Bloque II: Programación orientada al sistema

Tema 9. Programación en Lenguaje C

Tema 10. Programación del intérpretes de órdenes

- Introducción
- Repaso de las principales órdenes del intérprete
- Redirección de entrada y salida
- Creación y ejecución de *scripts*
- Paso de parámetros a un *script*
- Instrucciones condicionales
- Listas de instrucciones
- Instrucciones de lazo
- Instrucción de lectura

1. Introducción

Una *shell* es un programa que lee instrucciones proporcionadas por un usuario, las interpreta, y encarga al sistema operativo su ejecución

Las instrucciones se pueden dar:

- **por teclado (intérprete de órdenes tradicional)**
- **mediante ratón (gestor gráfico de ficheros)**
- **desde un fichero, llamado un *script*, con un texto escrito en un lenguaje que la *shell* entiende: el lenguaje de la *shell***

El lenguaje de la *shell*

El lenguaje de la *shell* incluye:

- todas las órdenes que se pueden dar por teclado
  - p.e., `cp` (copia), `mv` (mueve), `rm` (borra)
- instrucciones de programación:
  - variables
  - parámetros
  - instrucciones condicionales
  - lazos
  - instrucciones de lectura
  - instrucciones remotas, para ejecutar en otras máquinas
  - etc.

2. Repaso de las principales órdenes del intérprete

Las órdenes simples tienen habitualmente este formato

orden opciones argumentos

Las opciones suelen ir precedidas de un signo '`-`'

Una orden se puede ejecutar en segundo plano (concurrentemente a la shell) añadiendo un '`&`' al final:

```
gcc programa.c &
```

Los metacaracteres o caracteres especiales `<>*?|&,;()[]#` tienen un significado especial para la *shell*

- ponerlos precedidos de '`\`' para perder su significado especial; p.e., `\*` o `\\`
- o ponerlos entre apóstrofes: `'<*****>'`

Nombres de ficheros

Cuando un argumento es un nombre de fichero se pueden usar caracteres comodín:

- **'*': se reemplaza por cualquier secuencia de cero o más caracteres que no empiece por '.'**
- **'?': se reemplaza por un carácter simple cualquiera**
- **[...]: se reemplaza por cualquiera de los caracteres entre los corchetes; por ejemplo:**
  - **[abc]*: todos los ficheros que comienzan por 'a', 'b', o 'c'**
  - **[a-zA-Z]*.txt: todos los ficheros que comienzan por una letra minúscula o mayúscula, y acaban en ".txt"**

Órdenes del sistema operativo

UNIX	Función	Sintaxis
ls ls -l	Muestra una lista del contenido del directorio	ls -l nombreDirectorio
cd	Cambiar el directorio de trabajo	cd nombreDirectorio
rm	Borrar un fichero	rm nombreFichero
	Borrar varios ficheros	rm nombreComodín
cp	Copiar un fichero en otro;	cp ficheroOrigen ficheroDestino
	Copiar uno o varios ficheros en otro directorio	cp nombreComodín dirDestino

Órdenes del sistema operativo (cont.)

UNIX	Función	Sintaxis
mv	Mover un fichero a otro (es decir cambiarle el nombre);	mv ficheroOrigen ficheroDestino
	Mover uno o varios ficheros a otro directorio	mv nombreComodín dirDestino
more less	Mostrar un fichero en pantalla	more nombreFichero less nombreFichero
mkdir	Crear un nuevo directorio	mkdir nombreDirectorio
rmdir	Borrar un directorio vacío	rmdir nombreDirectorio
rm -r	Borrar un directorio y todos sus contenidos	rm -r nombreDirectorio
lpr	Imprimir un fichero	lpr nombreFichero
man	Pedir info sobre una orden	man orden

Órdenes del sistema operativo (cont.)

