Programando en UNIX

Objetivos

En esta primera práctica nos familiarizaremos con el uso de las máquinas del laboratorio y de las herramientas de programación necesarias para realizar las siguientes prácticas. En las prácticas de esta asignatura programarán en C sobre un sistema operativo UNIX. Las máquinas del Laboratorio de Telemática tienen instalada la distribución de Linux Fedora Core.

Consiguiendo una cuenta

Los sistemas UNIX son multiusuario, 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 sólo 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, 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.

A cada alumno se le ha asignado una cuenta que le permite utilizar cualquiera de los PCs de propósito general del laboratorio. Esta cuenta se llamará `rc<numero>` y deberá apuntarse para obtenerla, el profesor le comunicará la contraseña en clase.

Login

Una vez conozca el nombre de la cuenta y la contraseña debe apropiarse de la cuenta que tiene asignada. Para ello debe cambiar la contraseña a una que sólo conozca cada uno.

Entre en el sistema (haga login), introduciendo el nombre de la cuenta y la contraseña en el interfaz gráfico. El ordenador arrancará el sistema gráfico con el escritorio para su cuenta. Lo primero de todo busque un navegador Web. Introduzca la dirección: http://www.tlm.unavarra.es y busque la página de esta asignatura. A partir de aquí podrá consultar los guiones de prácticas en la Web mientras los hace. Lea la práctica 1 hasta este punto.

Antes de probar a utilizar más aplicaciones debe cambiar la contraseña a su contraseña privada. Para ello siga las instrucciones:

- Lance una aplicación de terminal (también llamada de consola) que le permita darle comandos de texto al sistema operativo. Normalmente estará en el menú desplegable que puede sacar con el botón derecho en el escritorio. Elija "Open Terminal". Si no, busque en el menú de "red-hat" (abajo a la izquierda) "System tools" > "Terminal".
- Elija una contraseña nueva que proteja su cuenta. Una buena contraseña debe ser suficientemente larga y ser difícil de averiguar. Los consejos típicos son que no se usen palabras que estén en el diccionario y para eso se recomienda normalmente usar letras mayúsculas y minúsculas mezcladas con números y símbolos. También es bueno que sea fácil de recordar para su dueño para evitar escribirla, si no está escrita nadie la puede ver. Usen el sentido común y piensen una contraseña para su cuenta.
- Utilice el comando `yppasswd`, para cambiar la contraseña de su cuenta en la red de Telemática. Normalmente, el comando para cambiar la contraseña en una maquina UNIX es `passwd`. Sin embargo en un sistema como el del laboratorio, las cuentas no pertenecen a un solo ordenador sino que funcionan en varios ordenadores porque están almacenadas en un ordenador central. En estos casos puede haber otros comandos para cambiar la contraseña. En el caso del laboratorio se hace con el comando `yppasswd`

```
$ yppasswd
Changing NIS account information for lc on bender.net.tlm.unavarra.es.
Please enter old password:     [metemos aquí la contraseña actual ]
Changing NIS password for lc 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  ]

The NIS password has been changed on bender.net.tlm.unavarra.es.

$
```

Observe cómo el comando nos informa de que la contraseña ha sido cambiada correctamente. En caso de que no haya sido así, descubra por qué y cámbiela correctamente. Tenga en cuenta que puede fallar si no pone bien la contraseña actual o si no escribe la misma nueva contraseña dos veces. También es posible que el comando se queje si intenta poner una contraseña demasiado fácil.

- Pruebe a salir de la cuenta y vuelva a entrar para comprobar que la contraseña original ya no funciona y la nueva contraseña funciona.

Asegúrese que su cuenta tiene una nueva contraseña antes de continuar. 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.

Familiarícese con el escritorio

Como ha visto, sólo tiene que poner su cuenta y su contraseña y entrará al sistema en modo gráfico. En breves instantes tendrá el escritorio de Linux. El sistema gráfico de UNIX se llama X (o X-Window o X11) que es el motor que dibuja en pantalla. Sobre X puede haber un administrador de ventanas o de escritorio que es el programa que le dice a X como dibujar las ventanas y los demás elementos de interfaz de usuario. Al ser Linux de código abierto y muy modular se han escrito

varios de estos administradores del interfaz. Puede elegir entre varios (aunque conforme van evolucionando cada vez se parecen más) los nombres de los principales entornos gráficos son Gnome y KDE. Buscando un poco podrá elegir entre esos y algún otro. Supondremos que ya tiene experiencia en moverse por un escritorio gráfico con un ratón, así que experimente por su cuenta. Pero debe buscar y aprender a hacer al menos las siguientes cosas:

1. Ya ha probado como lanzar un terminal. Un terminal es una aplicación que le permite abrir una sesión de comandos. El terminal lanza un programa llamado shell que le permite escribir comandos o lanzar programas. Una vez que consiga ésto, puede escribir los comandos que ya conoce y muchos otros que aprenderá. Aunque vaya a usar las X en la mayoría de las prácticas necesitará seguir escribiendo comandos.
2. Localice, si no lo ha hecho ya, un navegador de Web.
3. Busque un editor de texto, los clásicos son `vi` o `emacs` pero quizás le parezca que tienen un aspecto anticuado, siguen siendo útiles porque pueden usarse en un interfaz de texto sin gráficos. Puede probarlos lanzándolos desde un terminal:

```
$ vi nombredefichero
$ emacs nombredefichero
```

   Si prefiere algo más fácil, hay cientos de editores donde probar. Busque por ejemplo `gedit` (del Gnome), `kedit` (del KDE), aunque por aquí preferimos `nedit`. Elija el que más le guste.

4. En cualquier caso sepa que puede lanzar los editores desde el terminal lo que es útil si está escribiendo comandos y quiere abrir un fichero desde el terminal solo tiene que escribir el nombre del editor seguido del fichero:

```
$ nedit nombredefichero
```

   El editor es sin más un programa que puede abrir ventanas. Con eso puede editar el fichero de forma más cómoda. Sin embargo fíjese que mientras el programa `nedit` está funcionando la shell se ha quedado esperándole, con lo que no puede hacer mas comandos en esa shell. En la siguiente práctica veremos mas sobre esto pero de momento puede usar el comando con un & detrás para que la shell no espere a que acabe el programa.

```
$ nedit nombredefichero &
```

5. Finalmente puede buscar un explorador gráfico del sistema de ficheros y buscar su directorio Home (en UNIX se llama así al directorio asignado a cada usuario que es el único en el que debería poder guardar cosas). Puede usarlo también para copiar y mover ficheros. En cualquier caso fíjese que tiene su directorio Home y que dentro de él tendrá un directorio llamado Desktop que corresponde a lo que tiene en el escritorio.

Compilando y usando make

A continuación nos familiarizaremos con las herramientas de programación en Linux, el uso de `gcc` y `make`. En el proceso se practicará con la programación en C en un sistema operativo UNIX, en especial la programación de herramientas de línea de comando que utilicen argumentos.

En primer lugar escriba un programa en C sencillo que escriba algo en pantalla y guárdelo en un fichero `programa.c`

Ejemplo de `programa.c`

```
#include <stdio.h>

int main() {
    printf("primer programa de RC... \n");

    return 0;
}
```

Para compilar el programa en C utilice el comando `gcc` que incluye las funciones de compilador y de linker. Haciendo:

```
$ gcc programa.c -o programa
```

Le indica al compilador que compile el programa y lo enlace con las librerías necesarias, construyendo un ejecutable. Si el compilador no da ningún error obtendrá un ejecutable en su directorio llamado `programa`. Puede ejecutar este programa lanzándolo desde el terminal con:

```
$ ./programa
primer programa de RC...
$
```

Pero normalmente utilizaremos el programa `make` para ayudarnos a organizar las fuentes de un programa y a compilarlo con una sola instrucción. Para ello hay que colocar un fichero con nombre `Makefile` (o también vale en minúsculas `makefile`) en el mismo directorio que las fuentes de nuestro programa. El fichero makefile contiene las instrucciones para compilar el programa a partir de las fuentes. Veamos por ejemplo cómo compilar el programa anterior con `make`. Se recomienda usar `make` para organizar la compilación de los programas de prácticas. En cualquier caso las prácticas que haya que presentar se presentarán en forma de un directorio en el que deberá haber un makefile para construir el programa previamente a probarlo.

