Práctica 1

Introducción a Linux

Objetivos

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: 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.

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 estás en una de las mesas con armarios de prácticas. Cada grupo de prácticas dispone de varios ordenadores con Linux etiquetados como *PC A, PC B, PC C y PC SC*. Los Pcs A, B y C se usarán en las prácticas que utilicen la red y la cuenta de usuario se proporcionará cuando sea necesario. **En esta primera práctica utilizaremos el PC SC**, este PC está bien configurado en red y utiliza el sistema de cuentas central del Laboratorio de Telemática, a cada grupo de prácticas 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. Selecciona en el conmutador del teclado tu *PC SC*. Dedica unos momentos a comprobar el funcionamiento del

conmutador de teclado y pantalla que obtendrás, pulsando 2 veces la tecla de *BloqueoDesplazamiento* del teclado.

Si estás en una mesa con un solo ordenador por grupo no hace falta que te preocupes por la selección de ordenador.

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

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

Entra utilizando tu 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 tecleas y las interpreta, lanza los programas que sean necesarios para cumplirlas 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 estás usando si el usuario no elige lo contrario 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 le llamamos proceso. Así pues, cuando se escribe algo en la *shell* y se pulsa *ENTER*, se lanza un nuevo proceso con el programa que se ha introducido. La *shell* se queda esperando a que el proceso termine y vuelve a preguntar por otro comando.

El primer comando que vamos a aprender te va a permitir cambiar la contraseña de tu cuenta. Este comando en una máquina UNIX es `passwd`. Debes tener en cuenta que en el laboratorio, tu cuenta y su contraseña no se almacenan en el ordenador donde estás trabajando (como sí se suele hacer si por ejemplo instalas Linux en tu propio ordenador) sino que se hace en un servidor central. Una vez cambies tu contraseña, deberás acceder con ella sea cual sea el ordenador que uses del laboratorio. Cambia tu contraseña por una más segura que conozcas sólo tú:

```
$ passwd
Changing NIS account information for arss on bender.net.tlm.unavarra.es.
Please enter old password:      [metemos aquí la contraseña actual ]
Changing NIS password for arss on bender.net.tlm.unavarra.es.
Please enter new password:      [metemos aqui la contraseña nueva  ]
Please retype new password:     [metemos otra vez la contraseña nueva para
                                 verificar]
$
```

Observa cómo el comando nos informa de que la contraseña ha sido cambiada correctamente. En caso de que no haya sido así, lee el mensaje de error para descubrir por qué y cámbiala correctamente. Ten en cuenta que puede fallar si no se pone bien la contraseña actual o si no se escribe la misma nueva contraseña dos veces. También es posible que el comando se queje si intentas poner una contraseña demasiado fácil, depende de las políticas de seguridad que tengan establecidas los administradores. Si quieres ver este error, prueba a poner como contraseña tu nombre de usuario.

Una regla muy importante es comprobar que la contraseña se ha cambiado

correctamente **antes de salir**. Usa otro terminal para comprobar que puedes entrar con la nueva contraseña. Tras esto, sal tecleando `exit` y pulsando *ENTER*..

***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.*

Al terminar de usar una máquina UNIX debes indicar a la *shell* que ya no quieres hacer más operaciones para que termine y vuelva a pedir login al siguiente usuario. Esto es muy importante por motivos de seguridad. Si olvidas cerrar la sesión cualquiera delante de ese ordenador puede acceder a tu cuenta, lo que implica que puede borrar todos tus ficheros o usar todos los privilegios de tu cuenta. Acuérdate siempre de dejar un ordenador UNIX en la pantalla de *login* igual que cuando entraste.

Primeros pasos

Si tienes el terminal en modo texto, sal de la cuenta con *exit*. Utiliza *Control+Alt+F8* para obtener el *login* gráfico y entra en tu cuenta en modo gráfico.

Familiarízate brevemente con el interfaz gráfico de tu cuenta. Deberías ser capaz de encontrar rápidamente varias herramientas que te serán de utilidad.

Localiza un navegador web en el que puedas pedir la página de la asignatura para poder seguir las prácticas a partir de ahora.

Localiza el navegador del sistema de ficheros de UNIX en el que podrás ver de forma gráfica tu directorio *home* y moverte por los directorios y subdirectorios.

Localiza 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 deberías encontrar `gedit` o `kate`, pruébalos.

Localiza un terminal en el que puedas abrir una ventana de comandos con una *shell* con la ventaja de que puedes tener varias abiertas simultáneamente.

Pero para empezar a conocer UNIX es mas educativo aprender algunos conceptos más sobre el terminal y la *shell*. Abre un terminal con lo que obtendrás una *shell* en una ventana.

