Práctica 1 - Introducción a Unix

1. Introducción

El objetivo principal que se persigue en esta práctica es que el alumno se familiarice con el entorno de trabajo en el que se desarrollarán las prácticas posteriores: Unix/Linux.

En esta primera parte aprenderá a usar el sistema operativo Linux. Gran parte de los conceptos no son específicos de Linux sino que pueden aplicarse igualmente a cualquier sistema operativo de tipo UNIX. Además se aprenderá a controlar los procesos que ha lanzado su usuario y a modificar los permisos de sus ficheros.

Gran parte de esta práctica (relacionada con comandos y utilidades básicas) se puede llevar a cabo fuera del laboratorio, y en cualquier distribución de Linux pues los conceptos son básicos y bastante generales. Esto incluye no solo Linux sino también un BSD como macOS.

2. Consiguiendo una cuenta

Los sistemas UNIX son multi-usuario, permitiendo que el mismo equipo sea usado por varios usuarios (incluso al mismo tiempo). Los usuarios pueden utilizar el ordenador tanto de forma local (estando sentados delante) como de forma remota (dándole órdenes a través de la red desde otro ordenador). Normalmente los propietarios de ordenadores UNIX no quieren que cualquiera pueda utilizar su ordenador así que es necesario que el sistema operativo UNIX almacene los datos de los usuarios que tienen derecho a usarlo. A esto se le llama cuenta. La cuenta se identifica por un nombre o identificador de usuario y una contraseña que se supone que solo es conocida por el usuario autorizado y que le permite probar su identidad.

Cada cuenta permite a su usuario utilizar una máquina UNIX con diferentes privilegios que otros usuarios, pudiendo acceder sólo a los ficheros que sean propiedad de ese usuario y utilizando las aplicaciones que se permitan a ese usuario. Hay una cuenta especial en las máquinas UNIX que es la cuenta del administrador del sistema. Esta cuenta se llama normalmente `root` y tiene permisos para hacer cualquier cosa en el sistema.

Al proceso de identificarse en el sistema mediante el nombre de cuenta y contraseña antes de utilizar el sistema lo llamaremos hacer *login*.

El primer paso, por tanto, para usar un sistema UNIX es conseguir una cuenta.

Si está en una de las mesas con armarios de prácticas, cada grupo de prácticas dispone de varios ordenadores con Linux instalado y etiquetados como *PC A, PC B, PC C y PC SC*. Si no están encendidos puede encenderlos. Los PCs A, B y C se usarán en las prácticas que utilicen la red, la cuenta de usuario para acceder a ellos es compartida por todos los estudiantes de la asignatura y se proporcionará cuando sea necesario. **En esta primera práctica utilizaremos el PC SC** (o cualquier PC de fuera de los armarios), este PC está bien configurado en red y utiliza el sistema de cuentas central del Laboratorio de Telemática, a cada estudiante se le ha asignado una cuenta que le permite utilizar cualquiera de los PCs SC o de los ordenadores de propósito general del

laboratorio. Esta cuenta se llamará `arss<número_de_grupo>` y el profesor comunicará la contraseña en clase.

Esta práctica se realizará en un solo ordenador. Si está en los ordenadores de los armarios de prácticas seleccione en el conmutador del teclado tu *PC SC*. Dedique unos momentos a comprobar el funcionamiento del conmutador de teclado y pantalla (KVM). Pulsando 2 veces la tecla de *BloqueoDesplazamiento* del teclado podrá ver el menú del KVM y las opciones para cambiar de un PC a otro el teclado, pantalla y ratón.

Si está en una mesa con un solo ordenador por grupo no hace falta que se preocupe por la selección de ordenador, claro.

En cualquier caso, observe que el ordenador presenta una pantalla de *login* gráfico.

Con la combinación de teclas `Control+Alt+F1` puede elegir un *login* en modo texto, pudiendo volver al modo gráfico con `Control+Alt+F7` (si este no funciona pruebe `Control+Alt+F8`).

Autentifíquese utilizando su cuenta y contraseña en el *login* de modo texto. Una vez identificado mediante usuario y contraseña, el sistema ha lanzado un proceso que recibe las órdenes que teclea y las interpreta, lanza los programas indicados por esas órdenes y muestra los resultados en pantalla. Este programa se llama genéricamente *shell*, y hay varias *shells* diferentes disponibles normalmente en los sistemas UNIX que el usuario puede elegir. En el Linux que está usando, si el usuario no elige otro, está usando la *shell* llamada `bash`. Otras *shells* típicas son `csh`, `tcsh`, `ksh` y `sh`.

