Práctica Opcional: Introducción al Lenguaje de Programación Shell.

Objetivos

En esta práctica el alumno se familiarizará con la programación en lenguaje de shell, concretamente su variante Bash. Para ello se revisará el manejo básico de la shell y los comandos y utilidades principales.

Contenidos

Preparación del entorno para la práctica

La realización de esta práctica únicamente requiere del entorno de desarrollo (compilador, editores y utilidades de depuración). Estas herramientas están disponibles en las máquinas virtuales de la asignatura y en la máquina física de los puestos del laboratorio.

En la realización de las prácticas se puede usar cualquier editor gráfico o de terminal. Además se puede usar tanto el lenguaje C (compilador gcc) como C++ (compilador g++). Si fuera necesario compilar varios archivos se recomienda el uso de alguna herramienta para la compilación de proyectos como make. Finalmente, el depurador recomendado en las prácticas es gdb. **No está permitido** el uso de IDE como Eclipse.

La *Shell* bash

La forma principal para interactuar con el sistema es la shell, y una herramienta fundamental para el desarrollo de aplicaciones de sistema. En esta sección revisaremos algunos de los aspectos más importantes de este programa. Estas características se utilizarán luego para la confección de programas (scripts) shell. Hay muchas shell disponibles en un sistema UNIX, pero en este curso nos centraremos en bash.

Páginas de Manual

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

Ejercicio 1. Consultar la página de manual de man (`man man`) . Especialmente las secciones y las partes 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).

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 "Ampliación de Sistemas Operativos y Redes" con un tabulador al principio.

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

Ejercicio 2. Estudiar para qué sirven los metacaracteres anteriores:

Listado 1. Diferentes listas de órdenes

```
$ 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 terminó bien || echo terminó mal
$ (echo En una sub-shell; exit 1) && echo terminó bien || echo terminó 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 *export*arlas 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 y ?. Ejemplo:

Listado 2. Ejemplos de Variables

```
$ (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 "Ampliación de SO", 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". Usando el comando echo y el contenido de la variable VAR1 escribir las cadena "1+1=2".

Las comillas

En la shell hay tres tipos de entrecomillados

- Comillas dobles “: se realizan sustituciones de 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.

El path

Ejercicio 1. Los comandos son programas (ficheros) en el sistema . Los directorios en los que la shell busca los comandos esta 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 qué tipo de archivo es con la orden `file`.

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 Ampliación de SO” con echo, ¿por qué?
- Usar el path absoluto del comando ls determinado en el ejercicio anterior para listar los archivos.

Manejo de cadenas y flujos de caracteres

Habitualmente un programa en shell script requiere recoger una cadena (o flujo de caracteres), procesarlo y enviarlo al flujo de salida. 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 abeja y paloma, usando el comando echo y 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 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 3. Ejecutar en el directorio $HOME el comando ls -l text* nada* > salida. ¿Qué sucede?, ¿cuáles son los contenidos del fichero salida (cat salida)?.

Ejercicio 2. Con el comando anterior redirigir la salida estándar a salida.out y la salida estándar de

error a salida.error. Comprobar el contenido.

Ejercicio 4. 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 y comprobar el resultado.

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

Ejercicio 1. cat muestra el contenido de un fichero. 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 ('>').

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 cómo 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 dmesg.
- Usar tail para quedarse con la última parte (ej. últimas 15 líneas)
- Usar head para quedarse con las primeras líneas (ej. primeras 3 líneas)

Nota: 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 (*tr*anslate) o eliminarlos. Poner todas las palabras del fichero texto1 en una línea sustituyendo el carácter fin de línea por un tabulador.

Expresiones Regulares

Las expresiones regulares son otra de las herramientas básicas en la programación en shell script. Permiten buscar de forma rápida en el contenido de archivos.
Los operadores principales del comando grep para la construcción de expresiones regulares 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

Ejercicio 1. Buscar las palabras con 'ja'

Ejercicio 2. Buscar las palabras que terminan en 'ja'

Ejercicio 3. Buscar las palabras que tiene dos aes con una letra en medio (aba,aca,aaa,ala,...).

Ejercicio 4 . Buscar el patrón alo ó allo (la l se repite una o dos veces)

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 3. Ejemplo de ejecución

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

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

Ejercicio 2. **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. Escribir una función que

Listado 4. 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 5. 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 6 . 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, a excepción de borrar
- La orden borrar se implementará con la orden sed que usa patrones similares a grep, consultar su funcionamiento en la página de manual.

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