Programación con sh

Joaquín Seoane Pascual
Departamento de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
joaquin@dit.upm.es

21 de octubre de 2001

Índice General

Decisiones

case

```
case var in
    patrón1 ) órdenes1 ;;
    patrón2 ) órdenes2 ;;
    patrón3 ) órdenes3 ;;
esac
```

if

```
if órdenes
then
    órdenes
{
elif órdenes
then
    órdenes
}
[
else
    órdenes
]
fi
```

Repeticiones

Repetición fija

`for` *var* [`in` *p1 p2 . . . pn*]
`do`
    órdenes
`done`

ó

`for` *var* [`in` *p1 p2 . . . pn*]*;* `do` *órdenes;* `done`

Mientras no fallen

```
while órdenes
do
    órdenes
done
```

Hasta que no fallen

```
until órdenes
do
    órdenes
done
```

Funciones

nombre-de-función ( ) {
    órdenes
}

Patrones

`case` y nombres de fichero	
`*`	casa con cualquier cadena, incluso la vacía
`?`	casa con cualquier carácter
`[`*ccc*`]`	casa con cualquier carácter en *ccc*
	(rangos como en `[a-z0-9]`)
`'...'`	casa ...exactamente
`"..."`	casa ...casi exactamente (reemplaza variables)
`\c`	casa c literalmente

sólo case	
`a|b`	casa con a o b

en nombres de ficheros	
`/`	debe ser explícito
`.`	debe ser explícito si es el primer carácter

Variables especiales

$#	número de argumentos
$*	todos los argumentos
$@	idem....
$$i$$	argumento *i-simo*
$?	valor de retorno de última orden
$$	id-proceso de la shell
$!	id-proceso de la última orden &
$PATH	directorios de búsqueda de órdenes
$PS1	*saludo* principal ($)
$PS2	*saludo* secundario (>)

Evaluación de variables

`$var`	valor de `var`; nada si no definida.
`${var}`	idem; útil si siguen alfanuméricos.
`${var-cosa}`	valor de `var` si definida; si no, `cosa`.
`${var=cosa}`	valor de `var` si definida; si no, `cosa`. Si indefinida, `$var` a `cosa`.
`${var+cosa}`	`cosa` si var definida, en caso contrario nada.

Programa `test` o `[`

`-f` fichero	fichero normal
`-d` fichero	directorio
`-b` fichero	dispositivo de bloques
`-c` fichero	dispositivo de caracteres
`-h` fichero	un enlace simbólico
`-p` fichero	un *pipe* con nombre
`-w` fichero	escribible
`-x` fichero	ejeutable
`-r` fichero	legible
`-s` fichero	el tamaño no es cero
`-u` fichero	set-uid

Programa `test` o `[` para cadenas

`-l` cadena	longitud de cadena
`-n` cadena	longitud no es cero
`-t`	salida estándar es terminal
`-t` fd	fd es terminal
`-z` cadena	la longitud de cadena es cero
s1 = s2	cadenas iguales
s1 != s2	cadenas distintas
s1	s1 no es nula

Programa `test` o `[` para números y expresiones

n1 `-eq` n2	$n1 = n2$
n1 `-ne` n2	$n1 \neq n2$
n1 `-gt` n2	$n1 > n2$
n1 `-ge` n2	$n1 \geq n2$
n1 `-lt` n2	$n1 < n2$
n1 `-le` n2	$n1 \leq n2$
`!` expr	negación
e1 `-a` e2	y
e1 `-o` e2	o
`( ... )`	agrupamiento

Ejemplos sencillos

Pregunta sí o no con `case`

```
#!/bin/bash

echo –n "¿vale? "
read respuesta

case $respuesta in
   [nN]*)          echo "Ha dicho que no" ;;
   [yY]*|[sS]*)    echo "Ha dicho que si" ;;
   *)              echo "No entiendo" ;;
esac
```

Pregunta sí o no con `if`

```
#!/bin/bash

echo −n "¿vale? "
read respuesta

if [ "$respuesta" = y ]
then
  echo "Ha dicho que si"
elif [ "$respuesta" = n ]
then
  echo "Ha dicho que no"
else
  echo "No entiendo"
fi
```

Los ficheros más nuevos

```
ultimos () {
   ls -lrt | tail -$1
}

# invocando

ultimos 5
```

Barrer ficheros

```
for fichero in *.c
do
   echo Examinando $fichero
   wc $fichero
done
```

Contar líneas de ficheros

```
#!/bin/sh

i=0
cat $1 |
while read linea
do
  i=‘expr $i + 1‘
  echo "$i líneas en $1"
done | tail −1
```

Los bloques redirigidos son procesos (con sus variables).

Contar líneas de ficheros con aritmética de bash

```
#!/bin/bash

i=0
cat $1 |
while read linea
do
  i=$(($i+1))
  echo "$i líneas en $1"
done | tail -1
```