La *shell* permite escribir nombres de programas que serán lanzados por la propia *shell*. UNIX es un sistema operativo multitarea que puede tener varios programas funcionando al mismo tiempo. A cada programa que está siendo ejecutado lo llamamos proceso. Así pues, cuando se escribe algo en la *shell* y se pulsa *ENTER*, se lanza un nuevo proceso que ejecuta las instrucciones del programa que está guardado en el archivo que tiene como nombre el nombre que se ha introducido. La *shell* se queda esperando a que el proceso termine y a continuación vuelve a preguntar por otro comando. Todos los procesos que se ejecutan están asignados a un usuario. Cuando ha *hecho login* se ha creado una *shell* asignada a su usuario, y todos los procesos que ella cree serán de ese usuario y se le aplicarán los privilegios y restricciones que correspondan al mismo.

3. Un primer comando

El primer comando que vamos a aprender va a permitir cambiar la contraseña de la cuenta con la que se está autentificado. Este comando en una máquina UNIX es `passwd`. Debe tener en cuenta que en el laboratorio su cuenta y su contraseña no se almacenan en el ordenador donde está trabajando (como sí se suele hacer si por ejemplo instala Linux en su propio ordenador) sino que se hace en un servidor central. Una vez cambie su contraseña, deberá acceder con ella sea cual sea el ordenador que use del laboratorio (excepto los PC A, B y C que son independientes ya que no están conectados a nuestra red general). Cambie su contraseña por una más segura con[1]:

[1] En estos ejemplos el "$" al princpio de cada uno hace referencia al "prompt" que muestra la shell. No tiene que escribirlo. Escriba lo que venga a continuación

```
$ passwd
Identity validation…
Enter your UNIX password:      [metemos aquí la contraseña actual ]
Changing UNIX and samba passwords for <usuario>
New password:      [metemos aqui la contraseña nueva  ]
Retype new password:     [metemos otra vez la contraseña nueva para
                                  verificar]
```

Tenga en cuenta que al igual que en el login en modo texto, no verá aparecer carácteres mientras introduce las contraseñas actual y nueva.

Los mensajes concretos pueden variar según la configuración de la administración de cuentas. Observe cómo el comando nos informa de que la contraseña ha sido cambiada correctamente. En caso de que no haya sido así, lea el mensaje de error para descubrir por qué y cámbiela correctamente. Tenga en cuenta que puede fallar si no se escribe bien la contraseña actual o si no se escribe la misma nueva contraseña dos veces. También es posible que el comando de un error si intenta poner una contraseña demasiado fácil; depende de las políticas de seguridad que tengan establecidas los administradores. Si quiere ver este error, pruebe a poner como contraseña tu nombre de usuario.

Una regla muy importante es comprobar que la contraseña se ha cambiado correctamente **antes de salir**. Use otra consola en modo texto para comprobar que puede entrar con la nueva contraseña. (Puede obtener otra consola de texto con Control+Alt+F2. Puede volver a la shell anterior con Control+Alt+F1). Tras esto, "termine" la *shell* tecleando `exit` y pulsando *ENTER*. Lo que está haciendo es indicándole a la *shell* que termine su propio proceso. Con eso el proceso, si está en modo gráfico el terminal (la ventana) terminará también. Si está en un login de texto (sin interfaz gráfico) entonces el proceso de *login* que le pidió el usuario al principio se despierta y vuelve a pedir usuario y contraseña.

Si olvida cerrar la sesión en el interfaz gráfico y en todas las consolas de texto cualquiera delante de ese ordenador puede acceder a su cuenta, lo que implica que puede borrar todos sus ficheros o usar todos los privilegios de su cuenta. Acuérdese siempre de dejar un ordenador UNIX en la pantalla de *login*. Usando el interfaz gráfico tendrá una opción de menú para salir de la cuenta.

***Nota importante**: Tenga en cuenta que tener los ordenadores con contraseñas conocidas es un problema grave de seguridad, así que las cuentas que sigan teniendo la misma contraseña se considerará que no tienen propietario y serán eliminadas o bloqueadas.*

4. Primeros pasos

Si tiene el terminal en modo texto, salga de la cuenta con *exit*. Utilice *Control+Alt+F7* para obtener el *login* gráfico y entra en su cuenta en modo gráfico, que es lo que por comodidad emplearemos a partir de ahora.

Familiarícese con el interfaz gráfico de su cuenta. Debería ser capaz de encontrar rápidamente varias herramientas que le serán de utilidad.

Localice un navegador web en el que pueda ir al MiAulario para poder seguir las prácticas a partir de ahora (Firefox está instalado, en algunos ordenadores puede estar también instalado Google Chrome).

