

Práctica 1: Introducción a la Administración de Sistemas

Objetivos

En esta práctica 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 utilizadas en la administración de sistemas y una breve introducción a la programación en *shell script*.

Contenidos

La Shell bash

- La línea de comandos
- Páginas de manual
- Comandos y secuencias de comandos
- Variables de entorno
- Las comillas
- Historia y atajos de teclado
- El path
- Para saber más...

Manejo de cadenas y flujos de caracteres

- Los comandos cat, wc, head, tail y tr.
- El editor de flujos Sed
- Para saber más...

El sistema de ficheros

- Listar contenidos
- Crear y borrar directorios
- Copiar, borrar y mover ficheros
- Caracteres comodín
- Buscar ficheros
- Para saber más...

Redirecciones, Tuberías y Expresiones regulares

Expresiones Regulares

- Para saber más...

Editar texto

- Para saber más...

Programación en Lenguaje Shell

Proyecto: Agenda en Shell script

La *shell* bash

La forma principal para interactuar con el sistema es la *shell*, y es la herramienta más importante del administrador de sistemas. En esta sección revisaremos algunos de los aspectos más importantes de este programa. Hay muchas shell disponibles, pero en este curso nos centraremos en bash.

La línea de comandos

El aspecto principal de la *shell* es la ejecución de comandos, que pueden estar implementados dentro del propio programa de la shell (*built-in*) o ser programas externos. La interacción con la shell se realiza por la línea de comandos encabezada por el *prompt*:

```
[cursoasr@CentOS ~]$
```

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

Toda la información sobre las aplicaciones del sistema se puede obtener con `man`.

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

Ejercicio 2. Se puede buscar en una sección determinada, 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` ó `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

A modo de ejemplo usaremos el comando `echo` en esta sección.

Ejercicio 1. Estudiar el funcionamiento del comando `echo`. ¿Qué hacen las opciones `-n` y `-e`? Imprimir la frase "Curso de Administración" 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 para qué sirven estos metacaracteres, ejemplo:

Listado 1. Ejecución de comandos en la shell

```
[asr@CentOS ~]$ echo línea uno;echo línea dos; echo línea tres
[asr@CentOS ~]$ echo línea uno && echo línea dos && echo línea tres
[asr@CentOS ~]$ echo línea uno || echo línea dos; echo línea tres
[asr@CentOS ~]$ (echo En una sub-shell; exit 0) && echo acabó bien || echo acabó mal
[asr@CentOS ~]$ (echo En una sub-shell; exit 1) && echo acabó bien || echo acabó mal
```

Variables de entorno

La ejecución de una shell tiene asociado un entorno (p. ej. el tipo de *prompt* usado) que incluye muchas variables. Estas variables pueden fijarse por el usuario y exportarlas al entorno de ejecución de otros procesos.

Ejercicio 1. Consultar el entorno mediante `env`, e identificar algunas variables importantes.

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:

Listado 2. Variables de entorno.

```
[cursoasr@CentOS ~]$ (exit 0);echo $?;(exit 4);echo $?  
[cursoasr@CentOS ~]$ echo $$ $PPID  
[cursoasr@CentOS ~]$ ps -p $$ -o "pid ppid cmd"
```

Ejercicio 3. Fijar el valor de las variables `VARIABLE1`, `VARIABLE2` y `VARIABLE3` a “Curso”, “de” y “administración”, respectivamente. Imprimir la frase “Curso de administración” accediendo a las variables. Ejecutar otra shell (comando `bash`) y volver a ejecutar la orden que imprime la frase, hacer que la frase se imprima correctamente en la nueva shell (orden `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 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 %).

Las comillas

En la shell hay tres tipos de entre-comillados

- Comillas dobles `“`: se resuelve el valor de las variables. “El usuario es `$USER`”
- Comillas simples `'`: no sustituye el valor de las variables. ‘El usuario es `$USER`’
- Comillas de ejecución ```: sustituye por el valor de la ejecución del comando. “El usuario es ``whoami``”

Ejercicio 1. Imprimir las frases anteriores usando `echo`. Probar la forma `$(comando)` como alternativa a ``comando`` en el último ejemplo.

Historia y atajos de teclado

La *shell* mantiene un listado (historia) de los comandos ejecutados. Esta lista se puede acceder con el comando `history`.

