Hoja de Ejercicios 1: Interfaces del Sistema

Objetivos

Estudiar los principales interfaces para interactuar con el sistema operativo:

- **Interfaz de línea de comandos,** se revisan los conceptos básicos del manejo de un sistema operativo UNIX. El objetivo principal será el manejo de la *shell* y sus características. Además se verán algunas de las herramientas básicas y una breve introducción a la programación en *shell script*.
- **API de un sistema Unix** y su entorno de desarrollo. En particular, se estudiarán las convenciones de llamadas al API de sistema y librería; y funciones específicas para la gestión errores y obtener información de usuarios.

Contenidos

Preparación del entorno para la práctica

Interfaz de línea de comandos

- **La shell bash**
  - La línea de comandos
  - Páginas de manual
  - Comandos y secuencias de comandos
  - Variables de entorno
  - Manejo de cadenas y flujos de caracteres
- **El sistema de ficheros**
  - Búsqueda de ficheros
- **Redirecciones, Tuberías y Expresiones regulares**
- **Programación en lenguaje shell**
- **Proyecto: Agenda en Shell script.**

API del Sistema Operativo

- **Compilación y depuración de programas.**
- **Gestión de errores**
- **Librerías y Llamadas al Sistema**

Ejercicios Prácticos

Todos los ejercicios de esta hoja son prácticos y requieren el entorno de usuario básico (shell) y de desarrollo (compilador, editores y depurador). Este entorno está disponible en las máquinas virtuales de la asignatura y en la máquina física del laboratorio.

Interfaz de línea de comandos

La *shell* bash

La línea de comandos

Ejercicio 1. Arrancar un terminal y comprobar el *prompt* de la *shell*. Cada usuario tiene un *prompt* distinto configurable. Para cambiar de usuario podemos usar el programa su. Ejecutar `su -` y comprobar si cambia el *prompt*, regresar al usuario inicial con la orden `exit`. El comando `exit` termina la ejecución de la *shell* y devuelve opcionalmente un código de retorno.

Páginas de manual

Ejercicio 1. Consultar la página de manual de man (`man man`) . Especialmente las secciones de una página de manual (`NAME, SYNOPSIS, DESCRIPTION`...).

Ejercicio 2. Para buscar en una sección determinada simplemente se puede especificar antes del nombre de la página. Por ejemplo, ver la diferencia entre `man 1 time` y `man 2 time`.

Ejercicio 3. Para buscar páginas de manual sobre un tema en particular se puede usar la opción -k, o las órdenes `whatis` o apropos. Buscar las órdenes relacionadas con la "hora" (`time`). **Nota:** Este comando accede a una base de datos sencilla para cada página de manual. La base de datos se gestiona con el comando `mandb`.

Comandos y secuencias de comandos

Ejercicio 1. Estudiar el funcionamiento del comando `echo` (`man echo`). ¿Qué hacen las opciones `-n` y `-e`? Imprimir la frase "`Curso de Sistemas Operativos`" con un tabulador al principio.

Ejercicio 2. Los comandos se pueden partir en varias líneas terminando cada una con '`\`'. Imprimir la frase del ejemplo anterior, escribiendo cada palabra en una línea. Comprobar que el *prompt* de la shell cambia en el modo multi-línea.

Ejercicio 3. La shell dispone de una serie de *meta-caracteres* que permiten controlar su comportamiento. En especial: `|| && ; ( ) | &`, sirven para generar secuencias o listas de comandos. Estudiar el funcionamiento de las siguientes secuencias:

```
echo línea uno;echo línea dos; echo línea tres
echo línea uno && echo línea dos && echo línea tres
echo línea uno || echo línea dos; echo línea tres
(echo En una sub-shell; exit 0) && echo acabó bien || echo acabó mal
(echo En una sub-shell; exit 1) && echo acabó bien || echo acabó mal
```

Variables de entorno

Ejercicio 1. Consultar el entorno de la shell mediante env.

