Introducción a Sistemas Operativos: Empezando

Clip 17
Francisco J Ballesteros

1. Usuarios y permisos

Cuando vimos cómo entrar al sistema, y al utilizar comandos para listar ficheros y directorios, hemos visto que UNIX tiene idea de que hay usuarios en el sistema. La idea de "*usuario*" es de nuevo una abstracción del sistema. Concretamente, cada programa que ejecuta lo hace a nombre de un usuario. Dicho "nombre" es un número para UNIX. El comando *id* muestra el identificador de usuario, o *user id*, o *uid*.

```
unix$ id
uid=501(nemo) gid=20(staff) groups=20(staff)
```

Para crear un usuario en UNIX normalmente editamos varios ficheros en `/etc` para informar al sistema del nuevo usuario. Esto suele hacerse utilizando comandos (descritos en la sección 8 del manual) para evitar cometer errores. Concretamente, hay que indicar al menos el nombre de usuario (por ejemplo, "`nemo`"), el número que utiliza UNIX como nombre para el usuario (por ejemplo, 501)., y qué directorio utiliza ese usuario como casa.

Además, habrá que crear el directorio casa para el usuario y darle permisos para que pueda utilizarlo. También se suele indicar qué shell prefiere utilizar dicho usuario, por ejemplo, `/bin/sh`, y otra información administrativa incluyendo el nombre real del usuario.

En UNIX, los usuarios pertenecen a uno o más grupos de usuarios. Una vez más, para UNIX, un grupo de usuarios es un número. En nuestro caso, *nemo* pertenece al grupo *staff* (y UNIX lo conoce como 20). El número que identifica al grupo de usuarios se denomina *group id* o *gid*. Al crear un usuario también hay que decir a qué grupos pertenece.

Aunque la forma de crear un usuario varía mucho de un UNIX a otro, este comando se utiliza en OpenBSD:

```
unix# adduser
Enter username []: loser
Enter full name []: Mr Loser
Enter shell bash csh ksh nologin rc sh [ksh]: ksh
Uid [1005]:
Login group loser [loser]: staff
Login group is ‘‘staff’’. Invite loser into other groups: guest no
[no]:
Enter password []:
Enter password again []:
```

Su efecto es editar el fichero de cuentas `/etc/passwd` (y otros ficheros) para guardar la información del usuario, de tal forma que *login* y otros programas puedan encontrarla. Por ejemplo, esta línea aparece en el fichero de cuentas tras crear el usuario:

```
loser:*:1005:50:Mr Loser:/home/loser:/bin/ksh
```

Además, el comando ha creado el directorio `/home/loser` y ha dado permisos al nuevo usuario para utilizarlo.

De hecho, los ficheros y directorios pertenecen a un usuario (UNIX guarda el *uid* del dueño) y a un grupo de usuarios (UNIX guarda el *gid*). El comando *adduser* ha hecho que `/home/loser` pertenezca al usuario `loser`, y al grupo `staff` en nuestro ejemplo.

Los programas en ejecución están ejecutando a nombre de un usuario (con un *uid* dado). Cuando un programa crea ficheros, estos ficheros se crean a nombre del usuario que ejecuta el programa.

Para realizar cualquier operación o llamada al sistema el usuario debe tener permiso para ello. Por ejemplo, normalmente no podrás ejecutar *adduser* para crear usuarios. Las llamadas al sistema que utiliza y los ficheros que edita no es probable que estén accesibles con tu usuario. En UNIX, el usuario número 0 (con *uid* 0) es especial. El kernel incluye cientos de comprobaciones que hacen que dicho usuario pueda efectuar la llamada que haga y que, en otro caso, se compruebe si el usuario tiene permiso para hacerla. Por eso a ese usuario se le llama *superusuario*. Su nombre suele ser *root*, pero UNIX lo conoce como *uid* 0. Recuerda que el nombre es tan sólo texto que utilizan programas como *ls* para mostrar el nombre de usuario. El kernel utiliza *uids*. El prompt del shell suele cambiar si el usuario es *root*, para avisarte de ello. En nuestro ejemplo el prompt era "`unix#`" y no "`unix$`".

Podemos cambiar de un usuario a otro utilizando el comando *su*, (*switch user*) que ejecuta en shell a nombre de otro usuario. Por ejemplo,

```
unix$ id
id=501(nemo) gid=20(staff) groups=20(staff)
unix$ su
Password: *******
unix# id
uid=0(root) gid=0(wheel) groups=0(wheel), 20(staff)
unix# adduser
...
unix# exit
unix$ id
id=501(nemo) gid=20(staff) groups=20(staff)
```