Ejercicio 1. Consultar la funcionalidad del comando `history`: listar la lista de comandos ejecutados, volver a ejecutar el último comando, ejecutar el 3er comando (por el principio y final de la lista)

Ejercicio 2. La historia se puede consultar interactivamente usando las teclas de movimiento de cursor arriba y abajo. Además se puede buscar en la historia con la combinación `Ctrl+r`. Buscar en la historia las distintas órdenes ejecutadas en la sesión.

Ejercicio 3. La línea de comandos (implementada por la librería `gnu readline`) admite dos modos

de edición emacs y vi que se pueden seleccionar con `set -o emacs` o `set -o vi`. En el modo por defecto (emacs) algunos comandos útiles son: Ctrl+a, Ctrl+e, Ctrl+k... Finalmente el tabulador sirve para auto-completar comandos y argumentos. Probar los atajos anteriores.

El path

Ejercicio 1. Los comandos son programas (ficheros con permisos de ejecución, +x) en el sistema. Los directorios en los que la shell busca los comandos está en la variable PATH. Consultar su valor.

Ejercicio 2. Añadir el directorio actual al PATH. ¿Qué diferencia hay entre añadir `./` y `$PWD`?

Ejercicio 3. La orden `which` determina qué comando se ejecutará cuando se usa en la línea de comandos. Determinar qué fichero se usa para ejecutar `bash`, `env`, `gzip`, `echo`, `set` y `ls`. Además se puede ver que tipo de archivo es con la orden `file` y `type`.

Ejercicio 4. Siempre se puede ejecutar un comando que no esté en el path especificando su ruta completa, ya sea de forma relativa o absoluta. Quitar el valor de la variable PATH de la shell usando `unset`. Se puede mostrar la frase “Curso de Administración” con `echo`, ¿por qué? Usar el path absoluto del comando `ls` determinado en el ejercicio anterior para listar los archivos.

Para saber más...

- Consultar la página de manual de bash
- Estudiar el uso del archivo de configuración `.bashrc`
- Familiarizarse con los atajos comunes de la shell

Manejo de cadenas y flujos de caracteres

Muchas de las tareas de la administración de sistemas requieren recoger una cadena (o flujo de caracteres), procesarlo y enviarlo al flujo de salida. Habitualmente este procesamiento se realiza construyendo tuberías para encadenar la entrada y salida de los comandos. En las siguientes secciones veremos en detalle este proceso y de momento nos limitaremos al uso básico de `|` y `>`.

Ejercicio 1. Estudiar el comando `sort` (`man sort`). Ordenar las palabras (cada una en una línea, \n) zorro pájaro vaca caballo y abeja usando el comando `echo` encadenando su salida (tubería `|`) al 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 y no la entrada estándar del comando `sort`.

Los comandos cat, wc, head, tail y tr.

Ejercicio 1. `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, cada una en una línea, pera, manzana, plátano y ciruela en el fichero `texto2`.
Nota: el flujo de entrada estándar se cierra con Ctrl+d. Comprobar el contenido de `texto2` con `cat`.

Ejercicio 2. `cat` se puede usar para concatenar ficheros si se especifican como argumentos. Unir los ficheros `texto1` y `texto2` en `texto3` usando la redirección (`>`) en el fichero `texto4`.

Ejercicio 3. `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 4. 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 5. El comando tr sirve para cambiar caracteres (translate) o eliminarlos. Poner todas las palabras del fichero texto1 en una línea sustituyendo el carácter fin de línea por un tabulador.

El editor de flujos sed

Sed es un editor de flujos extremadamente potente. Veremos en esta sección una pequeña introducción a esta utilidad. Como las utilidades anteriores sed puede trabajar con ficheros o con la entrada estándar.

Ejercicio 1. Sed usa expresiones regulares (en detalle en la siguientes secciones) para determinar las partes del texto que se modificarán. Usar el comando de sustitución s para cambiar 'a' por 'A' (s/a/A/). Se puede usar el modificador g (global al final s/a/A/g), observar la diferencias.

Ejercicio 2. Eliminar la segunda línea del fichero (las líneas se eliminan con el comando <num_línea>d) (2d).