Ejercicio 2. El valor de las variables se puede acceder con el prefijo $. Consultar el valor, y determinar el significado de: `USER`, `UID`, `HOME`, `PWD`, `SHELL`, `$`, `PPID`, `?`, `PATH`. Ejemplo:

```
(exit 0);echo $?;(exit 4);echo $?
echo $$ $PPID
ps -p $$ -o "pid ppid cmd"
```

Ejercicio 3. Fijar el valor de las variables `VARIABLE1, VARIABLE2 y VARIABLE3` a "`Curso`", "`de`" y "`Sistemas Operativos`", respectivamente. Imprimir la frase "`Curso de Sistemas Operativos`" accediendo a las variables.

Entrar en otra shell, ejecutando `bash`, repetir la orden que imprime la frase ¿Cuál es el resultado? Hacer que la frase se imprima correctamente en la nueva shell definiendo las variables con el comando `export`.

Ejercicio 4. Las colisiones entre variables se pueden evitar poniendo el nombre de la variable entre llaves {}. Fijar la variable `VAR1` a "`1+1=`" y `VAR12` a "`error`". ¿Qué imprime echo `$VAR12`? ¿Cómo se imprimiría "`1+1=2`" (sin espacios) usando el valor de `VAR1`?

Ejercicio 5. La variable `PS1` guarda la estructura del *prompt*. Consultar la sección `PROMPTING` en la página de manual de bash y cambiar el *prompt* a "`12:39|~% `" (hora | directorio %).

Ejercicio 6. `PATH` es una de las variables más importantes del entorno de la *shell*. Consultar su valor y añadir el directorio actual al `PATH`. ¿Qué diferencia hay entre añadir `./` y `$PWD`?

Ejercicio 7. La orden `which` determina qué comando se ejecutará cuando se usa en la línea de comandos. Determinar qué fichero se usa para ejecutar `bash, cd, echo, y test`. De forma similar `type` sirve para identificar como la shell interpreta el comando ¿cuáles de los anteriores comandos son *built-in*?

Manejo de cadenas y flujos de caracteres

Ejercicio 1. Estudiar el comando `sort` (`man sort`). Ordenar las palabras: zorro, pájaro, vaca, caballo y abeja, imprimiendo cada una en una línea (`\n`) con el comando `echo` y encadenando su salida (tubería '|') con el comando `sort`.

Ejercicio 2. Escribir las palabras anteriores en un fichero (`texto1`) usando el comando echo y la redirección ('>'). Repetir el ejercicio anterior usando en este caso el fichero `texto1` y no la entrada estándar del comando sort.

Ejercicio 3. El comando `cat` muestra el contenido de un fichero, probar con `texto1`. Además, si no se especifica un fichero (o se usa '-', ver `man cat`), recoge la entrada estándar que puede redirigirse a un fichero. Escribir las palabras: pera, manzana, plátano y ciruela, cada una en una línea, en el fichero texto2. **Nota:** el flujo de entrada estándar se cierra con Ctrl+d. Comprobar el contenido de `texto2` con `cat`.

Ejercicio 4. cat se puede usar para concatenar ficheros si se especifican como argumentos. Unir los ficheros texto1 y texto2 en texto3 usando la redirección ('>').

Ejercicio 5. wc sirve para contar palabras, caracteres y líneas de un fichero de texto. Comprobar el tamaño de los ficheros texto1, texto2 y texto3 con wc, y compararlo con la salida de ls -l *. Consultar la página de manual para ver como controlar la salida del comando.

Ejercicio 6. Los comandos head y tail permiten ver el principio y el final de un fichero. Vamos a usar dmesg (mensajes del sistema) para probar el funcionamiento de estos comandos:

- En primer lugar determinar el número de líneas que produce el comando.

- Usar tail para quedarse con la última parte (ej. últimas 15 líneas)
- Usar head para quedarse con las primeras líneas (ej. primeras 3 líneas)
- Una opción importante de tail es -f que permite mostrar de forma continua las últimas líneas del fichero. Comparar con la opción -F.

