ~/Documentos
4. Comprime el fichero tar anterior
5. Vuelve a hacer los ejercicios 2 y 3, pero en un sólo paso
6. Descomprime el fichero tar.gz anterior en un nuevo directorio llamado Documentos2

Enlaces duros y blandos

Podemos pensar en el nombre de un fichero como en una etiqueta que apunta a una posición concreta en el disco duro, en realidad es un puntero a un inodo.

Podemos pensar en un enlace duro como en un nombre adicional para un archivo. Si tenemos un archivo en el disco y creamos un enlace duro tendremos dos nombres para ese único archivo.

$ ls

archivo1.txt

$ ln archivo1.txt nombre2.txt

$ ls

archivo1.txt nombre2.txt

Las dos referencias, nombres, al archivo serán indistinguibles. Si borramos un nombre quedará el otro. Si modificamos un archivo se modifica independientemente del nombre por el cual estemos accediendo a él. No es muy común utilizar enlaces duro salvo en aplicaciones muy concretas, por ejemplo en versiones de copias de seguridad.

Un enlace blando, más comúnmente conocido como un enlace simbólico, es una referencia al nombre de un archivo, no al archivo en sí.

$ ls

archivo1.txt

$ ln -s archivo1.txt nombre3.txt

$ ls -l

-rw-rw-r-- 1 jose jose 0 sep 27 15:16 archivo1.txt

lrwxrwxrwx 1 jose jose 12 sep 27 15:16 nombre3.txt -> archivo1.txt

Si eliminamos el archivo original el enlace quedará roto.

$ rm archivo1.txt

$ cat nombre3.txt

cat: nombre3.txt: No existe el archivo o el directorio

El comportamiento de ambos tipos de enlaces cambia si sobrescribimos el fichero.

x

$ echo "hola" > hola.txt

$ cat hola.txt

hola

$ ln hola.txt hola2.txt

$ ln -s hola.txt hola3.txt

$ ls -l

-rw-rw-r-- 2 jose jose 5 sep 27 15:23 hola2.txt

lrwxrwxrwx 1 jose jose 8 sep 27 15:25 hola3.txt -> hola.txt

-rw-rw-r-- 2 jose jose 5 sep 27 15:23 hola.txt

$ echo "adios" > adios.txt

$ mv adios.txt hola.txt

$ cat hola.txt

adios

$ cat hola2.txt

hola

Los enlaces blandos funcionan incluso entre distintos sistemas de archivos o particiones, los duros no.

Ejercicios

1. Crea un enlace simbólico a un fichero de texto dentro del directorio ~/Documentos
2. Crea un enlace duro al mismo fichero.

3.Edita el fichero de texto y observa como cambian ambos enlaces

4.Crea un nuevo fichero de texto con otro contenido. Sustituye el primer fichero con el segundo y observa el resultado en ambos enlaces

5.Crea dos enlaces, uno simbólico y otro duro, a un fichero. Elimina el fichero y observa el resultado en ambos enlaces

Acceso remoto

Una de las grandes ventajas de utilizar la terminal es que podemos acceder a terminales en otros ordenadores muy fácilmente. El protocolo más utilizado para acceder a terminales de forma remota es ssh (Secure Shell). ssh tiene un gran número de posibilidades, pero el uso más habitual es utilizarlo para abrir terminales en ordenadores remotos que tienen un servicio ssh. ssh es seguro porque cifra las comunicaciones entre el cliente y el servidor. ssh se diseñó como una alternativa segura a telnet. No debemos usar el protocolo telnet porque las comunicaciones en telnet, incluidas las claves de acceso, no están cifradas y cualquiera puede tener acceso a ellas.

Para acceder a una computadora que implemente el protocolo ssh podemos usar el programa ssh, pero previamente tenemos que tener una cuenta en esa computadora. Imaginemos que alicia tiene una cuenta en un ordenador que tiene un servicio ssh. Para conectarse puede hacer:

$ ssh alicia@ordenador.upv.es

Si el nombre de la cuenta de usuario en el ordenador cliente y en el servidor es el mismo puede obviar el nombre de usuario.

$ ssh ordenador.upv.es

A continuación el servidor le pedirá la clave correspondiente a ese usuario.

Existen clientes ssh para windows con los que nos podemos conectar a servidores ssh. Uno muy común es putty.

Una tarea muy habitual cuando estamos trabajando en un ordenador remoto es enviar o traer ficheros desde el mismo. Esto también lo podemos hacer utilizando el protocolo ssh por lo que podremos hacerlo de un modo seguro en cualquier ordenador que no de acceso ssh. El programa más sencillo para hacerlo desde Unix es *scp* (Secure CoPy). Scp tiene una interfaz muy similar a cp pero acepta que los ficheros de origen y destino estén en distintos ordenadores:

$ scp alicia@remotehost.edu:/remote/directory/seq.txt /some/local/directory

$ scp /some/local/directory/seq.txt alicia@remotehost.edu:/remote/directory/

En windows también hay distintos clientes scp, uno de ellos es winscp.

Una alternativa a scp que tiene más capacidades, como enviar fragmentos de ficheros, es rsync. Rsync está diseñado para mantener varios archivos sincronizados entre dos

ordenadores, pero también ser puede utilizar para copiar archivos de un ordenador a otro como scp. Rsync puede establecer la conexión utilizando distintos protocolos, pero uno de ellos es ssh por lo que funcionará también con cualquier servidor ssh.

Si lo que queremos es descargar un fichero desde un servidor en internet, por ejemplo desde una página web, al ordenador remoto en el que estamos trabajando en una sesión ssh podemos utilizar el comando wget o su alternativa curl.

$ wget https://http://ncbi.nlm.nih.gov/una_secuencia.fasta

Ejercicios

1. Conectate a un servidor remoto usando ssh
2. Transfiere un fichero desde tu ordenador al servidor
3. Descarga el fichero https://www.gnu.org/licenses/gpl.txt directamente en el ordenador remoto
4. Copia el fichero gpl.txt a tu ordenador