PROGRAMACIÓ DE *SHELLSCRIPTS*

L'objectiu d'aquesta sessió és explicar els fonaments de la programació de *shellscripts* sobre l'**intèrpret de comandes `bash`**.

INTRODUCCIÓ

Tots els intèrprets de comandes incorporen un llenguatge de programació amb sentències de control de fluxe, assignació de variables, funcions,... Els usuaris poden escriure programes (*shellscripts*) utilitzant aquest llenguatge i així automatizar l'execució de seqüències de comandes. Els *shellscripts* seran interpretats pel *shell*.

Per crear un *shellscript* invocareu un editor de textos i escriureu el codi corresponent. La primera línia ha de ser `#!/bin/bash` (indica el *shell* que interpretarà el programa). Un cop gravat, heu d'activar el permís d'execució del *shellscript* amb la comanda `chmod 700 nom_script`, i ara ja el podeu executar utilitzant el nom del *shellscript*.

Podem utilitzar el caràcter `#` per inserir comentaris dins del *shellscript*. Quan l'intèrpret de comandes troba aquest caràcter, deixa d'interpretar els següents caràcters de la línia.

Per buscar informació al manual del sistema sobre les sentències pròpies del `bash` relacionades la programació de *shellscripts* heu d'executar `man bash`.

COMANDES QUE ACOSTUMEN A ESTAR PRESENTS A SHELLSCRIPTS

A continuació es descriuen algunes comandes que acostumen a estar presents als *shellscripts*, tot i que també poden executar-se fora dels *shellscripts*.

sleep

Atura l'execució del *shellscript* durant el nombre de segons indicat com a paràmetre.

- `prompt$` **`sleep 5`** Retorna el control passats 5 segons

echo

Mostra un missatge per la sortida estàndar. Permet imprimir el valor de les variables.

- `prompt$` **`echo Hola`** Mostra un missatge
- `prompt$` **`echo Valor de home: $HOME`** Mostra el contingut de la variable `HOME`
- `prompt$` **`echo -n Sense salt de línia`** No mostra salt de línia (paràmetre `-n`)
- `prompt$` **`echo -e "\ta\t"`** El paràmetre `-e` activa la interpretació de caràcters especials (com ara el `\t`, que representa el tabulador).

test

Avalua condicions respecte un arxiu (si existeix, si és llegible, si és modificable, si és de tipus directori,...), respecte a cadenes de caràcters (si són iguals,...) o numèriques (igualtat, desigualtat, major que, major o igual que,...). `test` no escriu res per la sortida estàndar, però té un codi d'acabament (que és accessible amb la variable del *shell* `?`) que és 0 si la condició és certa i un valor diferent de 0 si la condició és falsa (noteu que

és el conveni contrari a l'utilitzat pel llenguatge C). Consulteu el manual per veure totes les opcions. Exemples (és necessari respectar els espais entre els paràmetres):

- `prompt$` **`test -d /bin; echo $?`** Escriu `0` perquè `/bin` és un directori
- `prompt$` **`test -w /bin; echo $?`** Escriu `1` perquè no podem escriure a */bin*.
- `prompt$` **`test hola = adeu; echo $?`** Escriu `1` perquè les cadenes són diferents
- `prompt$` **`test 4 -gt 5; echo $?`** Escriu `1` perquè 4 no és més gran que 5
- `prompt$` **`test 3 -ne 6; echo $?`** Escriu `0` perquè 3 no és igual a 6.
- `prompt$` **`test 3 -gt 2 -a 5 -lt 7; echo $?`** Escriu `0` perquè es compleixen ambdues condicions (paràmetre `-a` indica funció *and*). El paràmetre `-o` avaluaria la funció *or* i el paràmetre `!` avaluaria la funció *not*.

expr

Avalua una expressió aritmètica i mostra el resultat per la sortida estàndar. Exemples:

- `prompt$` **`expr 3 + 4`** Mostra un 7 com a resultat. Observeu que és precís que existeixi algun espai en blanc entre els operands i l'operador.
- `prompt$` **`expr 3 \* 4`** Mostra un 12. Observeu que és precís protegir l'operador `*` amb el metacaràcter `\` perquè el *shell* no substitueixi `*` pels fitxers del directori actual (l'ús del metacaràcter `\` s'explicarà més endavant).

read

Permet llegir una línia de l'entrada estàndar i guardar-la a una variable. També permet guardar les paraules que composen la línia a variables diferents. Exemples:

- `prompt$` **`read lin; echo L: $lin`** Carrega la línia llegida a la variable `lin`
- `prompt$` **`read word1 word2; echo L1: $word1; echo L2: $word2`** Carrega la primera paraula llegida a `word1`; la segona paraula (i la resta, si és que n´hi ha) és guardaran a `word2`.

exit

Finalitza l'execució del `shellscript`.

true/false

Comandes que es fan servir per generar les condicions de control dels bucles infinits.

COMETES

El *shell* disposa de tres metacaràcters de tipus cometa:

- Cometes simples (`' '`): Indiquen que cal interpretar literalment la cadena que hi ha entre les cometes (és a dir, sense expandir metacaràcters, sense interpretar els espais en blanc com a separadors,...). Per interpretar literalment un caràcter també es pot utilitzar el metacaràcter `\`.
- Cometes dobles (`" "`): Difereixen de les anteriors en què únicament permeten interpretar el metacaràcter `$` per remplaçar el valor de les variables.
- Cometes inverses (`` ` ` ``): Indiquen que cal executar la comanda que hi ha entre les cometes i utilitzar el resultat de l'execució com a paràmetre d'una altra comanda. Aquestes cometes permeten interpretar el metacaràcter `$`.