UNIX	Función	Sintaxis
pwd	Muestra el directorio de trabajo	pwd
who	Muestra los usuarios conectados	who
grep	busca un texto en un fichero	grep "texto" nombreFichero
echo	muestra los argumentos especificados	echo arg1 arg2
wc	cuenta el número de caracteres, palabras y líneas de su entrada estándar	ls | wc cat nombre | wc
cat	muestra los ficheros especificados, uno a continuación del otro	cat nombre1 nombre2
find	Busca en un directorio ficheros con determinadas características: -name: por nombre	find nomDir -name nombre

Órdenes del sistema operativo (cont.)

UNIX	Función	Sintaxis
chmod	Añade(+) o quita(-) para el usuario(u), grupo (g) u otros (o) los permisos de lectura(r), escritura (w) o ejecución (x)	chmod u+rw fichero chmod ugo-x fichero chmod ug+rwx fichero
diff	Muestra diferencias entre dos ficheros	diff fichero1 fichero2

3. Redirección de entrada y salida

En Unix los procesos tienen tres ficheros abiertos al arrancar:

- **entrada estándar, salida estándar, y salida de error**
- **suelen ser el teclado, pantalla, y pantalla, respectivamente**

Se puede modificar el fichero al que están ligados:

- **entrada estándar: `< nombre`**
  - **número de líneas del fichero nt.txt: `wc -l < nt.txt`**
- **salida estándar: `> nombre`**
  - **listado del directorio al fichero lista: `ls -l > lista`**
  - **para añadir a un fichero, usar `>>`: `ls -l >> lista`**
- **salida de error: `2> nombre`**

Tuberías y filtros

Podemos conectar la salida estándar de una orden con la entrada estándar de otra, mediante una *tubería*:

```
ls -l | wc
```

Un *filtro* es una orden que lee la entrada estándar, la transforma, y escribe en la salida estándar

- `grep palabra`: busca líneas que contengan la palabra
- `sort`: ordena alfabéticamente

Podemos conectar varias órdenes y/o filtros con tuberías:

```
ls -l | grep txt | wc -l
```

- muestra el número de líneas que contienen la palabra "txt" del listado de ficheros del directorio actual

4. Creación y ejecución de *scripts*

Un *script* es un programa escrito en el lenguaje de la shell en un fichero de texto

Para poder ejecutar un script el fichero debe tener permiso de ejecución

Ejemplo:

- crear el fichero lista con el editor de texto (*emacs*) y el siguiente contenido

```
# Este script pone en pantalla el directorio de trabajo
# y los ficheros contenidos en él
echo "Directorio de trabajo:" $PWD
echo "Ficheros que contiene:"
ls
```

Ejemplo (cont)

- A continuación, darle permiso de ejecución:

  chmod +x lista
- Finalmente, ejecutarlo:

  ./lista

Observaciones:

- El carácter '#' inicia un comentario
- El signo '$' identifica una variable; $PWD es una variable predefinida, que indica el directorio de trabajo
- El script contiene una lista de instrucciones simples (echo, ls)

5. Paso de parámetros a un *script*

Al invocar un *script* se le pueden pasar *parámetros*

- cada palabra o string entre comillas es un parámetro

Dentro del *script* cada parámetro se identifica con $n, donde n=1, 2, 3,... representa el primer, segundo, tercer,... parámetro

```
#añade al fichero $1 la terminación ".txt"
#además, copia las líneas con "texto" al fichero $2
mv $1 $1.txt
grep "texto" $1.txt > $2
```

El parámetro especial $* representa a todos los parámetros, desde el $1

```
ls -al $*
```

Variables

Se pueden crear *variables*

XREF="texto simple"

Tener cuidado de no poner espacios a los lados del '='

Para usar la variable:

$XREF

Ejemplo:

```
#cambia los permisos del fichero $1 recursivamente
ORDEN="chmod"
ARG="-R +rwx"
$ORDEN $ARG $1
```


Variables (cont.)