Localice un navegador del sistema de ficheros de UNIX en el que pueda ver de forma gráfica su directorio *home* y moverse por los directorios y subdirectorios.

Localice al menos un editor de texto plano para editar ficheros de texto (no confundir con un editor de texto con formato, tipo OpenOffice). Al menos debería encontrar `gedit` o `kate`; pruébelos.

Localice un terminal en el que pueda abrir una ventana de comandos con una *shell,* con la ventaja en comparación de las consolas de que puede tener varias abiertas simultáneamente.

Para empezar a conocer UNIX, lo más educativo en ingeniería es aprender algunos conceptos más sobre el terminal y la *shell*. Abra un terminal, con lo que obtendrá una *shell* en una ventana.

5. Manejando ficheros

Los sistemas UNIX, al ser multi-usuario, requieren que cada cuenta de usuario pueda tener diferente acceso a los directorios y ficheros. Nada más ser lanzada, la *shell* está esperando órdenes y está posicionada en el directorio *home* del usuario. Esto quiere decir que todas las referencias a ficheros que se le den las entenderá relativas a ese directorio si no se le da una "ruta absoluta" (comienzan por "/", que es el indicador de la raíz del sistema de ficheros). Observe cuál es su directorio *home* con el comando "print working directory":

```
$ pwd
```

El que muestre es el directorio en el que puede guardar sus ficheros.

Puede crear un fichero con cualquiera de los siguientes comandos:

```
$ echo "hola" > fichero1
$ cat > unfichero
escriba algo
^D (pulsar control+D)
$ touch fichero_tres
```

Use el comando `ls` para listar los ficheros del directorio actual y comprobar que se han creado. Puede emplear los siguientes comandos para ver su contenido:

```
$ cat unfichero
$ more fichero1  # para salir pulse q
$ less unfichero # para salir pulse q
```

El sistema de ficheros soporta una serie de permisos que dicen quién tiene derecho a usar o no cada fichero. Puede ver estos permisos con el comando `ls`. Puede consultar todas las opciones de un comando mediante el sistema de ayuda de UNIX proporcionado por el comando `man`, por ejemplo `man ls`.

Haga un listado de los ficheros de su directorio en formato largo. Observará algo parecido a esto:

```
$ ls -l
drwxr-xr-x   3 usuario staff     4096 Dec 16 08:14 Desktop
-rw-r--r--   1 usuario staff 41685513 Sep 27 13:57 usuario.tgz
-rw-r--r--   1 usuario staff  1000000 Nov 12 20:00 oo
```

```
-rw-r--r--   1 usuario staff  1000000 Nov 12 20:02 ooo
drwxr-xr-x   2 usuario staff     4096 Oct  6 13:07 prueba_c
-rwxr-xr-x   1 usuario staff       94 Nov 25 11:56 vnc_to_usuario.bash
drwx------  10 usuario staff     4096 Oct  6 11:26 W2000
```

El primer bloque de letras y guiones en cada línea indica los permisos sobre el fichero. La primera letra indica el tipo de fichero. Las 9 letras restantes se deben interpretar en grupos de 3. Cada grupo indica los permisos que tienen diferentes tipos de usuarios.

Ejemplo:

Para el fichero `usuario.tgz` tenemos:

```
[-] [rw-] [r--] [r--]   1 usuario staff 41685513 Sep 27 13:57 usuario.tgz
```

-	rw-	r--	r--
Tipo: fichero	Permisos de lectura y escritura para el propietario	Permisos de lectura para el grupo	Permisos de lectura para los demás

Para el directorio `Desktop` tenemos:

```
[d] [rwx] [r-x] [r-x]   3 usuario staff     4096 Dec 16 08:14 Desktop
```

d	rwx	r-x	r-x
Tipo: directorio	Permisos de lectura, escritura y acceso para el propietario	Permisos de lectura y acceso para el grupo	Permisos de lectura y acceso para los demás

Los permisos para cada tipo de usuario se componen por tanto de un bloque de 3 letras que pueden ser `r w x` o bien en lugar de la letra puede haber un – (hay más posibilidades para ciertos casos especiales que no detallaremos en este momento). Cada espacio de letra se refiere a un tipo de operación sobre ese fichero. Si está la letra quiere decir que ese usuario tiene permiso para hacer esa operación y si está el guion quiere decir que no tiene permiso. Las operaciones son ligeramente diferentes según se trate de ficheros o directorios. Puede verlas en la siguiente tabla

