Prácticas de Fundamentos del Software

Módulo I. Órdenes UNIX y Shell Bash

Sesión Nº3: Permisos y Redirecciones

3-Oct-2011

1 Objetivos principales

- Modificar los permisos de un archivo.
- Comprender cómo se manejan las entradas y salidas de las órdenes con los operadores de redirección.
- Ver algunas formas de combinar varias órdenes, mediante los metacaracteres sintácticos más usuales.

Además, se verán las siguientes órdenes:

Órdenes Linux			
`chmod`	`wc`	`echo`	`date`

Tabla 1. Órdenes de la sesión.

2 Modificación de los permisos de acceso a archivos

En la sesión anterior se han estudiado cuáles son los permisos de acceso a un archivo (lectura, escritura y ejecución) y cómo se pueden conocer dichos permisos (`ls -l`). En este apartado veremos cómo se pueden modificar dichos permisos con la orden **`chmod`**. Obviamente, se debe ser el propietario del archivo para poder cambiar dichos permisos. Dicha orden tiene dos modos de funcionamiento (sólo estudiaremos el primero de ellos):

- Simbólico, para poder cambiar uno o varios de los bits de protección sin modificar el resto.

- Absoluto, que cambia todos los permisos, expresándolos como tres cifras en base 8 (octales). En este caso, los valores tanto para el propietario, grupo o resto de usuarios oscilan entre el `0` (ningún privilegio) hasta el `7` (todos los privilegios). Si dicho valor se codifica en binario, se necesitan tres bits que se corresponden con cada permiso (lectura, escritura y ejecución).

En el modo simbólico, se debe indicar primero a qué grupo de usuarios se va a aplicar el cambio con una letra minúscula:

- **`u`** : propietario
- **`g`** : grupo
- **`o`** : resto de usuarios
- **`a`** : todos los grupos de usuarios

Después, se debe indicar si va a permitir el acceso (+) o se va a denegar el acceso (-). Por último, se indica qué tipo de permiso es el que estamos modificando (r , w , x) y el archivo al que se le van a modificar los permisos. A continuación, se muestran algunos ejemplos de utilización de la orden **`chmod`**:

```
ls -l
-rw-r--r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-r--r--  1 quasimodo alumnos    3410 May 18 2010 ej2
chmod g+w ej2
ls -l
-rw-r--r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-rw-r--  1 quasimodo alumnos    3410 May 18 2010 ej2
chmod o-r ej1
ls -l
-rw-r-----  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-rw-r--  1 quasimodo alumnos    3410 May 18 2010 ej2
chmod a+x ej1
ls -l
-rwxr-x--x  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-rw-r--  1 quasimodo alumnos    3410 May 18 2010 ej2
```

Si en algún momento se permite el acceso (+) a un permiso que ya estaba activado en el archivo o se quita un permiso (-) que no estaba activado en el archivo, la orden **chmod** no tiene ningún efecto. Es posible combinar varias letras en el apartado de grupo de usuarios para que se aplique a más de uno y también al tipo de permisos. También es posible juntar varios cambios en una única orden **chmod** si los separamos por comas y los cambios pueden aplicarse a más de un archivo. Ejemplos:

```
ls -l
-rw-r--r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-r--r--  1 quasimodo alumnos    3410 May 18 2010 ej2
chmod og+w ej2
ls -l
-rw-r--r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-rw-rw-  1 quasimodo alumnos    3410 May 18 2010 ej2
chmod ug-r ej1
ls -l
--w----r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-rw-r--  1 quasimodo alumnos    3410 May 18 2010 ej2
chmod ug+rw ej1
-rw-rw-r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-rw-r--  1 quasimodo alumnos    3410 May 18 2010 ej2
chmod u+x,g-w ej2
-rw-rw-r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rwxr--r--  1 quasimodo alumnos    3410 May 18 2010 ej2
chmod g+x ej*
-rw-rwxr--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rwxr-xr--  1 quasimodo alumnos    3410 May 18 2010 ej2
chmod 754 ej*
-rwxr-xr--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rwxr-xr--  1 quasimodo alumnos    3410 May 18 2010 ej2
```

Ejercicio 1. Se debe utilizar solamente una vez la orden **`chmod`** en cada apartado. Los cambios se harán en un archivo concreto del directorio de trabajo (salvo que se indique otra cosa). Cambiaremos uno o varios permisos en cada apartado (independientemente de que el archivo ya tenga o no dichos permisos) y comprobaremos que funciona correctamente:

- Dar permiso de ejecución al "resto de usuarios".
- Dar permiso de escritura y ejecución al "grupo".
- Quitar el permiso de lectura al "grupo" y al "resto de usuarios".
- Dar permiso de ejecución al "propietario" y permiso de escritura el "resto de usuarios".
- Dar permiso de ejecución al "grupo" de todos los archivos cuyo nombre comience con la letra "e". Nota: Si no hay más de dos archivos que cumplan esa condición, se deberán crear archivos que empiecen con "e" y/o modificar el nombre de archivos ya existentes para que cumplan esa condición.