Variables *predefinidas*

$?	Valor de salida del último proceso ejecutado
$#	Número de argumentos
$$	Pid de la shell
$HOME	Directorio inicial del usuario
$PATH	Ruta de búsqueda
$PWD	Directorio de trabajo

Para que una variable sea visible desde otras órdenes debe exportarse:

export PATH

Sustitución de órdenes

Podemos *sustituir* una orden por su resultado, encerrándola entre apóstrofes inversos:

`FICH=‘ls‘`
- **`FICH`** toma como valor la lista de ficheros del directorio actual

`DIR=‘pwd‘`
- **`DIR`** toma como valor el directorio de trabajo

6. Instrucciones condicionales

Instrucción condicional *simple*

```
if condicion
then
   ordenes
fi
```

Instrucción condicional simple con *alternativa* (*else*)

```
if condicion
then
   ordenes
else
   ordenes
fi
```

Instrucciones condicionales

Instrucción condicional *múltiple*

```
if condicion
then
   ordenes
elif condicion
then
   ordenes
elif condicion
then
   ordenes
else
   ordenes
fi
```


Condiciones

La instrucción `test` retorna verdad o falso y se usa en las condiciones

test operando operador operando

test operador operando

O abreviadamente

[operando operador operando]

[operador operando]

Notas:

- los espacios entre operandos y operadores son imprescindibles
- es conveniente poner los operandos entre comillas

Operadores

Operador	Produce true si...	Nº de operandos
`-n`	el operando tiene longitud distinta de cero	1
`-z`	el operando tiene longitud cero	1
`-d`	existe un directorio cuyo nombre es el operando	1
`-f`	existe un fichero cuyo nombre es el operando	1
`-eq`	los operandos son números enteros iguales	2
`-ne`	lo contrario de -eq	2
`=`	los operandos son strings iguales	2
`!=`	lo contrario de =	2
`-lt -gt -ge -le`	respectivamente <, <=, >=, > para operandos enteros	2

Ejemplo

```
if [ $# != 1 ]     # num parametros no es uno
then
   echo "formato: perm directorio"
elif [ -d $1 ]     # directorio $1 existe
then
   #cambia los permisos del directorio $1 recursivamte
   echo "cambiando permisos de $1"
   chmod -R +rwx $1
else
   echo $1 "no es un directorio"
fi
```

7. Listas de instrucciones

Una *lista* de instrucciones es una secuencia de una o varias instrucciones o instrucciones encadenadas con tuberías, separadas por uno de los operadores:

- **&&**: operador "y"

  orden1 && orden2
  - la orden2 se hace si y sólo si la orden1 termina bien (con valor retornado 0)
- **||**: operador "o"

  orden1 || orden2
  - la orden2 se hace si y sólo si la orden1 termina mal (con valor retornado distinto de 0)

Se pueden agrupar órdenes con paréntesis **( )**

Ejemplos de listas

Compila y ejecuta:

```
gcc $1.c && ( echo "ahora ejecuta" && ./a.out )
```

Ejecuta y muestra mensaje de error:

```
./$* || echo "la orden falla"
```

8. Instrucciones de lazo

Instrucción *for*:

```
for var in valor1 valor2 valor3 ...
do
   ordenes
done
```

Se crea la variable `$var`, que va tomando los sucesivos valores valor1, valor2, valor3, ...

Los valores se pueden:

- **escribir explícitamente**
- **o tomar de una orden, por sustitución: `‘orden‘`**

Ejemplo de instrucción for

```
#muestra el nombre y número de líneas de cada fichero
#del directorio actual
for i in `ls`
do
   echo -n "líneas del fichero $i"
   wc -l < $i
done

#busca ficheros de texto con el texto "tex"
for i in *.txt
do
   grep -l "tex" "$i"
done
```

Lazo con condición de permanencia

Instrucción *while*:

```
while orden
do
   ordenes
done
```

La condición de permanencia es que la orden de la 1ª línea sea correcta (su valor de retorno sea cero)

La palabra `until` puede sustituir a `while` para hacer la condicion contraria

```
until [ -f file ]  # espera hasta que file exista
do sleep 1;
done
```


9. Instrucción de lectura

La instrucción `read` permite leer una línea del teclado y guardar las palabras tecleadas en variables:

read var1 var2 var3 ...

Ejemplo

```
#compila y pregunta si quieres ejecutar
gcc $1.c
echo -n "quieres ejecutar? "
read quieres
if [ $quieres = "si" ]
then
   a.out
fi
```