Ejercicio 7. El comando `tr` sirve para cambiar caracteres (*tr*anslate) o eliminarlos. Poner todas las palabras del fichero texto1 en una línea sustituyendo el carácter fin de línea por un tabulador.

El sistema de ficheros

Ejercicio 1. Para cambiar de directorio se usa la orden `cd`.

- Cambiar al directorio de usuario (cd sin argumentos).
- Averiguar la ruta (*path*) del directorio (comando `pwd`, o variable `PWD`).
- Pasar al directorio `/usr/bin`, y comprobarlo con `pwd`.
- Volver al directorio anterior (`cd -`).
- Cambiar ahora de nuevo al directorio `/usr/bin` pero de forma relativa desde el directorio de usuario.
- Volver finalmente al directorio de usuario

Nota: El . hace referencia al directorio actual y .. hace referencia al directorio directamente superior

Ejercicio 2. La orden `mkdir` sirve para crear directorios. Hacer el directorio '`mis_archivos`' en el directorio de trabajo. Comprobar su creación con el comando `ls` (***list***).

Ejercicio 3. La opción `-p` de `mkdir` sirve para crear un directorio y todos aquellos que sean necesarios en la ruta. Crear el directorio `mis_archivos/prueba/texto/tmp`, probar con la opción -p y sin ella (`man mkdir`).

Ejercicio 4. Un directorio se puede borrar con `rmdir`, pero debe estar vacío. Eliminar el directorio prueba (y todos sus contenidos con `rmdir`).

Ejercicio 5. Copiar el fichero `texto1` a `copia1` con `cp`. Los ficheros y directorios se pueden renombrar con el comando mv, renombrar `copia1` a `fichero1`.

Ejercicio 6. Crear un directorio llamado '`copia`' y copiar en él los archivos `texto1`, `texto2` y `fichero1` con la orden `cp` en una única línea.

Ejercicio 7. Los directorios se pueden copiar usando la opción -r. Copiar el directorio `copia` al directorio `otra_copia`. Consultar el propósito de las opciones `-f` y `-i` en la página de manual de `cp`.

Ejercicio 8. Para borrar usar el comando rm. Borrar los ficheros del directorio `copia` con rm, y finalmente el propio directorio copia con rmdir. Los comandos `cp` y `rm` aceptan opciones recursivas (`-r`) para copiar y borrar directorios respectivamente. Borrar los directorios `otra_copia` con `rm -r`.

Búsqueda de ficheros

Ejercicio 1. La forma básica de uso de find es `find <ruta_de_búsqueda> -name <patrón_nombre>`. Por ejemplo buscar todos los ficheros que empiezan por `texto` en `/home`.

Ejercicio 2. Se puede además buscar por tipo usando la opción type. Consultar la página de manual de find para buscar todos los directorios en `$HOME`.

Ejercicio 3. La opción `size` permite buscar por tamaño con los modificadores c, b, k, M y G. Si se

antepone + al tamaño se seleccionen ficheros mayores, con - se seleccionan ficheros más pequeños
Buscar ficheros mayores de 10M en el sistema.

Redirecciones, Tuberías y Expresiones regulares

Ejercicio 1. Ejecutar en el directorio $HOME el comando ls -l text* nada* > salida. ¿Qué sucede?, ¿cuáles son los contenidos del fichero salida?. Con el comando anterior redirigir la salida estándar a `salida.out` y la salida estándar de error a `salida.error`.

Ejercicio 2. El operador de redirección (>) 'trunca' el contenido del fichero. Se puede añadir al contenido ya existente mediante (>>) . Repetir el comando anterior pero sin borrar el contenido de los ficheros.