3 Metacaracteres de redirección

Dispositivo	Valor
`stdin`	`0`
`stderr`	`1`
`stdout`	`2`

El tratamiento de las entradas y salidas en UNIX/Linux es muy simple ya que todas ellas se tratan como flujos de bytes. Cada programa tendrá siempre un dispositivo[1] de entrada estándar (por defecto, el teclado, identificado con el número 0), un dispositivo de salida estándar (por defecto, la pantalla, identificada con el número 1) y un dispositivo estándar para la salida de errores u otra información (por defecto, también es la pantalla, identificada con el número 2). En general, se maneja el siguiente convenio: Si a una orden que lee o escribe en un archivo, no se le especifica un nombre de archivo, la lectura o escritura se realizará por defecto desde la entrada o en la salida estándar. Por ejemplo, si a la orden `cat` no le pasamos un nombre de archivo, al ejecutarla leerá de la entrada estándar, es decir, de lo que escriba el usuario desde el teclado. Pruébalo.

Los metacaracteres de redirección permiten alterar ese flujo por defecto y, por tanto, redireccionar la entrada estándar desde un archivo, y redirigir tanto la salida estándar como el error estándar hacia archivos, además de poder enlazar la salida de una orden con la entrada de otra permitiendo crear un cauce (*pipeline*) entre varias órdenes. La tabla siguiente muestra los metacaracteres de redirección más usuales:

Metacarácter	Descripción
`< nombre`	Redirecciona la entrada de una orden para que la obtenga del archivo *nombre*.
`> nombre`	Redirige la salida de una orden para que la escriba en el archivo *nombre*. Si dicho archivo ya existe, lo sobreescribe.
`&> nombre`	La salida estándar se combina con la salida de error estándar y ambas se escriben en el archivo *nombre*.
`>> nombre`	Funciona igual que el metacarácter "**>**" pero añade la salida estándar al final del contenido del archivo *nombre*.
`&>> nombre`	Igual que el metacarácter "**&>**", pero añadiendo las dos salidas combinadas al final del archivo *nombre*. En algunas versiones de Linux (por ejemplo, Ubuntu), se sustituiría por: `>> nombre 2>& 1` Donde el 1 final representa el dispositivo estándar de salida.
`2> nombre`	Redirige la salida de error estándar a un archivo (sólo con "bash").
`|`	Crea un cauce entre dos órdenes. La salida de una de ellas se utiliza como entrada de la otra.
`|&`	Crea un cauce entre dos órdenes utilizando las dos salidas (estándar y error) de una de ellas como entrada de la otra.

1 Los dispositivos en UNIX/Linux se representan como archivos.

3.1 Redirección de la entrada estándar (<)

Algunas órdenes toman su entrada de archivos cuyo nombre se pasa como argumento, pero si no se especifica dicho archivo, la lectura se lleva a cabo desde la entrada estándar. Otras órdenes sólo leen de la entrada estándar (como la orden **`mail`**), por lo que si queremos que lean desde un archivo debemos usar el metacarácter de redirección de entrada.

Como ejemplo, obtendríamos el mismo resultado ejecutando las siguientes órdenes:

```
cat archivo

cat < archivo
```

3.2 Redirección de la salida estándar (> , >>)

Las salidas de las órdenes se dirigen normalmente a la pantalla, pudiéndose almacenar en un archivo utilizando los metacaracteres de redirección de salida. Si el nombre del archivo al que se redirecciona la salida no existe, ambos metacaracteres lo crean. La diferencia entre ellos aparece en el caso de que dicho archivo existiera previamente, ya que si usamos "**>**" se borra completamente dicho archivo antes de escribir la salida de la orden mientras que usando "**>>**" no se pierde la información previa que contenía el archivo y se añade la salida de la orden al final del archivo. A continuación, se muestran algunos ejemplos de utilización de estos metacaracteres, suponiendo que inicialmente sólo hay dos archivos en nuestro directorio de trabajo (*notas* y *listado*).

```
pwd
/home/users/quasimodo
ls
listado  notas
ls > temporal
ls
listado  notas  temporal
cat temporal
listado
notas
pwd > temporal
cat temporal
/home/users/quasimodo
ls >> temporal
cat temporal
/home/users/quasimodo
listado
notas
temporal
```