Operación	**Sobre ficheros**	**Sobre directorios**
`r`	El usuario puede leer el contenido del fichero	El usuario puede leer el contenido del directorio, esto es, ver la lista de ficheros que hay dentro
`w`	El usuario puede modificar el contenido del fichero	El usuario puede modificar el contenido del directorio, es decir, añadir o quitar ficheros

x	El usuario puede ejecutar el fichero. Sólo tiene sentido si el fichero contiene un programa	El usuario puede acceder a los ficheros y subdirectorios dentro del directorio

Como se puede ver, el significado de `r` y `w` es prácticamente el mismo para ficheros y directorios, para entenderlas sólo hay que pensar que el directorio es como un fichero que contiene la lista de lo que hay dentro. Así entenderá fácilmente que leer un directorio es ver lo que hay dentro, pero solo los nombres de los ficheros, no su contenido. Escribir en un directorio es modificar la lista de ficheros que contiene, pero no modifica el contenido de un fichero que está dentro.

Por otra parte, el sistema operativo debe decidir qué conjunto de permisos debe aplicar a un usuario que intenta acceder a un fichero. Para ello todos los ficheros tienen un usuario y un grupo al que pertenecen. Todos los ficheros del ejemplo son del usuario `usuario` y del grupo `staff`, es decir para acceder al directorio `Desktop`, al usuario `usuario` se le aplican los permisos del propietario `rwx` (puede hacer cualquier operación), a los usuarios que pertenezcan al grupo `staff` se les aplica el segundo bloque `r-x` (pueden leer y acceder al interior, pero no modificarlo) y a todos los demás usuarios se les aplica el ultimo bloque `r-x`.

Pruebe a editar con editores de texto de terminal alguno de los ficheros. Hay varios editores de texto de línea de comandos en un UNIX típico. Pruebe por ejemplo:

```
$ pico unfichero
$ nano unfichero
$ vi unfichero
```

Los primeros son más sencillos de utilizar. Los dos últimos son clásicos en la historia de UNIX y son muy completos a la vez que más complejos (y más fáciles de encontrar instalados en muchos sistemas). Normalmente utilizará editores en el entorno gráfico, pero siempre es útil saber manejarse con cualquiera de estos para las ocasiones en las que no pueda usar un interfaz gráfico (o la máquina no disponga del mismo, por ejemplo un router o casi cualquier sistema embebido). Si se quedas atascado puede salir de `vi` pulsando `ESC` y luego escribiendo `:q!` y *ENTER*. Para salir guardando cambios escriba `:wq!` .

También puede usar el terminal para lanzar editores gráficos. Pruebe a ejecutar:

```
$ gedit unfichero &
```

(Más adelante explicaremos para qué sirve el `&` final)

Ahora puede volver a pensar en los permisos de los ficheros. Cambie los permisos del propietario del fichero. Podemos quitar el permiso de escritura con el comando `chmod`. Escriba:

```
$ chmod u-w unfichero
```

Compruebe si puedes ver el contenido y modificarlo.

Compruebe después el efecto de estas otras modificaciones:

```
$ chmod u-rwx unfichero  # quita todos los permisos al propietario
$ chmod u-r unfichero    # quita el permiso de lectura al propietario
$ chmod u+w unfichero    # da permiso de escritura al propietario
```

Compruebe si quitar el permiso de escritura impide borrar un fichero. Use los comandos `cp` para copiar y `rm` para borrar.

```
$ cp unfichero fichero2
$ chmod u-w fichero2
$ rm fichero2
```

¿Se ha borrado fichero2? ¿A qué se debe ésto? ¿Cómo podemos conseguir protegerlo?

El sistema de ficheros nos permite organizar los ficheros en directorios y subdirectorios. Pruebe el funcionamiento de los siguientes comandos

```
$ pwd            # muestra el directorio actual
$ mkdir midir    # crea un subdirectorio midir en el directorio actual
$ ls             # para comprobar que hay un directorio nuevo
$ cd midir       # el directorio actual pasa a ser midir
$ pwd            # comprobación de lo anterior
$ cd ..          # el directorio actual pasa a ser el padre de midir
$ pwd            # comprobación
```

Los ficheros y directorios pueden referenciarse con paths absolutos o relativos al directorio actual. El directorio raíz es /

Prueba estos comandos:

```
$ more /etc/services
$ cd /etc
$ more services
$ more ~/unfichero    # ~ es el directorio home de ese usuario
$ cd                  # o cd ~
$ pwd
$ cd ..               # .. es el directorio padre del directorio actual
$ pwd
$ cd ../../../etc
$ more ./services     # . es el directorio actual
```

Compruebe el efecto del permiso de ejecución (x) para un directorio:

```
$ cd
$ mkdir midir/cosas
$ cp /etc/services midir/cosas/servicios
$ chmod u-x midir
```

¿Puede acceder al fichero midir/cosas/servicios? ¿Puede posicionar la *shell* en el directorio `cosas`? ¿Por qué?