Cada fichero en unix incluye información respecto a qué cosas puede hacer qué usuario con el mismo. A esto se le llama *lista de control de acceso*, (o ACL, por *access control list*). Normalmente, UNIX guarda 9 bits al menos correspondientes a 9 permisos para cada fichero: 3 permisos para el dueño del fichero, otros 3 para cualquier usuario que no sea el dueño, pero que pertenezca al grupo al que pertenece el fichero y otros 3 permisos para cualquier otro usuario. Estos bits se guardan en la estructura de datos que utiliza el sistema operativo para cada fichero. Pues bien, los permisos son *lectura*, *escritura* y *ejecución*. Si miramos el listado largo de un fichero podemos verlos:

```
unix$ ls -l fich
-rwxr-xr-- 1 nemo staff 1024 May 30 16:31 fich
```

Concretamente, "`rwxrwxr-x`" son los 9 bits para permisos a los que nos hemos referido. Hay más bits y más permisos, pero los veremos más adelante. En el fichero que hemos listado, `fich`, los tres primeros permisos (esto es, "`rwx`") se aplican al dueño del fichero (esto es, a *nemo*, tal y como indica *ls*). Dicho de otro modo, cualquier programa que ejecute a nombre de *nemo* e intente hacer llamadas al sistema para utilizar el fichero, estará sujeto a los permisos `rwx`. Para cualquier usuario que, no siendo el dueño, pertenezca al grupo *staff*, se aplicarán los siguientes tres permisos: `r-x`. Y para cualquier otro usuario, se aplicarán los últimos tres permisos: `r--`.

En cualquiera de los casos, la "`r`" indica permiso de lectura, la "`w`" indica permiso de escritura y la "`x`" indica permiso de ejecución. En un fichero, leer el contenido de fichero requiere permiso de lectura. Los comandos como *cat* y *cp* leen el contenido de ficheros y requieren dicho permiso. Escribir el contenido del fichero require permiso de escritura. Por ejemplo,

```
unix$ echo hola >fich
```

require permiso de escritura en `fich`. El permiso de ejecución se utiliza para permitir que un fichero se ejecute (por ejemplo, en respuesta a un comando que escribimos en el shell).

En el caso de los directorios, la "`r`" indica permiso para listar el contenido del directorio, o leer el directorio. La "`w`" indica permiso para modificar el directorio, ya sea creando ficheros dentro o borrándolos o cambiando su nombre. Y la "`x`" indica permiso para entrar en el directorio (cambiando de directorio dentro de él, por ejemplo).

Para modificar los permisos se utiliza el comando *chmod*, (*change mode*). Si primer argumento se utiliza para dar o quitar permisos y el resto de argumentos corresponden a los ficheros a los que hay que ajustarle los permisos. Por ejemplo,

```
unix$ chmod +x fich
```

hace que `fich` sea ejecutable. Para UNIX, cualquier fichero con permiso de ejecución es un ejecutable. Otra cosa será si cuando se intente ejecutar su contenido tiene sentido o no.

Para evitar escribir un fichero por accidente, podemos quitar su permiso de escritura:

```
unix$ chmod -w fich
```

Cuando utilizamos "`+`", damos permiso. Cuando utilizamos "`-`", lo quitamos. Detrás podemos escribir uno o más permisos:

```
unix$ chmod +rx fich
```

En ocasiones, queremos dar permisos sólo para el usuario que es propietario del fichero:

```
unix$ chmod u+w fich
```

La "`u`" quiere decir *user*. Igualmente, podemos quitar un permiso sólo para el grupo de usuarios al que pertenece el fichero:

```
unix$ chmod g-w fich
```

La "`g`" quiere decir *group*. Del mismo modo podemos usar inicialmente una "`o`" (por *others*) para cambiar los permisos para el resto de usuarios.

En otros casos, resulta útil utilizar como argumento para los permisos el número que guarda UNIX para los mismos. Igual sucede al programar en C, donde hemos de utilizar una llamada al sistema que acepta un entero para indicar los permisos que deseamos. Es fácil si pensamos que estos nueve bits son tres grupos de tres bits. Cada grupo de permisos puede ir de "000" a "111". Escribiendo en base 8 podemos utilizar un sólo dígito para cada grupo de bits, y es justo lo que se hace. La figura 1 muestra el esquema. El comando

```
unix$ chmod 755 fich
```

dejaría los permisos de `fich` como "`rwxr-xr-x`", dado que 7 es 111 en binario, con los tres bits a 1. Además, el segundo y tercer dígito en octal valen 5, por lo que los permisos para el grupo serían "`r-x`" (4 mas 1, o 101) y lo mismo los permisos para el resto de usuarios.

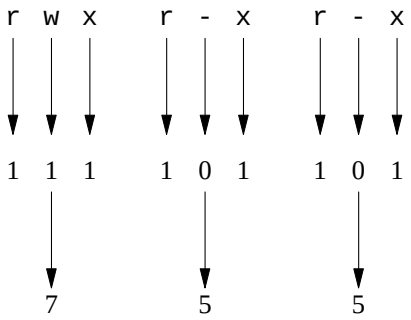

Figura 1: *Los permisos en UNIX se codifican como un número en octal en muchas ocasiones.*