Esperar a que fichero exista

```
#!/bin/sh

until [ -f $1 ]
do
  echo "No veo el fichero $1"
  sleep 5
done
echo "El fichero $1 ya ha aparecido"
```

mcal: cal modificado

```
#!/bin/sh
# interfaz agradable a cal
PATH=/bin:/usr/bin
LANG=es_ES
case $# in
   0) set ‘date‘; mes=$2; año=$6 ;;
   1) mes=$1; set ‘date‘; año=$6 ;;
   *) mes=$1; año=$2 ;;
esac
case $mes in
   ene*|Ene*)     mes=1 ;;
   feb*|Feb*)     mes=2 ;;
   mar*|Mar*)     mes=3 ;;
   abr*|Abr*)     mes=4 ;;
   may*|May*)     mes=5 ;;
   jun*|Jun*)     mes=6 ;;
   jul*|Jul*)     mes=7 ;;
   ago*|Ago*)     mes=8 ;;
   sep*|Sep*)     mes=9 ;;
   oct*|Oct*)     mes=10 ;;
   nov*|Nov*)     mes=11 ;;
   dic*|Dic*)     mes=12 ;;
   [1−9]|1[012]) ;;
   *) año=$mes; mes="" ;;
esac
cal $mes $año
```

Reemplazo de argumentos

```
$ LANG=es_ES
$ date
dom oct 14 20:38:17 CEST 2001
$ set ‘date‘
$ echo "hoy es $1, dia $3 de $2 de $6"
hoy es dom, dia 11 de oct de 2001
```

Ejecución con trazas

```
$ sh -x mcal
+ PATH=/bin:/usr/bin
+ LANG=es_ES
++ date
+ set Sat Dec 11 19:33:06 CET 1999
dom oct 14 20:38:17 CEST 2001
+ mes=oc
+ año=2001
+ month=10
+ cal 10 2001
    Octubre 2001
do lu ma mi ju vi sá
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

cual

```
#!/bin/sh
# localiza que programa se va a ejecutar

oldpath=$PATH
PATH=/bin:/usr/bin

case $# in
  1) ;;
  *) echo "Uso: $0 programa" 1>&2 ; exit 2 ;;
esac

for dir in `echo $oldpath |
    sed -e 's/^:/.:/' \
        -e 's/::/:.:/g' \
        -e 's/:$/:./' \
        -e 's/:/ /g'`
do
  if [ -x $dir/$1 ]
  then
    echo $dir/$1
    exit 0
  fi
done
exit 1
```

watchfor

```
#!/bin/sh
# espera a que alguien determinado entre

PATH=/bin:/usr/bin

case $# in
  1) ;;
  *) echo "Uso: $0 persona" 1>&2
     exit 1 ;;
esac

until who | egrep "$1"
do
  sleep 60
done
```

watchwho

```
#!/bin/sh
# Mira quien entra y quien sale

PATH=/bin:/usr/bin
new=/tmp/who1
old=/tmp/who2

cp /dev/null $old

while true
do
  who > $new
  diff $old $new |
     awk '/>/ { $1= "in: "; print }
          /</ { $1= "out: "; print }'
  mv $new $old
  sleep 1
done
```

Problema: `/tmp/who*` pueden colisionar.

watchwho sin colisión de temporales

```
#!/bin/sh
# Mira quien entra y quien sale, sin colision

PATH=/bin:/usr/bin
new=/tmp/who1.$$
old=/tmp/who2.$$

cp /dev/null $old

while true
do
  who > $new
  diff $old $new |
     awk '/>/ { $1= "in:  "; print }
          /</ { $1= "out: "; print }'
  mv $new $old
  sleep 1
done
```

Problema: `/tmp/who[12].$$` pueden existir o ser un enlace.

watchwho seguro

```
#!/bin/sh
# Mira quien entra y quien sale (seguro)

PATH=/bin:/usr/bin
new=‘mktemp /tmp/who1.XXXXXX‘ || exit 1
old=‘mktemp /tmp/who2.XXXXXX‘ || exit 1



while true
do
  who > $new
  diff $old $new |
     awk ’/>/ { $1= "in: "; print }
          /</ { $1= "out: "; print }’
  mv $new $old
  sleep 1
done
```

Problema: `/tmp/who*` se quedan al terminar o ser interrumpida.

watchwho sin dejar basura

```
#!/bin/sh
# Mira quien entra y quien sale, limpiando /tmp

PATH=/bin:/usr/bin
new=‘mktemp /tmp/who1.XXXXXX‘ || exit 1
old=‘mktemp /tmp/who2.XXXXXX‘ || exit 1

trap ’rm -f $new $old; exit 1’ 0 1 2 15

while true
do
  who > $new
  diff $old $new |
    awk ’/>/ { $1= "in: "; print }
         /</ { $1= "out: "; print }’
  mv $new $old
  sleep 60
done
```

Se trata la salida del programa, la interrupción, el colgado y el asesinato.

Señales

`trap` *cmd sig*
cmd se ejecuta cuando *sig*.
`trap ''` *sig*
ignora *sig*.
`trap` *sig*
pone *sig* acción básica.

0	salida de shell
1	colgar
2	interrumpir `^C`
3	interrupción con `core` `^\`
9	asesinar (inevitable, inatrapable)
13	escribe en pipe sin lector
15	terminación por programa (kill)

Ejercicios

1. Ejercitar los ejemplos.
2. Modificar `contar` para que use `bc`, en lugar de `expr`.
3. Hacer un programa que use el procesador tantos segundos como indica su primer parámetro.
4. Buscar los procesos que lleven más de un cierto gasto de procesador (usuario + sistema). Para ello puede usarse

   ```
   ps -eo pid,time,command
   ```

   (ver `man ps`) o buscar en

   ```
   /proc/[1-9]*/stat
   ```

   la suma tiempo de usuario y de sistema en *jiffies* (1/100 de segundo). Ver `man 5 proc`. Probarlo con el programa anterior.
5. Modificar el programa anterior para que, para cada proceso que encuentre, nos pregunte si queremos matarlo o no (debe informarnos el nombre del programa).