6. Más sobre la *shell*

Como ya se ha comentado, el programa típico para leer lo que escribe el usuario y lanzar los programas que pide se llama genéricamente *shell*. En UNIX se han escrito distintas *shells,* aunque todas tienen las mismas funciones básicas. Si abre un terminal, Linux lanzará una *shell* para atender sus órdenes. La *shell* que se está usando es en realidad un programa llamado `bash`. Puede observarlo usando el comando `ps` que

muestra los procesos que están ejecutando en el ordenador. Pruebe ésto:

```
$ ps
  PID TTY          TIME CMD
11486 pts/3    00:00:00 bash
11585 pts/3    00:00:00 ps
```

El comando `ps` sin opciones muestra sólo los procesos asociados a la terminal asociada a la shell. Como puede ver al lanzar `ps`, hay dos procesos, uno es el proceso `ps` propiamente dicho, el otro es el proceso que está corriendo `bash`.

`bash` no es más que un programa. Para comprobarlo intente lanzarlo. Algo así:

```
$ bash
$
```

¿Qué es lo que ha pasado? ¿Nada? Utilice el comando `ps` para comprobar que en realidad ahora hay dos *shells* corriendo. Básicamente `bash` estaba esperando comandos y se le ha ordenado que lance un segundo `bash` y lo ha puesto a funcionar. El nuevo `bash` ha tomado el control y espera órdenes mientras que el primer `bash` está esperando a que acabe el segundo, como haría con cualquier comando. Se muestran todos ellos porque están asociados al mismo terminal.

¿Cómo puede cerrar el segundo `bash` y volver al primero?

Puede probar también otras *shells* diferentes; las tradicionales se llaman `sh`, `ksh`, `csh`, `tcsh` y `bash`. Las dos últimas son los más modernas e incluyen todo lo que incluían las anteriores (`tcsh` es una evolución de `csh` y `bash` es una evolución de `sh`/`ksh`, así que hay más bien dos familias). Puede que los Linux del laboratorio no tengan instalada alguna de ellas.

Como recordará, la función principal de la *shell* es permitir lanzar programas (comandos) y comunicarse con ellos. En UNIX los programas tienen una entrada estándar de la que reciben datos llamada *stdin* y una salida estándar llamada *stdout* en la que se representan los datos que generan. Para los usuarios avanzados diremos que también hay una salida de errores diferente de la salida estándar, pero eso dejaremos que lo aprendan por su cuenta.

Por defecto, la entrada estándar de la *shell* es el teclado y la salida estándar es la pantalla (la ventana del terminal en el caso del interfaz gráfico). Cualquier proceso lanzado por otro hereda la configuración de entrada y salida estándar del padre (en realidad hereda todos sus ficheros abiertos, pero de nuevo es algo para quien tenga más interés). Gran parte de la flexibilidad de uso de programas desde la *shell* viene de que podemos cambiar o redirigir estas entradas y salidas.

Para redirigir la salida de un comando hacia un fichero podemos usar el símbolo `>` seguido del nombre del fichero detrás de un comando. Por ejemplo

```
$ date
$ date   > fichero1
$ more fichero1
$ cal  > fichero1
$ more fichero1
```

También podemos redirigir la salida con el símbolo `>>` , lo que tiene un efecto un poco diferente. Descúbralo.

```
$ date   >> fichero1
$ more fichero1
$ cal  >> fichero1
$ more fichero1
```

La entrada se redirige con el símbolo `<`. Por ejemplo, puede contar las palabras o las líneas de un fichero lanzando el comando `wc`

```
$ wc -w  < fichero1
$ wc -l  < fichero1
```

Pruebe también el comando `grep` que permite buscar líneas que contengan una cadena determinada:

```
$ grep linea
```

Este programa busca líneas que contengan la cadena `línea`. Observe que muestra automáticamente las líneas en las que lo encuentra. Consulte la ayuda de este comando ejecutando `man grep`.

Pruebe a usar el comando `grep` para encontrar todas las líneas del fichero `/etc/services` que contengan el texto "80" utilizando la redirección de entrada.

La tercera forma de redirigir la salida es encadenar la salida de un programa con la entrada del siguiente. Esto se conoce en UNIX como usar una *pipe* (tubería) y se hace con el símbolo `|`

Como veremos más adelante, esto permite encadenar programas que realizan procesados simples sobre ficheros para lograr una tarea mayor combinando procesados. De momento veamos qué se puede hacer con los comandos que conoce:

```
$ date | wc -c
$ ls -l | wc -l
$ cat /etc/services | grep web | wc -l
```

¿Qué hace el último de los comandos anteriores?