Ejercicio 3. Se pueden *duplicar* los descriptores con >& y >>&, en la forma: `comando > salida 2>&1`. Redirigir la salida estándar y de error del comando ls anterior al fichero `salida.txt`. Comprobar que el comportamiento es distinto a `comando 2>&1 > salida` (las "duplicaciones" se hacen secuencialmente)

Ejercicio 4. Muchas veces no resulta interesante la salida del comando, sólo su resultado ($?); esto se consigue redirigiendo las salidas a /dev/null. Probar el funcionamiento con las órdenes anteriores.

Ejercicio 5. También es posible redirigir la entrada estándar. Comparar la ejecución de las secuencias:

- `cat texto1 | sort`
- `sort < texto1`

Ejercicio 6. Usando el comando grep y la una expresión regular adecuada, buscar en el fichero texto1 las siguientes palabras:

```
$ cat texto1
zorro
pájaro
vaca
caballo
abeja
```

- las palabras con 'ja'
- las palabras que terminan en 'ja'
- las palabras que tiene dos aes con una letra en medio (aba,aca,aaa,ala,...)
- el patrón alo ó allo (la l se repite una o dos veces)

Programación en lenguaje shell

Importante: Para ejecutar un script debe tener permisos de ejecución, que se pueden otorgar con el comando `chmod +x <ruta_del_script>`

Ejercicio 1. **Argumentos de un programa**. Escribir un script que muestre el nombre del fichero del script ($0), el primer ($1) y segundo ($2) argumento, cada uno en una línea.

Ejercicio 2. **Sentencias Condicionales.** Escribir un programa que acepte exactamente un argumento. Si no es así debe terminar con código 1 y mostrar el mensaje correspondiente. El argumento se interpretará como una ruta. Si la ruta es un fichero regular (ver `man test`), el script mostrará el número de líneas que tiene.

Nota: La ejecución de un programa se puede asignar a una variable que guardará la salida estándar. Además el comando `cut` puede ser útil para separar una cadena y seleccionar un campo.

```
$./ejercicio2.sh /etc/passwd
El fichero /etc/passwd tiene 22 líneas
$ ./ejercicio2.sh /etc/pas
El fichero /etc/pas no existe o no es regular
$ ./ejercicio2.sh
Uso ejercicio2.sh <ruta>
```

Ejercicio 3. **Bucles.** Escribir un *script shell* que itere por cada fichero de un directorio dado (argumento del script) y si es un fichero regular muestre en un línea el nombre y el número de palabras.

Nota: Para realizar un número de iteraciones dado se puede usar la forma C (ver manual de bash) o el comando `seq`.

Ejercicio 4. **Funciones.** Para escribir una función en bash se usa la siguiente sintaxis. Probar la ejecución del siguiente script. ¿Qué valor se almacena en la variable A?

```
#!/bin/bash
function hola(){
  echo “Hola $1!”
}

hola mundo
A=`hola mundo`
echo “el valor de A es ${A}”
```

Proyecto: Agenda en Shell script.

Escribir un programa que sirva de agenda. El programa almacenará los datos de la agenda en un base de datos en plano, cada línea del fichero será un registro con el formato:

```
nombre:teléfono:dirección de mail
```

La agenda dará las opciones de *listar*, para mostrar todos los registros formateados; *añadir* un registro, *borrar* un registro identificando por el campo nombre; *buscar* un registro por un patrón aplicado a cualquier campo y *salir* del programa. El siguiente ejemplo muestra una ejecución:

```
$ ./agenda.sh
1) listar
2) buscar
3) borrar
4) añadir
5) salir
```

```
#? 4
Nombre: juan
Teléfono: 2345
Mail: juan@ucm.es

#? 4
Nombre: maría
Teléfono: 2345
Mail: maria@ucm.es

#? 2
Buscar: juan
Nombre: juan
Teléfono: 2345
Mail: juan@ucm.es

#? 1
Nombre: juan
Teléfono: 2345
Mail: juan@ucm.es

Nombre: maría
Teléfono: 2345
Mail: maria@ucm.es
```

Notas sobre la implementación:

- Usar una función que imprima el menú y ejecute cada acción. La función debe usar las construcciones `select` y `case`. Ejemplo:

```
select opt in "Ficheros" "Fecha" "Salir"
do
    case $opt in
        "Ficheros")
            ls -l
            ;;
        "Fecha")
            date
            ;;
        "Salir")
            echo "Exiting..."
            break
            ;;
        *)
            echo "Error en opción"
            ;;
    esac
done
```

- Las acciones de la agenda se pueden implementar con las órdenes explicadas en la práctica.