Ejercicio 3. Se puede especificar el rango en el que se aplica el comando sed, que por defecto es todas las líneas del fichero. Los rangos se pueden determinar como:

- Un número de línea. Ej: '3 s/[0-9]//g' para eliminar los números de la línea 3
 - Un rango de líneas. '2,4 d' ó '4,\$' para borrar las líneas 2 a 4, y desde la 4 hasta el final
- Probar a cambiar 'a' por 'A' de las la línea 3 del fichero texto1. Simular el comando head, obtener la primera línea del fichero texto1.

Ejercicio 4. Muchas veces es necesario referirse al patrón encontrado para añadir algún carácter o modificación. Se puede incluir el patrón encajado con el carácter '&'. Poner todas las palabras de texto1 dos veces en cada línea.

Para saber más...

- <http://sed.sourceforge.net/sed1line.txt>
- Más comandos de utilidad od, uniq, join

El sistema de ficheros

En esta sección veremos los comandos básicos para manejar el sistema de ficheros. Partimos de los archivos texto1...texto3 creados en las secciones anteriores.

- Un archivo puede referirse de forma absoluta con su path completo (empieza por /)
- Puede referirse de forma relativa al directorio actual
- . hace referencia al directorio actual
- .. hace referencia al directorio directamente superior

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

- Cambiar al directorio de usuario (cd sin argumentos).
- Averiguar el 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

Listar contenidos

Ejercicio 1. La orden para listar el contenido de un directorio es ls. Consultar la página de manual y estudiar el uso de las opciones -a -l -d -h -i -R -1 -F y --color. Probar estas opciones con el directorio de trabajo. Estudiar el significado de cada campo del listado extendido (-l).

Ejercicio 2. Listar los contenidos de los directorios /usr y /etc.

Ejercicio 3. Listar los inodos de los directorios anteriores, uno por línea. Eliminar de la salida el encabezado del directorio (/usr: y /etc:) con el comando sed. El resultado ordenarlo por inodo (ordenación numérica) con el comando sort y guardarlo en el fichero texto4.

Crear y borrar directorios

Ejercicio 1. La orden mkdir sirve para crear directorios. Hacer el directorio 'mis_archivos' en el directorio de trabajo. Comprobar su creación con el comando ls.

Ejercicio 2. 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 3. Un directorio se puede borrar con rmdir, pero debe estar vacío. Eliminar el directorio prueba (y todos sus contenidos con rmdir)

Copiar, borrar y mover ficheros

Ejercicio 1. Copiar el fichero texto1 a copia1 con cp

Ejercicio 2. Crear un directorio llamado 'copia'. Copiar en él los archivos texto1, texto2 y copia1 al directorio copia con la orden cp y en una única línea.

Ejercicio 3. Los directorios se pueden copiar usando la opción -r. Copiar el directorio copia al directorio otra_copia. Consultar la página de manual de cp y estudiar las opciones -f y -i.

Ejercicio 4. Los ficheros y directorios se pueden renombrar con el comando mv:

- Entrar en el directorio copia. Renombrar texto1 a fichero1.
- Mover fichero1 al nivel superior (\$HOME), y comprobarlo con ls (sin usar cd).
- Dentro de copia, hacer el directorio test y mover el resto de los archivos a test.
- Mover test al directorio \$HOME usando la ruta absoluta.
- Subir al nivel superior y borrar el directorio copia.
- Mover fichero1 a test con nombre texto1 (en una orden mv).
- Renombrar test a copia, la situación debe ser igual a la inicial.

Nota: Recordar el uso del tabulador en la shell

Ejercicio 5. Para borrar usar el comando rm. Borrar los ficheros del directorio copia con rm, y

finalmente el propio directorio copia con `rmdir`.

Ejercicio 6. `cp` y `rm` aceptan opciones recursivas (`-r`) para copiar y borrar directorios respectivamente. Cambiar al directorio `$HOME`, crear un directorio prueba y copiar todos los archivos texto en él. Copiar este directorio prueba a otro de nombre `otra_prueba` con la opción `-r`. Borrar ambos directorios con `rm`. Comprobad los resultados parciales con `ls`.