Pruebe también a redirigir a un paginador, como el comando `more` o `less`:

```
$ cat /etc/services | more
$ cat /etc/services | less
```

También se puede usar una combinación de redirecciones. Por ejemplo, podemos usar el comando `cat` para mostrar un fichero, redirigiendo la entrada y la salida. El comando `cat` en realidad lo único que haría es leer su entrada estándar y copiarla a su salida estándar. ¿Cómo copiaría un fichero con eso?

Otra utilidad de la *shell* es que permite especificar nombres de ficheros que cumplan un patrón determinado mediante el uso de comodines. Los comodines nos permiten especificar una cadena que se comparará con los nombres de ficheros y se sustituirá por todos los nombres que coincidan. Así podemos usar el comando:

```
$ ls /etc/s*
```

para listar todos los nombres de ficheros que empiezan por `/etc/s`. Los comodines simples que se permiten son los siguientes:

```
*       cualquier cadena, incluyendo la cadena vacía
?       cualquier caracter
[...]   cualquiera de los caracteres entre [ y ]
```

Consulte el resto de tipo de condiciones permitidas en la entrada del manual del comando `grep` ejecutando `man grep`. Por supuesto hay comandos que aceptan expresiones regulares, muy útil, pero tampoco vamos a entrar en eso.

Con un solo comando, liste los ficheros del directorio `/usr/lib` que empiecen por `lib`, liste también los ficheros de `/etc` que terminen en `.conf`

7. Controlando los procesos

La multitarea es una de las funcionalidades más importantes de UNIX. Todo programa que lanzamos en UNIX se ejecuta en un proceso. El proceso es el programa en ejecución. El sistema operativo tiene una tabla de procesos e identifica a cada proceso con un identificador de proceso, o PID, que es un número entero. Como ya hemos visto, el comando `ps` nos deja ver los procesos. Hasta ahora sólo lo habíamos usado para ver los procesos asociados a un terminal, pero con opciones podemos ver más cosas.

Mira el manual del comando `ps`. Dado que al comando sólo se le pasan opciones, ha evolucionado hasta no utilizar el – delante de ellas. Estas son las combinaciones más útiles. Averigüe qué hacen:

```
$ ps
$ ps ax
$ ps axu
$ ps axw
$ ps axl
$ ps axj
```

Los procesos en UNIX son lanzados por otros procesos. Se dice que un proceso se bifurca (*fork*) y crea un proceso hijo. Por tanto, los procesos se agrupan en una jerarquía de descendientes. Por ejemplo, todos los procesos que lance desde un terminal serán hijos del proceso *shell* que está usando.

Al consultar el manual de `ps` habrá visto que algunos de los comandos anteriores muestran el identificador del proceso y también el identificador del proceso padre. Puede probar también estas opciones de `ps` que dejan más clara la relación:

```
$ ps ef
```

El comando tiene multitud de opciones. Al final conocerá las que usa más habitualmente y mirará el manual si necesita algo concreto. Al menos recuerde una forma de listar todos los procesos de la máquina. Para ello practique un poco siguiendo los pasos que vienen a continuación.

Use el comando `sleep` para crear un proceso simple:

```
$ sleep numero_de_segundos
```

Este comando espera el número de segundos indicado y acaba. Pruebe a lanzar un sleep 600 en un terminal. Mientras se está ejecutando, abra un nuevo terminal y pida la tabla global de procesos. Busque el proceso ejecutando sleep.

```
$ ps axu
```

¿Cómo podría hacer un comando o secuencia de comandos que muestre una línea por cada proceso `sleep` que está corriendo en la máquina?

Compruebe que no solo está ejecutándose el sleep sino también la shell que es su padre.

Como ha visto, un proceso corriendo no devuelve el control a la *shell* hasta que termina, de forma que un proceso con un tiempo de ejecución largo que se lanza en una terminal la bloquea hasta que acabe. Esto no supone mucho problema si estamos en el modo gráfico ya que podemos lanzar más terminales (aunque nos puede llenar de ventanas el interfaz). Un proceso que está corriendo en la terminal pudiendo leer del teclado mientras la *shell* espera se dice que está corriendo en primer plano (*foreground*). No todos los procesos que están corriendo en un UNIX están en el *foreground* de un terminal. Un proceso corriendo de fondo (*background*) funciona independientemente de las acciones del usuario. Se entiende que debe estar preparado para no necesitar datos del teclado, y probablemente tampoco imprimir en una terminal. Para lanzar un proceso que no bloquee el terminal debe añadir un `&` al final del comando. Esto hará que la *shell* lo desasocie del teclado y que no espere a que termine, de modo que a continuación la *shell* mostrará su *prompt*.