API del Sistema Operativo

Compilación y depuración de programas.

Ejercicio 1. Considerar el siguiente programa:

```
#include <stdio.h>

char * init_string() {
    char s[] = "Esto es una cadena";
    return s;
}

int main() {
    char * msg = init_string();
    printf("Cadena: %s\n", msg);
    return 0;
}
```

- Compilar el programa con las opciones `-g` (símbolos de depuración) y `-o` (especificar salida). Ejemplo: `gcc -g -o error error.c`. Observar los posibles avisos por parte del compilador (`warnings`). Ejecuta el programa y comprueba su funcionamiento.

- Depura el programa, con el comando `gdb ./error` (observa cómo cambia al *prompt* interactivo del depurador):
    - Consulta el código del programa, **`list`**
    - Añade un punto de parada en la primera línea del programa, **`break 10`**
    - Ejecuta el programa, **`run`**
    - Consulta el valor de la variable `msg`, **`print msg`**
    - Entra en la función (**`step`**) y avanza hasta la sentencia `return s` (**`next`**). Consulta el valor de s antes de retornar, **`print s.`**
    - Continúa trazando el programa hasta la línea printf("..."). Consulta el valor de msg en ese momento.

- Identifica el problema y arregla la función `init_string` para que el código muestre el resultado esperado.

Gestión de errores

Ejercicio 2. Considera el siguiente programa:

```
#include <fcntl.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    int fd = open(argv[1], O_RDONLY);
    /* Gestionar el error de open */
    return 0;
```

```
}
```

Añadir el código necesario para gestionar correctamente los errores generados por `open(2)` con `perror(3)`. Consultar en el manual como `open(2)` señaliza el error y los posibles errores. Comprobar que se identifica correctamente el error en los siguientes casos:

- No se especifica argumento: `./mi_open`
- Se usa una ruta de un fichero que no existe: `./mi_open /no/existe`
- Se usa la ruta de un fichero sin permiso de acceso: `./mi_open /etc/shadow`

Ejercicio 3. Imprimir el código numérico de error generado por la llamada del código anterior (`errno` definido en `errno.h`) y el mensaje asociado obtenido con `strerror(3)`.

Librerías y Llamadas al Sistema

Ejercicio 1. Consultar la página de manual del comando `id(1)` y comprobar su funcionamiento.

Ejercicio 2. Escribir un programa que (`mi_id`) que reciba un único argumento que será el ID de un usuario del sistema. Usando la llamada a librería `getpwuid(3)` imprimir el nombre de usuario, grupo principal y directorio *home:*

```
$ ./mi_id 0
Nombre de usuario: root
ID de usuario,grupo: 0,0
Directorio home: /root

$ ./mi_id 89
Usuario no existe
```

Ejercicio 3. Modificar el programa anterior para que si no se usa ningún argumento se obtenga el ID del proceso con `getuid(2)` y muestre la información asociada a ese usuario.

```
$ ./mi_id
Nombre de usuario: ruben
ID de usuario,grupo: 1000,1000
Directorio home: /home/ruben
```

Ejercicio 4. Consultar la página de manual de `uname(1)` y obtener información del sistema.

Ejercicio 5. Escribir un programa (`mi_uname`) que muestre usando la llamada al sistema `uname(2)` la información del sistema.

```
$ ./miuname
Nombre SO: Linux
Version SO: 6.12.44_1 - #1 SMP PREEMPT_DYNAMIC
Nombre Host: pc-ruben
Arquitectura: x86_64
```