Caracteres comodín

La shell expande una serie de patrones (*wildcards*) que permiten operar sobre muchos archivos simultáneamente, en particular:

- `?` encaja con un único carácter
- `*` encaja con cualquier cadena incluso la vacía
- `[]` define un conjunto de caracteres que encajan con cualquier carácter en el conjunto. El `-` define rangos y la `!` sirve para excluir un carácter, ej. las letras `[a-z]` ó cualquier carácter menos los números `[!0-9]`

Ejercicio 1. De los ficheros `texto1`, `texto2...` `texto4`:

- Listar todos los ficheros que empiecen por `texto`
- Listar todos los ficheros que acaben por un número
- Listar los ficheros que no terminen en `3` ó `4`
- Hacer un directorio `texto5` copiar alguno ficheros en él y repetir el comando anterior. Añadir las opciones necesarias a `ls` para que no se muestren los contenidos de `texto5`.

Buscar ficheros

El comando `find` sirve para buscar ficheros en el árbol de directorios según uno o más criterios y permite opcionalmente ejecutar un comando sobre los ficheros encontrados.

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 el directorio `/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.

Ejercicio 4. Se pueden seleccionar archivos por tiempos de modificación (`-mtime`), acceso (`-atime`) y cambios de estado (`-ctime`). Buscar los archivos modificados en las últimas `24h` (`-mtime 0`) en el directorio `$HOME`.

Ejercicio 5. Se pueden ejecutar acciones sobre cada resultado de la búsqueda. Hay predefinidas (`-ls`, `-delete` o `-print`) o genéricas `-exec`. Cuando se usa `-exec` la combinación `'{}' \;` representa el resultado y se puede usar como argumento. Mostrar el número de inodo de cada fichero (no directorio o de cualquier otro tipo) en el ejemplo anterior.

Para saber más...

- Estudiar los comandos de compresión: `gzip`, `zip`, `unzip`
- Estudiar el comando para archivar ficheros: `tar`
- Estudiar el comando `dd`

Redirecciones, Tuberías y Expresiones regulares

Los procesos tienen tres flujos por defecto, la entrada estándar (descriptor 0), la salida estándar (descriptor 1) y la salida estándar de error (descriptor 2). Cada uno de estos flujos se pueden tratar independientemente usando su número de descriptor.

Ejercicio 1. Ejecutar en el directorio \$HOME el comando `ls -l text* nada* > salida`. ¿Qué sucede?, ¿cuáles son los contenidos del fichero salida?. **Nota:** Además de `cat`, se pueden mostrar los contenidos de un fichero con el comando `less` (man `less`, especialmente la búsqueda con `/`)

Ejercicio2. Con el comando anterior redirigir la salida estándar a `salida.out` y la salida estándar de error a `salida.error`. Comprobar el contenido.

Ejercicio3. 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, comprobar el resultado.

Ejercicio 4. Se pueden duplicar los descriptors con `&` (`>&` y `>>&`), en la forma comando `> salida 2>&1`. Comprobar que el comportamiento es distinto a comando `2>&1 > salida` (las “duplicaciones” se hacen secuencialmente). Redirigir la salida estándar y de error del comando `ls` anterior al fichero `salida.txt`

Ejercicio 5. 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 6. También es posible redirigir la entrada estándar. Comparar `cat texto1 | sort` con `sort < texto1`

Ejercicio 7. Una forma muy útil de usar la redirección de entrada (especialmente en scripts) es la que se muestra a continuación. Probar su funcionamiento

Listado 3. Delimitación de entrada estándar

```
[cursoasr@CentOS ~]$ cat << FIN_ARCHIVO > salida
> uno
> dos
> tres
> FIN_ARCHIVO
```

Expresiones Regulares

Las expresiones regulares son otra de las herramientas básicas para administrar sistemas. Permiten buscar de forma rápida en el contenido de archivos. Los operadores principales del comando `grep` son:

- Caracteres y grupos de caracteres. Ejemplos: `a`, `b`, `[aA]`, `[0-9]`, `[A-Za-z]`, `[:blank:]`, `[:alnum:]`
- Posicionamiento (anclas): `^` ppio de línea, `$` final de línea, `\<` ppio de palabra, `\>` final de palabra
- *wildcards*: `.` cualquier carácter, `*` el patrón anterior 0 o más veces, `+` el patrón anterior 1 o más veces
- Repeticiones: `{N}` `{N,}` `{N,M}` el patrón se repite N veces, N veces o más, N veces y no más de

M.