Haga un directorio para el programa de ejemplo y coloque en ese directorio el fichero con las fuentes `programa.c` anterior. Construya un fichero makefile con el siguiente contenido:

Ejemplo de `Makefile`

```
# En los ficheros makefile puede haber comentarios
# los comentarios se ponen con líneas que empiezan por #
```

```
programa: programa.c
    gcc programa.c -o programa

# El programa make utiliza los tabuladores para indicar información
# en la línea anterior con gcc el espacio debe estar hecho con el tabulador,
# no vale poner espacios para llevar la línea a la derecha
```

Con el fichero `Makefile` y el fichero `programa.c` en el mismo directorio puede construir el programa ejecutable con sólo ir a ese directorio y hacer `make`. Obtendrá algo parecido a esto:

```
$ make
gcc programa.c -o programa
$ ls
Makefile  programa  programa.c
$
```

Puede verse que el programa `make` ha leído del fichero makefile la descripción de como compilar el programa y lo ha compilado. Obteniendo el ejecutable `programa`. Puede parecer que no es una gran ganancia escribir sólo `make` en lugar de escribir `gcc programa.c -o programa`. Pero conforme los programas van haciéndose mas complejos la construcción incluirá compilar varios módulos y unir los módulos entre si o con librerías externas para generar uno o varios ejecutables. Make nos permite automatizar todas estas funciones para compilar con órdenes simples.

Comprendiendo el `makefile`

En el `makefile` básicamente se explica cómo construir unos ficheros a partir de otros. Las líneas sin tabulador indican de qué ficheros depende un fichero que se va a construir. Por ejemplo en el caso anterior el fichero `programa` depende del fichero `programa.c`. Después de una línea de dependencias vienen una serie de líneas con tabulador que indican como hay que hacer para construir el fichero de la izquierda, a partir del fichero (o los ficheros) de la derecha de los dos puntos. En nuestro caso las líneas:

```
programa: programa.c
    gcc programa.c -o programa
```

Indican que para construir el fichero `programa` a partir de `programa.c` hay que ejecutar una sola línea que es el `gcc`. El `make` ejecuta la línea si ve que el fichero `programa.c` ha sido modificado más recientemente que `programa`. Puede observar que si hace `make` varias veces, a la segunda le indicará que el programa ya está compilado y no hará nada. Podemos forzar la recopilación borrando el programa ejecutable. También se puede obligar a recompilar haciendo cualquier cambio en el fichero fuente de forma que sea más nuevo que el ejecutable. También existe el comando `touch` que modifica la fecha de modificación de un fichero como si hubiera sufrido un cambio. Puede hacer `touch programa.c` lo que provocará que a make le parezca que el fichero ha sido modificado y recompile.

Al hacer `make` se intenta construir la primera dependencia que haya en el `makefile`. El `makefile` puede contener varias descripciones de dependencias. Normalmente sólo se construye la

primera, pero se le puede indicar a `make` qué dependencia queremos construir en la línea de comandos

Ejemplo:

```
# para construir la primera dependencia
$ make
# para construir una dependencia concreta
$ make programa
$ make programa2
$
```

También se pueden hacer que las dependencias no construyan un fichero concreto sino que sean simples claves para decirle a `make` qué construir. Y también puede ser que los ficheros de los que dependa una clave dependan a su vez de otros ficheros. En ese caso make resolverá todas las dependencias necesarias hasta que pueda construir el programa. Pruebe por ejemplo este makefile:

Ejemplo de otro `Makefile` más complicado

```
# Makefile

programa: programa.c
    echo "Compilando el programa programa.c"
    gcc programa.c -o programa

programa2: programa2.c
    echo "Compilando el programa programa2.c"
    gcc programa2.c -o programa2

todos: programa programa2
```

Escriba un programa en el fichero `programa2.c` y pruebe que hace el makefile anterior si hacemos los siguientes comandos:

```
$ make programa
$ make programa2
$ make todos
$ make
```

¿Cómo haría que se construyan todos los programas al hacer sólo `make`?

Compilando librerías

En ocasiones los programas se organizan en forma de diversos módulos de forma que pueden compilarse por separado diferentes funciones. El compilador nos permite obtener código objeto que se almacena en ficheros `.o`. El código objeto consiste en funciones ya compiladas que pueden añadirse al crear un ejecutable. Estos código objeto, especialmente cuando se agrupan para dar una funcionalidad en común se suelen llamar librerías.

Construyamos una librería simple con una sola función. Cree un nuevo directorio para un nuevo programa de ejemplo y escriba un fichero `mifuncion.c` donde defina una función simple.