En el siguiente ejemplo, mostramos por pantalla, de todos los archivos del directorio de trabajo que empiezan por la letra "e", el listado (en formato largo) de los dos primeros (para ello se hace uso de la orden **`head -n <file>`**, que muestra las **`n`** primeras líneas del archivo dado como argumento **`<file>`**).

```
ls e*
ej1  ej31  ej32  ej4
```

```
ls -l e* > temporal
head -2 temporal
-rw-r--r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-r--r--  1 quasimodo alumnos    3410 May 18 2010 ej31
rm temporal
```

Ejercicio 2. Utilizando solamente las órdenes de la sesión anterior y los metacaracteres de redirección de salida:

- Crear un archivo llamado *ej31* , que contendrá el nombre de los archivos del directorio padre del directorio de trabajo.
- Crear un archivo llamado *ej32* , que contendrá las dos últimas líneas del archivo creado en el ejercicio anterior.
- Añadir al final del archivo *ej32* , el contenido del archivo *ej31* .

3.3 Redirección del error estándar (&> , &>>)

Las salidas de las órdenes se dirigen normalmente a la salida estándar. Sin embargo, muchas órdenes escriben mensajes de error o información adicional en otro flujo de bytes que es la salida de error estándar (normalmente, la pantalla). Si se redirige la salida de una orden sólo con el metacarácter ">", los mensajes de error siguen saliendo por la pantalla. Si añadimos el carácter "&" a los metacaracteres de redirección de salida, se redirigen hacia el archivo indicado tanto la salida estándar como el error estándar. Para redirigir solamente la salida de error estándar a un archivo se utiliza el metacarácter "2>", aunque sólo funciona con "bash".

A continuación, se muestra un ejemplo de utilización de estos metacaracteres, suponiendo que inicialmente sólo hay dos archivos en nuestro directorio de trabajo `notas` y `listado` y, por tanto, el intentar visualizar el contenido de un archivo inexistente `practica`, origina un mensaje de error.

```
ls
listado  notas
cat practica
cat: practica: No existe el archivo o el directorio
cat practica > temporal
cat: practica: No existe el archivo o el directorio
ls
listado  notas  temporal
cat temporal

cat practica 2> temporal
cat temporal
cat: practica: No existe el archivo o el directorio
```

3.4 Creación de cauces (|)

Los cauces (*pipelines*) son una característica distintiva de UNIX/Linux. Un cauce conecta la salida estándar de la orden que aparece a la izquierda del símbolo | con la entrada estándar que aparece a la derecha de dicho símbolo. Se produce un flujo de información entre ambas órdenes sin necesidad de usar un archivo como intermediario de ambas órdenes.

Como ejemplo de utilización de este mecanismo, a continuación se muestra otra alternativa para realizar el ejemplo presentado al final de la sección 3.2 , que permite implementar el ejercicio de una forma más compacta y sin tener que usar un archivo intermedio.

```
ls e*
ej1  ej31  ej32  ej4
ls -l e* | head -2
-rw-r--r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
-rw-r--r--  1 quasimodo alumnos    3410 May 18 2010 ej31
```

Ejercicio 3. Utilizando el metacarácter de creación de cauces y sin utilizar la orden **`cd`**:

- Mostrar por pantalla, el listado (en formato largo) de los últimos 6 archivos del directorio padre en el directorio de trabajo.
- La orden **`wc`** muestra por pantalla el número de líneas, palabras y caracteres de un archivo (consulta la orden **`man`** para conocer más sobre ella). Utilizando dicha orden, mostrar por pantalla el número de caracteres (sólo ese número) de los archivos del directorio de trabajo que comiencen por los caracteres "e" o "f".

4 Metacaracteres sintácticos

Sirven para combinar varias órdenes y construir una única orden lógica. La tabla siguiente muestra los metacaracteres sintácticos más usuales:

Metacarácter	Descripción
`;`	Separador entre órdenes que se ejecutan secuencialmente.
`( )`	Se usan para aislar órdenes separadas por ";" o por "|". Las órdenes dentro de los paréntesis son tratadas como una única orden.
`&&`	Separador entre órdenes, en la que la orden que sigue al metacarácter "&&" se ejecuta sólo si la orden precedente ha tenido éxito (no ha habido errores).
`||`	Separador entre órdenes, en la que la orden que sigue al metacarácter "||" se ejecuta sólo si la orden precedente falla.