Los caracteres anteriores se pueden escapar con \, para encajar el patrón con el propio carácter. En los siguientes ejercicios usaremos el archivo texto1

Listado 4. Contenidos del fichero de práctica de patrones.

```
[cursoasr@CentOS ~]$ > cat texto1
zorro
pájaro
vaca
caballo
abeja
```

Ejercicio 1. Buscar en el fichero:

- 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)

Para saber más...

- Consultar la página de manual de grep
- Estudiar el uso de () para agrupar expresiones | para hacer o-lógicas. Dependiendo de la versiones de grep puede ser necesario escapar los operadores anteriores.

Editar texto

Además manipular ficheros con los comandos anteriores, la edición de textos es una de las actividades principales del administrador de sistemas. El comando que se usa es vi o en su versión mejorada vim.

Ejercicio 1. Familiarizarse con el manejo de vim, con el programa vimtutor.

Para saber más...

- Consultar tutoriales (muchos de ellos interactivos) en internet sobre vim.
- Es posible usar el editor nano, de paradigma similar a los editores convencionales. Sin embargo es aconsejable el manejo de vi ya que muchas de sus comandos son equivalentes en otros programas (e.g. patrones sed y grep, búsqueda en man y less...).

Programación en lenguaje shell

Un script (guión) shell es una secuencia de órdenes que se ejecuta secuencialmente como si fuera un programa. La secuencia de órdenes se escriben en un archivo que se puede invocar (si tiene los permisos adecuados) como cualquier otro comando. Las características de este archivo son:

- Deben comenzar por #! y el path del intérprete que se usará para procesar el script, p. ej. #!/bin/bash
- Debe observar la sintaxis del intérprete shell, en este caso bash (p. ej. uso de comillas)
- Los comentarios comienzan por #

Ejercicio 1. Argumentos de un programa. Los argumentos del script se acceden por su posición, p. ej. \$1 para el valor del primer argumento, \$2 el segundo... \$0 hace referencia al nombre del propio

fichero, \$* a todos los argumentos y \$# al número de argumentos.

Escribir un script que muestre el nombre del fichero del script, el primer y segundo argumento, cada uno en una línea.

Nota: El entrecomillado doble sirve para agrupar argumentos de un programa. ¿hay alguna diferencia entre ejecutar el script con argumentos uno dos y “uno dos”?

Ejercicio 2. Sentencias Condicionales. Las condiciones se evalúan mediante la orden test, consultar en la página de manual todas las posibles comparaciones. La orden test también se puede escribir como [, así test -f /bin/bash es equivalente a [-f /bin/bash]:

- Comprobar en un terminal el resultado de la ejecución de las sentencias anteriores mostrando \$? en cada caso.
- Las condiciones se usan junto con la construcción if-then-else-fi de bash (ver página de manual). 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 es un fichero, 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.

Listado 5. Ejemplo de ejecución

```
$/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>
```

Nota: Bash también implementa la construcción case, que se basa en el uso de patrones para ejecutar cada rama y select que permite crear menús sencillamente.

Ejercicio 3. Bucles. Bash implementa las opciones habituales para escribir bucles: for, while y until. La sentencia for itera sobre una lista de elementos que habitualmente es el resultado de la ejecución de otro comando, es decir se realiza la expansión. Escribir un *script shell* que muestre el número de palabras de cada fichero (sólo si es un fichero) en un directorio dado (argumento del script).

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

Ejercicio 3. 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?

Listado 6. Ejemplo de función

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

hola mundo
```

```
A=`hola mundo`
```

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:

Listado 7. Formato de la base de datos

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

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

Listado 8. Ejemplo de 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: ruben
Teléfono: 1234
Mail: ruben@ucm.es

Nombre: juan
Teléfono: 2345
Mail: juan@ucm.es

Nombre: maría
Teléfono: 2345
Mail: maria@ucm.es
#? 3
Nombre: ruben
```

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

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

#? 5
```

Notas sobre la implementación:

- Usar una función que imprima el menú y ejecute cada acción. La función debe usar las construcciones `select` y `case`.
- Las acciones de la agenda se pueden implementar usando las órdenes básicas explicadas en la práctica.

Opcional: Completar la funcionalidad de la agenda con los comandos aprendidos, p. ej. mostrar el número de registros, detectar nombres duplicados antes de añadir, comprobar la corrección de la entrada de usuario...