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 entrar la *shell* está esperando ordenes y está posicionada en el directorio *home* del usuario. Observa cuál es tu directorio *home* con el comando

```
$ pwd
```

Puedes crear un fichero con cualquiera de los siguientes comandos:

```
$ echo “hola” > fichero1
```

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

Usa el comando `ls` para comprobar que se han creado y los comandos siguientes para ver su contenido:

```
$ cat unfichero
$ more fichero1
$ less unfichero (para salir pulsa q)
```

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

Haz un listado de los ficheros de tu directorio en formato largo y observarás algo parecido a ésto:

```
$ 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 restantes se pueden considerar 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	Permisos de lectura	Permisos de lectura

	lectura, escritura y acceso para el propietario	y acceso para el grupo	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 letra significa una operación. 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 entenderás fácilmente que leer un directorio es ver lo que hay dentro. Escribir en un directorio es modificar la lista de ficheros que contiene pero no modificar el contenido de un fichero que está dentro.

Por otra parte el sistema operativo debe decidir qué bloque 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`. Practica un poco siguiendo las siguientes instrucciones.

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

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

```
$ vi unfichero
$ emacs unfichero
```

Los primeros son más fácil de usar para un usuario de hoy en día por su sencillez. Los dos últimos son clásicos en la historia de UNIX y son muy completos a la vez que más complejos. Normalmente utilizarás editores en el entorno gráfico pero siempre es útil saber defenderse en cualquiera de estos para las ocasiones en las que no puedas usar un interfaz gráfico (o la máquina no dispone del mismo, por ejemplo un router o casi cualquier sistema embebido). Si te quedas atascado puedes salir de `vi` pulsando `ESC` y luego escribiendo `:q!` y *ENTER*. Para salir guardando cambios escribe `:wq!` .Y de `emacs` pulsando `Control-x` y luego `Control-c` (también se suele indicar como `^X^C` o `C-x, C-y`).

También puedes usar el terminal para lanzar editores gráficos. Prueba a ejecutar:

```
$ gedit unfichero &
```

Luego entenderemos para qué sirve el `&` final

Ahora puedes volver a pensar en los permisos de los ficheros. Cambia los permisos del propietario del fichero. Podemos quitar el permiso de escritura con el comando `chmod`. Escribe:

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

Y comprueba si puedes ver el contenido y modificarlo.

Comprueba ahora 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
```

Comprueba si quitar el permiso de escritura impide borrar un fichero. Usa 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. Prueba 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
$ 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
```

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

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

¿Puedes acceder al fichero midir/cosas/servicios? ¿Puedes entrar en el directorio cosas? ¿Por qué?

Más sobre la *shell*

Como ya se ha comentado, el programa encargado de leer lo que escribe el usuario y lanzar los programas que pide se llama genericamente *shell*. En UNIX se han escrito distintas *shells* aunque todas tienen las mismas funciones básicas. Si abres un terminal Linux lanzará una *shell* para atender sus órdenes. La *shell* que se está usando es en realidad un programa llamado `bash`. Puedes observarlo usando el comando `ps` que muestra los procesos que están ejecutando en nuestro ordenador. Prueba 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 misma terminal. Como puede ver al lanzar `ps`, vemos 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 intenta lanzarlo. Algo así:

```
$ bash
$
```

¿Qué es lo que ha pasado? ¿Nada? Utiliza 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.

¿Cómo puedes cerrar el segundo `bash` y volver al primero? A estas alturas debes de

conocer al menos dos formas...

Puedes probar también otras *shells* diferentes, los tradicionales se llaman `sh`, `ksh`, `csh`, `tcsh` y `bash`. Los dos últimos que son los más modernos e incluyen todo lo que incluían los demás (`tcs` es una evolución de `csh` y `bash` es una evolución de `sh`/`ksh` así que hay más bien dos familias). Verás que tu Linux no lleva `ksh`.

Como recordarás la función principal del *shell* 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 aprendas por su cuenta cuando conozcas más UNIX)

Por defecto la entrada estándar de la *shell* y de cualquier proceso que se lance desde ella es el teclado y la salida estándar será la pantalla. Pero 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. Descúbre tú mismo...

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

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

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

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