Por ejemplo:

Ejemplo de fichero `mifuncion.c`

```
/* Los 3 includes siguientes son en general recomendables */
/* en todos los programas que vayan usar ficheros */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void hazunfichero( char *nombre ) {
    FILE *f;
    f = fopen( nombre , "w" );
    if ( f == NULL ) {
        printf("No puedo abrir el fichero %s\n",nombre);
        return;
    }

    fprintf(f,"Fichero %s creado por la función hazunfichero()\n",nombre);
    fclose(f);
}
```

Aparte del fichero definiendo la función, se debe realizar un fichero de cabecera, al que llamaremos normalmente `mifuncion.h` que describirá la función, los tipos de datos que acepta y que devuelve. En nuestro caso sería simplemente:

Ejemplo de fichero `mifuncion.h`

```
/* La función hazunfichero crea un fichero con el nombre indicado  */
/* El comentario no es obligatorio pero es recomendable            */

void hazunfichero( char *nombre );
```

Para escribir un programa que use la función, basta con incluir el fichero de header y llamar a la función desde el código. Por ejemplo el siguiente programa:

Ejemplo de fichero `creaficheros.c` que utiliza mifuncion.h

```
#include <stdio.h>
#include "mifuncion.h"

int main() {
    printf("Voy a construir unos ficheros...\n");
    hazunfichero("fichero1");
    hazunfichero("f-dos");
    hazunfichero("otro");
    return 0;
}
```

Para compilar este programa hay varias maneras. Podemos compilar directamente todos los `.c` y generar el ejecutable:

```
$ gcc creaficheros.c mifuncion.c  -o programa
```

Pero la ventaja de las librerías es que normalmente pueden ser programadas por otras personas y hay que compilarlas independientemente. Por ejemplo podemos compilar la función por separado. No podemos generar un ejecutable porque `mifuncion.c` no tiene función `main()` por la que empezar. Pero podemos generar un fichero con la función ya compilada con el siguiente comando:

```
$ gcc -c mifuncion.c
$ ls
mifuncion.c mifuncion.o
```

Esto nos genera el fichero `mifuncion.o` que contiene el código de la función compilado, listo para ser incorporado a otros programas. El fichero `mifuncion.o` junto con el fichero `mifuncion.h` permite a un segundo programador incorporar la función en su programa sin necesidad de tener el código fuente. Para eso basta con que incluya el fichero `mifuncion.h` con el #include y al compilar le indique al `gcc` que ponga el código de la función en el ejecutable:

```
$ gcc creaficheros.c mifuncion.o  -o programa
```

Este comando hace prácticamente lo mismo que el anterior pero la función no se compila sino que se añade la que ya esta compilada. Así, podríamos separar la creación del programa en dos fases en el makefile:

Ejemplo de `Makefile` que construye la librería por separado

```
# Makefile

programa: creaficheros.c mifuncion.o
    echo "Compilando el programa final"
    gcc creaficheros.c mifuncion.o -o programa

mifuncion.o: mifuncion.c
    echo "Compilando la función"
    gcc -c mifuncion.c
```

Haciendo `make` primero se construirá la función y luego se unirá. Esto también tiene la ventaja, en programas grandes,de que si hacemos un cambio en el programa pero no cambiamos la función, al hacer `make` sólo se recompila el programa pero no se reconstruye el fichero `mifuncion.o` porque no se ha modificado su código fuente.

Por supuesto `gcc` y `make` tienen muchas más posibilidades. Pero con estas funcionalidades básicas se pueden hacer muchas cosas y en concreto todas las que se piden en esta asignatura. Si quiere más información sobre ellos puede utilizar el comando `man` o consultar la documentación existente online.

Empezando a programar en UNIX

Como ha visto ya puede programar herramientas que sean comandos de UNIX. En las siguientes prácticas programaremos comandos que permitan utilizar la red. La mayoría de los comandos necesitarán argumentos de entrada para indicarles diferentes opciones.

Escriba el programa de ejemplo y compruebe su funcionamiento:

Ejemplo de `argumentos.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int i;
    printf("Has escrito %d argumentos \n",argc);
    for (i=0;i<argc; i++) {
        printf("El argumento %d es: %s\n",i,argv[i]);
    }

    return 0;
}
```

Compílelo y pruebe los resultados con los siguientes ejemplos:

```
$ ./argumentos 1 dos 3 4 -5
$ ./argumentos cuantos argumentos "hay aquí"
$ ./argumentos
$ ./argumentos -a
```

Como puede ver, en UNIX a la función main se le pasan dos argumentos. El primero `argc` es un entero que indica cuantos elementos tiene el segundo que es un array. El segundo `argv` es un array de punteros a char. Cada puntero apunta a una cadena con uno de los argumentos. El primer argumento `argv[0]` no es un argumento sino más bien el nombre del comando. Se puede utilizar para saber en que directorio esta el programa que se ha lanzado pero en general lo ignoraremos. El resto de elementos desde `argv[1]` hasta `argv[argc-1]` son las cadenas de texto que venían después del comando.

Así que si el dato que queremos leer es una cadena es fácil de extraer de los argumentos. Por ejemplo en el siguiente programa se comprueba que hay al menos un primer argumento y se utiliza ese argumento como nombre de fichero para escribir los demás argumentos en ese fichero en lugar de en la pantalla.

Ejemplo de programa que usa un argumento tipo cadena de texto.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int i;
    char *filename;
    FILE *f;

    if (argc<2) {
        exit(0);
    }

    filename=argv[1];

    f = fopen(filename,"w");
    if (f==NULL) {
        printf("No puedo abrir el fichero %s\n",argv[1]);
    }


    fprintf(f,"Has escrito %d argumentos \n",argc);
    for (i=0;i<argc; i++) {
        fprintf(f,"El argumento %d es: %s\n",i,argv[i]);
    }

    return 0;
}
```

Sin embargo, a veces querrá utilizar argumentos numéricos. En ese caso si le pasa un número al comando. Por ejemplo haciendo

```
$ repite 3 hola
hola
hola
hola
$
```

En la variable `argv[1]` el contenido apuntará a la cadena "3" y no a un entero con el valor 3. Debe obtener la variable entera haciendo a C que convierta o que lea la cadena. Para eso quizás haya usado la función `atoi()` y similares (`atoi("3")` devuelve el entero 3). Pero es más cómodo usar la función `sscanf()`. Se usa como `scanf()` pero para extraer datos de una cadena (`sscanf("3","%d",&n)` intenta leer la cadena "3" como si fuera un entero por usar el %d y pone el resultado en la variable n)

Pruebe como ejercicio a construir el programa repite. Que funcione como el del ejemplo anterior. O sea que cumpla la especificación:

```
FORMATO

    repite <n> <cadena>

DESCRIPCION

Escribe por pantalla <cadena> tantas veces como indique <n>
```

Leyendo del teclado

Frecuentemente, en las prácticas de red necesitaremos leer líneas escritas por teclado por el usuario. Para ello lo mas recomendable es usar la función `fgets()`. Lea su funcionamiento usando `man fgets`. La función `fgets( buf, n, f )` lee la siguiente línea disponible en el fichero indicado por `f` y la coloca en la zona de memoria indicada por `buf`. La zona de memoria habrá sido construida con un array o un `malloc` y tendrá un tamaño finito por lo que se le indica en `n` el máximo tamaño que puede tener la línea.

Recuerde que en todos los lugares donde deba pasar un fichero (`FILE*`) puede utilizar stdin para indicar la entrada de texto y `stdout` para enviar la salida a la pantalla.

Pruebe por ejemplo el siguiente programa que va leyendo líneas de texto del teclado y guardándolas en un fichero.

Ejemplo de programa que lee el teclado.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int i;
    char *filename;
    FILE *f;
    char buf[256];

    if (argc<2) {
        filename="out";
    } else {
        filename=argv[1];
    }

    f = fopen(filename,"w");
    if (f==NULL) {
        printf("No puedo abrir el fichero %s\n",argv[1]);
    }

    printf("> ");
    while ( fgets(buf,255,stdin) != NULL ) {

        fprintf(f,"%s",buf);
        printf("> ");
    }
    return 0;
}
```

Escriba un programa que cumpla la siguiente especificación:

```
FORMATO

    escribe <nombredefichero> <n>

DESCRIPCION

<n> debe ser un numero mayor que 0. El programa lo comprueba y sale con un error
en caso de que sea otra cosa.
El programa debe quedarse esperando una línea de la entrada de teclado.
Una vez leída la escribirá en el fichero dado por <nombredefichero>
tantas veces como indique <n>, numerando las líneas en el proceso

Ejemplo:
$ ./escribe out 4
hola
$ more out
1: hola
2: hola
3: hola
$
```

A lo largo de los distintos ejemplos de esta práctica se han visto los conceptos de programación de comandos UNIX que necesitaremos para programar las prácticas con la red pero que no son estrictamente programación de redes. Puede utilizar estos ejemplos como base cuando realice programas que deban leer parámetros o utilizar la entrada de teclado.

No es necesario entregar nada en esta práctica.