Pruebe con este ejemplo:

```
$ ( sleep 30 ; echo "fin" )
```

Los paréntesis nos permiten agrupar varios procesos que se lanzan en una *shell* aparte como si fuera un comando. Mire con `ps` si quiere saber lo que está pasando. El comando esperará 30 segundos y escribirá `fin`. Puede parar el programa con `Control+c` (a veces lo verá escrito como `^C`).

Pruebe a lanzar el ejemplo con un `&` para que se ejecute en segundo plano:

```
$ ( sleep 30 ; echo "fin" ) &
 [1] 2325
$ ps
  PID TTY          TIME CMD
 2314 pts/0    00:00:00 bash
 2325 pts/0    00:00:00 bash <<<< se ha lanzado un bash para hacer el parentesis
 2326 pts/0    00:00:00 sleep  << el bash a su vez ha lanzado un proceso sleep
 2327 pts/0    00:00:00 ps
```

Verá que vuelve a tener el control del terminal y se indica el número de proceso que se le ha dado al comando (en este caso, debido a los paréntesis el el PID de una shell que los ejecuta consecutivamente). Obsérvelo con el comando `ps`.

Los procesos pueden recibir "señales" mediante el comando `kill` y permiten cierto control sobre ellos. Las señales son un tipo de comunicación entre procesos, hay muchos otros (como los sockets, para quien le suenen). El comando se llama así porque la señal que envía por defecto causa (normalmente) que el proceso que la recibe termine y sea eliminado de la tabla de procesos, pero no es necesario que sea así con todas las señales. La sintaxis básica es:

```
$ kill -nombre_de_la_señal PID
$ kill -numero_de_la_señal PID
```

Y puede interpretarse como: manda la señal indicada (bien con `nombre_de_la_señal` o con su equivalente `numero_de_la_señal`) al proceso con el identificador `PID`. Hay varias señales en UNIX y éstos son los nombres y números de las de uso más común (puede consultar el resto en el `man`):

```
1   HUP    Hung Up (literalmente colgar (el teléfono))
2   INT    Interrumpir programa
9   KILL   Parada del programa no interrumpible
15  TERM   Terminar el programa
```

La señal `HUP (1)` era originalmente para avisar de que el módem ha colgado la llamada telefónica a la que está asociado el proceso (el pleistoceno de la Internet). Dependiendo del programa, su reacción puede ser terminar, o en la mayoría de los programas que implementan servicios clásicos de red (como un servidor web o de DNS) lo que hacen es reiniciarse y/o volver a leer su fichero de configuración. Si cierra la ventana del terminal que contiene una *shell* y tal vez otros procesos asociados hará que reciban esta señal.

La señal `INT (2)` está pensada para interrumpir el programa. La mayoría de los programas la interpretan como señal de que el programa debe acabar. Algunos programas pueden interceptarla y entender otra cosa (por ejemplo `emacs` lo hace). Esta es la señal que se manda a un proceso cuando pulsamos `^C` en la consola.

La señal `KILL (9)` es una señal de que el programa va a ser eliminado de la tabla de procesos. El programa para y se elimina sin poder interceptar la señal.

La señal `TERM (15)` se emplea para indicar al proceso que termine de manera controlada (no abortiva). Es la señal que se envía por defecto con el programa `kill` si no se indica otra.

Así que normalmente se utiliza

```
$ kill pid
```

para matar un proceso y

```
$ kill -9 pid
```

para matar un proceso que se resiste al `kill` normal.

Un usuario sólo puede mandar señales a sus procesos y sólo el superusuario (`root`) es capaz de mandar señales a cualquier proceso de cualquier usuario.

Si tiene el navegador web abierto pruebe a buscar el proceso que lo controla y termínalo con un `kill`.

También puede utilizar el comando `killall` que a veces es más cómodo. Mire el manual para ver cómo funciona. Pero `killall` es también más peligroso, ¿por qué?

8. Usando la red

Quizás llegados a este punto se pregunte ¿Por qué tengo que aprender a manejar esto con comandos de texto si hay todo este interfaz gráfico? La respuesta es porque está aprendiendo UNIX por su uso en redes de comunicaciones como Internet. Ahora está

sentado delante de la pantalla del ordenador que está manejando pero no siempre va a ser así. Al usar un ordenador de forma remota el interfaz gráfico la mayoría de las veces no es una opción o es una mala opción. Por otro lado, como ya hemos comentado brevemente, los sistemas operativos UNIX se emplean hoy en día en muchos equipos que no necesitan ni tienen la capacidad de tener instalado y ejecutar todo lo necesario para un interfaz gráfico (por ejemplo, muchos routers, set-top-boxes y otros dispositivos embebidos). Finalmente, no desprecie la potencia y velocidad que puede ofrecer la línea de comandos a la hora de automatizar tareas, comparada con prácticamente cualquier interfaz gráfico. Como se suele decir: "In the beginning... was the command line".

