Programación del shell

2. Control del flujo de programa

1. El mandato *test* ó *[]*
2. Esquemas Condicionales
   1. if_then_fi
   2. if_then_else_fi
   3. if_then_else if ... fi
   4. case_esac
3. Esquemas repetitivos
   1. for_done
   2. while_do_done
   3. until_do_done
   4. break y continue

2. Control del flujo de un programa
2.1. El mandato test ó []

Para comprobar si una <expresión> es cierta (devuelve 0) o falsa (devuelve 1), se puede evaluar con el comando **test**.

Sintaxis permitidas para **test**

- **test** [<expresión>]
- **[** [<expresión>] **]**

Es importante notar que:

- Los corchetes externos, **[],** no implican opcionalidad, son la sintaxis alternativa de **test**
- Tanto los **[]** como los operadores tienen que tener un espacio en blanco a cada lado
- Las expresiones pueden alargarse más de una línea utilizando el carácter \
- Existen múltiples operadores y conectores para evaluar en **test / []**

2.1. El mandato test (cont)

Operadores para test / [ ]

- **Expresiones sobre ficheros**

  -d *fichero*: ¿es un directorio?

  -f *fichero*: ¿es un fichero normal?

  -x *fichero*: ¿es ejecutable?

  -w *fichero*: ¿es un fichero sobre el que se puede escribir?

  -r *fichero*: ¿es un fichero que se puede leer?

  -s *fichero*: ¿tiene longitud mayor que cero?

  -t *[descriptor_fichero]*: ¿el descriptor de fichero está asociado al terminal?

- **Expresiones sobre enteros**

  ent1 –eq *ent2*: ¿son iguales?

  ent1 –ne *ent2*: ¿son distintos?

  ent1 –ge *ent2*: *¿ent1* es mayor o igual que *ent2?*

  ent1 –gt *ent2*: *¿ent1* es mayor que *ent2?*

  ent1 –le *ent2*: *¿ent1* es menor o igual que *ent2?*

  ent1 –lt *ent2*: *¿ent1* es menor que *ent2?*

2.1. El mandato test (cont)

- **Expresiones sobre cadenas / *strings***

  cadena: cierto si *cadena* no es una cadena vacía

  cadena1 = *cadena2*: ¿son la misma cadena?

  cadena1 != *cadena2*: ¿son distintas?

  -n *cadena*: ¿tiene longitud mayor que cero?

  -z *cadena*: ¿tiene longitud cero (es la cadena vacía)?

- **Conectores**

  !: NOT

  -a: AND

  -o: OR

2.2. Esquemas condicionales del shell

Permiten seleccionar unas partes del código u otras en función de la evaluación de expresiones

2.2.1. Esquema condicional 1: *if_then_fi*

if <expresión>
then
<sentencias_then>
fi

- <expresión>: se corresponde con una lista de comandos que deben evaluarse a *true* para entrar en <sentencias_then>
- <expresión>: si se evalua a *false* se pasa a la siguiente instrucción tras **fi**

2.2.2. Esquema condicional 2: if_then_else_fi

if <expresión>
then
<sentencias_then>
else
<sentencias_else>
fi

Esquema condicional 3: if_then_else if_fi

if <expresión>
then
<sentencias_then>
elif <expresión2>
then
<sentencias_elif_1>
...
else
<sentencias_else>
fi

2.2.3. Esquema condicional generalizado

```
case <expresión_texto> in
    patrón_1) <sentencias_1>
            ;;
    patrón_2) <sentencias_2>
            ;;
    ...
    patrón_n) <sentencias_n>
            ;;
esac
```

- realiza las sentencias asociadas al *patrón_i* que coincida con *<expresión_texto>*
- **;;** rompe la evaluación de patrones
- se pueden combinar varios patrones con: *patrón_1* | *patrón_2* | patrón_3) <sentencias_comunes> ;;
- admite casos por defecto con ***)**

2.3. Esquemas repetitivos del shell

2.3.1. Esquema *for_done*

Sintaxis:

```
for <var> [in <lista_argumentos>]
do
  <ordenes>
done
```

Semántica:

- Ejecuta <ordenes> tantas veces como argumentos haya en <lista_argumentos>
- Si no existe <lista_argumentos>, se toman los elementos de la linea de mandatos

2.3.2. Esquema *while_do_done*

Sintaxis:

```
while <expresion>
do
  <ordenes>
done
```

Semántica:

- Ejecuta <ordenes> mientras <expresion> se evalue a cierto
- Hay que tener cuidado de no generar bucles infinitos
- No obstante, hay programas (como los servidores) que se ejecutarán mediante bucles infinitos

2.3.3. Esquema *until_done*

Sintaxis:

```
until <expresion>
do
  <ordenes>
done
```

Semántica:

- Ejecuta <ordenes> hasta que <expresion> se evalue a cierto, esto es, mientras <expresion> se evalue a falso