```
$ grep linea
```

Este programa busca líneas que contengan la cadena `linea` observa que repite automáticamente las líneas en las que encuentra algo. Consulta la ayuda de este comando ejecutando `man grep`.

Prueba 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 conoces...

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

¿Qué hace el último de los comandos anteriores?

Prueba también a redirigir a un visor de pantalla completa 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 copiar un fichero redireccionando la entrada y la salida. El comando `cat` en realidad lo único que hace es leer su entrada estándar y copiarla a su salida estándar. ¿Cómo copiarías un fichero con éso?

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 que se permiten son los siguientes:

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

Consulta el resto de tipo de condiciones permitidas en la entrada del manual del comando `grep` ejecutando `man grep`.

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

Controlando los procesos

La multitarea es una de las funcionalidades más importantes de UNIX. Todo programa que lanzamos en UNIX genera 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, 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. Averigua 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 su terminal serán hijos del proceso *shell* que está usando.

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

```
$ ps ef
```

No te asustes por la multitud de opciones. Al final sabrás las que usas más habitualmente y miras el manual si necesitas algo concreto. Pero al menos recuerda una forma de listar todos los procesos de la máquina. Para ello practica un poco siguiendo los pasos que vienen a continuación:

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

```
$ sleep numero_de_segundos
```

Este comando espera el número de segundos indicado y acaba. Prueba a lanzar un sleep 600 en un terminal. Mientras se está ejecutando abre un nuevo terminal y pide la tabla de procesos y prueba a buscarlo con:

```
$ ps
$ ps axu
```

¿Cómo podrías hacer un comando que muestre una línea por cada proceso `sleep` que está corriendo en la máquina?

Como has visto, un proceso corriendo no devuelve el control a la *shell* hasta que termina, de forma que un proceso 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. En cualquier caso, un proceso que está corriendo en la terminal se dice que está corriendo en primer plano (*foreground*). No todos los procesos que están corriendo en un UNIX han sido lanzados por 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 ni esperar imprimir en la terminal. Para lanzar un proceso que no bloquee el terminal debes añadir un `&` al final del comando.

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. Mira con `ps` si quieres saber lo que está pasando. El comando esperará 30 segundos y escribirá `fin`. Puedes parar el programa con `Control+c` (a veces lo verás escrito como `^C`).

Prueba 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
```

Verás que vuelves a tener el control del terminal y se indica el número de proceso que se le ha dado al comando. Obsérvalo con el comando `ps`.

Los procesos pueden recibir señales mediante el comando `kill` y permiten cierto control sobre ellos. 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. 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 más importantes (puedes 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
```

La señal `HUP (1)` era originalmente para avisar de que el módem ha colgado la llamada al que está asociado el proceso, dependiendo del programa su reacción puede ser terminar o en la mayoría de los servicios de red lo que hacen es reiniciarse y volver a leer el fichero de configuración.

La señal `INT (2)` está pensada para interrumpir el programa. La mayoría de 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.

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 tienes el navegador abierto prueba a buscar el proceso que lo controla y deténlo con un `kill`.

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

Usando la red

Quizás llegados a este punto te preguntarás ¿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. Por otro lado, como ya hemos comentado brevemente, los sistemas 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 desprecies la potencia y velocidad que puede ofrecer la línea de comandos a la hora de automatizar tareas, comparada con prácticamente cualquier interfaz gráfico.

Para finalizar esta practica se mostrará el uso de los sistemas UNIX de dos aplicaciones sencillas pero fundamentales de red. En primer lugar observa que tu ordenador está conectado a una red de área local (*LAN*). Ya hemos visto que la información de las cuentas reside en un ordenador central y del mismo modo tu directorio `home` está exportado por un servidor de discos y montado en todas las máquinas del laboratorio de forma que puedes ver tus ficheros aunque el próximo día te sientes en otro ordenador. Pero veamos un uso más claro de la red.

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

```
$ hostname
```

Seguramente será algo así como `tlm43`. También puedes averiguar la dirección IP 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 10.1.1.43.

Usa el comando `ping` para enviar paquetes sonda a un ordenador y ver te responde. Prueba:

```
$ 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>
```

Esta es la herramienta básica y primera prueba a hacer para saber si un ordenador en la red está encendido o no. Utiliza el programa `ssh` para establecer una conexión con otro ordenador y obtener una *shell* de comandos en el ordenador remoto. Prueba con el ordenador de al lado

```
$ ssh arss10@tlm31    <<< poniendo tu usuario y 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 te conectes pedirá que confirmes 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 tu contraseña y tendrás una sesión de *shell* remota establecida. En estas condiciones, que son las normales cuando se trabaja con servidores en Internet, son útiles las habilidades de línea de comandos de los primeros apartados.

Comprueba que estás en otro ordenador con `hostname` e `ifconfig`.

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

EVALUACIÓN: test individual on-line en la web de la asignatura (http://www.tlm.unavarra.es) durante los últimos 15 minutos de clase. Se recomienda encarecidamente consultar los manuales de los comandos implicados en cada pregunta.