Para finalizar esta práctica se mostrará el uso de los sistemas UNIX de dos aplicaciones sencillas pero fundamentales de red. En primer lugar, observe que tu ordenador está conectado a una red de área local (*LAN*). Ya hemos visto que la información de las cuentas en el laboratorio reside en un ordenador independiente y del mismo modo su directorio `home` está exportado por un servidor de discos (otro ordenador, aunque podría ser el mismo de las cuentas) y montado en todas las máquinas del laboratorio de forma que pueda ver sus ficheros aunque el próximo día se siente ante otro ordenador. Pero veamos un uso más claro de la red.

Averigüe el nombre en la red de su ordenador (*hostname*) con el comando

```
$ hostname
```

Seguramente será algo como `tlm43`. También puede averiguar la dirección IP configurada en su interfaz principal consultando el comando `ifconfig` o el comando `ip`:

```
$ ifconfig eth0
eth0      Link encap:Ethernet  direcciónHW 00:24:8c:b7:79:4c
          Direc. inet:10.1.1.43  Difus.:10.1.255.255  Másc:255.255.0.0
          Dirección inet6: fe80::224:8cff:feb7:794c/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:16431 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:650 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:1575272 (1.5 MB)  TX bytes:67243 (67.2 KB)
          Interrupción:18 Dirección base: 0x6c00

$ ip -f inet addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP …
    inet 10.1.1.43/16 brd 10.1.255.255 scope global eth0
```

En este caso es la dirección 10.1.1.43. En las asignaturas de redes veremos todo lo relacionado con el protocolo IP y la Internet.

Use el comando `ping` para enviar paquetes sonda a un ordenador y ver que le responde. Pruebe[2]:

```
$ ping tlm43
$ ping 10.1.1.31
$ ping <nombre de su ordenador>
$ ping <nombre del ordenador de al lado>
$ ping <dirección del ordenador de al lado>
```

[2] Los <> no los debe poner, los hemos puesto para indicar que sustituya todo eso

Esta es la herramienta básica y primera prueba que se suele hacer para saber si un ordenador (que emplea IP) en la red está encendido o no. No se preocupe ahora mismo por qué es IP, qué es una dirección, o qué envía ping, que lo iremos viendo en diferentes asignaturas.

Utilice el programa `ssh` para establecer una conexión (esto también veremos lo que es, con tiempo) con otro ordenador y obtener una *shell* de comandos en el ordenador remoto. Pruebe con el ordenador de al lado suya por ejemplo:

```
$ ssh tlm31    # poniendo el nombre de ordenador correcto
The authenticity of host 'tlm31 (10.1.1.31)' can't be established.
RSA key fingerprint is e2:5d:c2:93:bc:cf:7e:07:34:2b:18:5f:87:3d:13:dd.
Are you sure you want to continue connecting (yes/no)? Yes
Warning: Permanently added 'tlm31,10.1.1.31' (RSA) to the list of known hosts.
arss10@tlm31's password:
```

La primera vez que se conecte le pedirá que confirme la identidad del servidor. Una vez confirmada, las siguientes veces que se conecte a ese ordenador solo verificará que es el mismo, sin preguntar. Seguidamente deberá probar tu identidad con su contraseña y tendrá una sesión de *shell* remota establecida. Esto está funcionando porque en la máquina remota existe un usuario con el mismo nombre y la misma contraseña. Si no fuera así tendría que indicar otro nombre de usuario con algo como `ssh usuario@tlm31` (si eso le recuerda a una dirección de e-mail debe saber que no es casualidad). En estas condiciones, que son las normales cuando se trabaja con servidores en Internet, son útiles las habilidades con la línea de comandos que hemos visto en los primeros apartados.

Compruebe que está ejecutando programas en otro ordenador con comandos como `hostname` e `ifconfig`. Puede parecer el mismo porque tiene acceso a los ficheros de su home desde cualquiera de las máquinas del laboratorio, pero si crea ficheros en un directorio local de la máquina (como por ejemplo /tmp) verá que son diferentes discos.

Pruebe el comando `who` para ver quién más está usando ese ordenador. Compruebe haciendo `ping` al ordenador en el que está sentado que el tiempo de respuesta es mayor que desde un terminal en dicho ordenador.