Exemples:

- `prompt$` **`echo $PATH *; echo '$PATH *'; echo "$PATH *"`** Al primer cas mostra el valor de la variable `PATH` i la llista de fitxers del directori actual. Al segon mostra la cadena literalment. Al tercer únicament substitueix el valor de `PATH`.
- `prompt$` **`ls -l `which sort``** Utilitzem el resultat d'executar la comanda `which sort` com a paràmetre del `ls`.
- `prompt$` **`echo \*; echo *`** Mostra com `\` inhibeix l'expansió del metacaràcter `*`

VARIABLES

Respasseu el que es va explicar sobre les variables a la sessió anterior.

CONTROL DE FLUXE

A continuació es descriuen les estructures de control de fluxe del `bash`.

if

La sintaxi d'aquesta sentència és:

```
if condició1
then
    sentències1
[ elif condicio2
  then
      sentències2 ]
...
[ else
      sentències3 ]
fi
```

Aquesta sentència avalua la *condició*. Si és certa executa *sentències1*; si és falsa i *condició2* és certa, executa *sentències2*,...; si totes les condicions són falses, executa *sentències3*.

A `condició` tindrem una o més comandes i, en funció del codi d'acabament de la darrera d'aquestes comandes, es decidirà si la condició es compleix o no (repasseu la comanda `test`). Les formes típiques de generar les *condicions* són:

- La comanda `test`
- La comanda `grep` (retorna cert si ha trobat la paraula).
- La resta de comandes acostumen a retornar cert si han pogut fer correctament la seva tasca i fals en cas d'error. En cas de dubte, consulteu el manual del sistema de la comanda i busqueu on es parli dels codis d'acabament (*return codes*).
- Si indiqueu vàries comandes comunicades mitjançant *pipes*, es fa servir el valor retornat per la darrera comanda.

while/until

La sintaxi d'aquestes sentències és:

```
while condició
do
    sentències
done
```

```
until condició
do
    sentències
done
```

La sentència `while` avalua la *condició*. Si el seu resultat és cert, executa les sentències de l'interior de bucle i torna a iterar; si és falsa, surt del bucle. La sentència `until` itera mentre la *condició* s'avaluï com a falsa. La condició es genera de la mateixa forma que la condició de la sentència `if`.

for

La sintaxi d'aquesta sentència és:

```
for variable in llista_valors
do
    sentències
done
```

La sentència `for` itera sobre cadascun dels elements de *llista_valors* i executa les sentències existents a l'interior del bucle. A cada iteració, *variable* pren el valor de l'element corresponent de *llista_valors*.

llista_valors es pot generar de moltes formes. Algunes de les més típiques són:

- `*` per iterar sobre els noms de fitxer del directori actual
- `` `comanda` `` per iterar sobre cada paraula mostrada per l'execució de `comanda`
- `$*` per iterar sobre els paràmetres del *shellscript* (explicat més endavant)

case

La sintaxi d'aquesta sentència és:

```
case paraula in
    patró1) sentències1 ;;
    patró2) sentències2 ;;
    ...
    patróN) sentènciesN ;;
esac
```

La sentència `case` busca quin és el primer patró corresponent a paraula i executa les sentències associades al patró.

La forma més típica de generar *paraula* és fent un accés a una variable. Per especificar *patró* podem fer servir metacaràcters: per exemple `a*` és el patró corresponent a paraules que comencen per `a`, el patró `a*|b*` correspon a paraules que comencen per `a` o per `b`, el patró `*` correspon a totes les paraules (acostuma a posar-se com a darrer patró per tractar el cas que una paraula no coincideixi amb cap dels patrons anteriors).

Comentaris finals respecte les sentències de control de fluxe

- Dins dels bucles `for`, `while` i `until` poden utilitzar les sentències `break` (fa sortir del bucle) i `continue` (fa que es comenci a executar la següent iteració).
- És possible redireccionar l'entrada estàndar i/o la sortida estàndar d'una sentència de tipus `for`, `while` i `until`. Aquesta redirecció afecta totes les comandes executades dins del bucle.
- Oblidar alguna de les paraules clau d'una sentència de control de fluxe (com ara `done`) o no tancar una cometa, pot provocar l'error d'execució *Unexpected EOF*.

PAS D'ARGUMENTS ALS *SHELLSCRIPTS*

Quan invoquem un *shellscript* podem passar-li arguments. Per accedir-hi cal considerar:

- `$#` permet obtenir el nombre de paràmetres del *shellscript*.
- `$*` conté la llista de tots els paràmetres del *shellscript*.
- El primer paràmetre és accessible amb `$1`, el segon amb `$2`,..., el novè amb `$9`.
- Si el *shellscript* té més de 9 paràmetres, per accedir als següents cal executar la comanda `shift`. Aquest comanda descarta el valor de `$1`, mou el valor de `$2` a `$1`, el de `$3` a `$2`,..., el de `$9` a `$8`, i posa el valor del primer paràmetre no accessible a `$9`. Aquesta comanda actualitza el valor de `$#` i `$*`. L'efecte de la comanda `shift` no pot ser desfet.
- La variable `$0` sempre conté el nom del *shellscript*.

Quan un *shellscript* espera arguments, és aconsellable comprovar que el valor de `$#` és l'esperat.