4.1 Unión de órdenes en la misma línea (;)

El uso del punto y coma permite escribir dos o más órdenes en la misma línea. Las órdenes se ejecutan secuencialmente (como si se hubiesen escrito en líneas sucesivas). En programas del shell permite una asociación visual de órdenes relacionadas (mejora la comprensión del programa y hace que tengas menos líneas). Trabajando de forma interactiva, permite ejecutar varias órdenes sin tener que esperar a que se complete una orden para poder introducir la siguiente. A continuación, se muestra un ejemplo de utilización.

```
pwd
/home/users/quasimodo
ls -l
-rw-r--r--  1 quasimodo alumnos   23410 Mar 15 2010 ej1
drw-r--r--  1 quasimodo alumnos    3410 May 18 2010 dir1
cd dir1 ; ls
programa1
programa2
pwd
/home/users/quasimodo/dir1
```

4.2 Combinación de órdenes con los paréntesis

Combinando órdenes podremos aislar, cuando nos interese, un cauce, o una secuencia de punto y coma del resto de la línea de órdenes. Lo ilustraremos con un ejemplo utilizando la orden **`date`** (que proporciona la fecha y hora del sistema) y la orden **`pwd`**, que unimos por un punto y coma y creamos un cauce que termina con la orden **`wc`**.

```
date
Wed oct 6 10:12:04 WET 2010
pwd
/home/users/quasimodo
pwd ; date | wc
/home/users/quasimodo
1 6 27
(pwd ; date) | wc
2 7 48
```

Como se puede comprobar, el uso de paréntesis produce un resultado diferente, ya que, en el primer caso, la orden **`wc`** se ejecuta solamente sobre la salida de la orden **`date`**, mientras que al utilizar paréntesis, es la combinación de las salidas de las 2 órdenes la que se pasa como entrada a la orden **`wc`**.

4.3 Ejecución condicional de órdenes (&& , ||)

El shell proporciona dos metacaracteres que permiten la ejecución condicional de órdenes según el estado de finalización de una de ellas. Separar dos órdenes con **`&&`** o **`||`**, provoca que el shell compruebe el estado de finalización de la primera y ejecute la segunda sólo si la primera tiene éxito o falla, respectivamente. En los siguientes ejemplos, ilustramos el uso de estos metacaracteres. En ambos casos, la primera orden es la visualización del listado (en formato largo) del archivo *notas* y el resultado varía según esté presente o no dicho archivo en el directorio de trabajo.

```
pwd
/home/users/quasimodo
ls
listado notas
ls -l notas && pwd
-rw-r--r--  1 quasimodo alumnos   3418 Mar 15 2010 notas
/home/users/quasimodo
ls -l notas || pwd
-rw-r--r--  1 quasimodo alumnos   3418 Mar 15 2010 notas
rm notas
ls -l notas && pwd
ls: notas: No existe el archivo o el directorio
ls -l notas || pwd
ls: notas: No existe el archivo o el directorio
/home/users/quasimodo
```

No existe ninguna restricción que limite el número de órdenes que aparecen antes del metacarácter **`&&`** o **`||`**, pero sólo se evalúa el estado de la última de ellas. Es posible conectar en una misma línea ambos metacaracteres, como vemos en el siguiente ejemplo, en el que, si existe un archivo, queremos que nos muestre el resultado de la orden **`wc`** aplicada sobre dicho archivo y, en caso de que no exista, nos muestre un mensaje indicativo por pantalla (para eso, utilizamos la orden **`echo`**).

```
ls
listado
notas
ls notas && wc notas || echo "no existe el archivo notas"
```

```
notas
86 324 5673
rm notas
ls notas && wc notas || echo "no existe el archivo notas"
ls: notas: No existe el archivo o el directorio
no existe el archivo notas
```

Ejercicio 4. Resuelva cada uno de los siguientes apartados.

a) Crear un archivo llamado `ejercicio1`, que contenga las 17 últimas líneas del texto que proporciona la orden **`man`** para la orden **`chmod`** (se debe hacer en una única línea de órdenes y sin utilizar el metacarácter ";").

b) Al final del archivo `ejercicio1`, añadir la ruta completa del directorio de trabajo actual.

c) Usando la combinación de órdenes mediante paréntesis, crear un archivo llamado `ejercicio3` que contendrá el listado de usuarios conectados al sistema (orden **`who`**) y la lista de archivos del directorio actual.

d) Añadir, al final del archivo `ejercicio3`, el número de líneas, palabras y caracteres del archivo `ejercicio1`. Asegúrese de que, por ejemplo, si no existiera ejercicio1, los mensajes de error también se añadieran al final de `ejercicio3`.

e) Con una sola orden **`chmod`**, cambiar los permisos de los archivos `ejercicio1` y `ejercicio3`, de forma que se quite el permiso de lectura al "grupo" y se dé permiso de ejecución a las tres categorías de usuarios.