Unitat 7 · Sistemes Operatius (SO)

Shell Scripting

Jordi Mateo jordi.mateo@udl.cat

Escola Politècnica Superior (EPS) https://www.eps.udl.cat/ · Departament d'Enginyeria Informàtica i Disseny Digital https://deidd.udl.cat/

Introducció

Què és una shell?

L'intèrpret de comandes d'UNIX, conegut com a shell, és una part fonamental del sistema operatiu que permet als usuaris interactuar amb el nucli del sistema i executar ordres. A lo llarg de la història d'UNIX, diversos programadors han creat intèrprets de comandes amb diferents funcionalitats i preferències personals. Entre els més destacats es troben:

- **sh**: Bourne shell, la primera shell de UNIX, creada per Stephen Bourne.
- **csh**: C shell, creada per Bill Joy.
- **ksh**: Korn shell, creada per David Korn.
- **bash**: Bourne again shell, creada per Brian Fox i Chet Ramey.
- **zsh**: Z shell, creada per Paul Falstad.
- **tcsh**: Ténex C shell, creada per Ken Greer.

```
echo $SHELL # Mostra la shell actual
cat /etc/shells # Mostra les shells disponibles
```

Comandes d'un sistema basat en UNIX

Una comanda és una seqüència de paraules separades per espais. La primera paraula és el nom de la comanda (**cat**) i les següents són els seus arguments (**/etc/shells**). Els arguments poden ser *opcions* o *paràmetres*. Les **opcions** són paraules que comencen amb el caràcter **-** i modifiquen el comportament de la comanda. Els paràmetres són paraules que **no comencen amb -** i són utilitzats per la comanda per a realitzar la seva funció.

```
cat -n /etc/shells
# Mostra el contingut del fitxer /etc/shells amb els números de línia
# cat és una comanda que mostra el contingut d'un fitxer
# -n és una opció que mostra els números de línia
# /etc/shells és un paràmetre que indica el fitxer a mostrar
```

Per coneixer les opcions disponibles d'una comanda, es pot utilitzar la opció **–help** o **-h**. Per exemple, **cat –help** o **cat -h**.

Execució de comandes

Quan una comanda s'executa, l'intèrpret de comandes inicia la seva execució en el context d'un nou procés (**sub-shell**) i espera que finalitzi abans de continuar (*execució en primer pla*). El resultat de l'execució d'una comanda és un valor que reflecteix l'estat de finalització. Convencionalment, un valor diferent de zero indica que s'ha produït algun error.

Per exemple, la comanda cat /etc/shells retorna zero, ja que s'ha executat correctament. En canvi, la comanda cat /etc/shell retorna un valor diferent de zero, ja que el fitxer /etc/shell no existeix.

```
cat /etc/shells # Retorna 0
echo $? # Mostra el valor de retorn de l'última comanda executada
cat /etc/shell # Retorna 1
echo $? # Mostra el valor de retorn de l'última comanda executada
```

Què és un shell script? (I)

Un script de bash és un fitxer de text que conté una seqüència de comandes de bash. Aquestes comandes poden ser una llista d'accions independents a executar, o bé una seqüència lògica d'accions. Per exemple, si volem un script que ens mostri la llista de shells disponibles al sistema, podem crear un fitxer anomenat list_shells.sh amb el següent contingut:

```
cat << EOF > list_shells.sh
#!/bin/bash
cat /etc/shells
EOF
```

Què és un shell script? (II)

Què fa aquesta comanda? Crea un fitxer anomenat **list_shells.sh** amb el contingut indicat. Aquest contingut és una seqüència de comandes que s'executaran quan s'executi el script. La comanda `cat` redirigeix la seva entrada estàndard (stdin) al fitxer **list_shells.sh**. La comanda `<<` indica que la seqüència de comandes finalitza quan es troba la paraula **EOF** (End Of File). I la comanda `>` redirigeix la sortida estàndard (stdout) de la comanda `cat` al fitxer **list_shells.sh**.

Què fa aquest script? Mostra per pantalla el contingut del fitxer /etc/shells. La primera línia del script `#!/bin/bash` indica que s'ha d'executar amb l'intèrpret de comandes bash. Aquesta línia s'anomena *shebang* i és obligatòria en tots els scripts de bash.

Execució d'un shell script

Per executar un script de bash, s'ha d'indicar el nom del fitxer que conté el script. Per exemple, per executar el script list_shells.sh, s'ha d'executar la comanda `bash list_shells.sh`.

```
bash list_shells.sh # Executa el script list_shells.sh
```

o bé, podem donar permisos d'execució al fitxer i executar-lo directament.

```
chmod +x list_shells.sh # Dona permisos d'execució al fitxer list_shells.sh
./list_shells.sh # Executa el script list_shells.sh
```

Objectius del Shell Scripting

- Automatitzar tasques repetitives.
- Crear programes senzills o complexos.
- Crear comandes pròpies.
- Crear scripts d'instal·lació.
- Crear scripts de configuració.
- Crear scripts de manteniment.
- Crear tests d'aplicacions.

Automatitzar tasques repetitives

```
#!/bin/bash
# Aquest script permet instal·lar el compilador de C gcc
# a partir del codi font.
VERSION=9.3.0 # Versió del compilador de C
# Descarrega el paquet
wget https://ftp.gnu.org/gnu/gcc/gcc-$VERSION/gcc-$VERSION.tar.gz
tar -xzf gcc-$VERSION.tar.gz # Descomprimeix el paquet
mkdir gcc-$VERSION-build # Crea un directori per a la compilació
cd gcc-$VERSION-build # Configura el paquet
../gcc-$VERSION/configure --enable-languages=c,c++ --disable-multilib
make -j$(nproc) # Compila el paquet
make install # Instal·la el paquet
```

Comanda: Echo

La comanda `echo` permet mostrar text per pantalla.

Sinòpsis:

- `echo [SHORT-OPTION]... [STRING]...`
- `echo LONG-OPTION`

Opcions principals:

`-n`: No imprimeix la línia final de sortida.

```
echo -n "Pikachu, t'elegeixo a tú!"
```

```
# 4 arguments separats per espais en blanc
echo Pikachu, t\'elegeixo a tú!
# 1 argument amb cometes simples
echo 'Pikachu, t\'elegeixo a tú!'
# 1 argument amb cometes dobles
echo "Pikachu, t'elegeixo a tú!"
```

Quina és la diferència entre utilitzar cometes simples i cometes dobles? Les cometes dobles us permeten utilitzar variables.

```
NOM="Pikachu"
echo "$NOM, t'elegeixo a tú!"
echo '$NOM, t\'elegeixo a tú!'
echo $NOM, t\'elegeixo a tú!
```

Printf

La comanda `printf` permet mostrar text per pantalla amb un format determinat.

Opcions principals:

- %s: cadena de text. %d: nombre enter. %f: nombre decimal. %-{val}: alineació a l'esquerra amb un espai de {val} caràcters. %.{val}: nombre de decimals a mostrar.

```
#!/bin/bash
printf "%-15s %-15s %-15s %-15s %-15s %-15s\n" \
"Nom" "Tipus1" "Tipus2" "Atac" "Defensa" "Velocitat"
printf "%-15s %-15s %-15s %-15d %-15.2f %5.1f\n" \
"Bulbasaur" "Planta" "Verí" 49 49,123 45,67
printf "%-15s %-15s %-15s %-15d %-15.2f %5.1f\n" \
"Charmander" "Foc" "" 52 43,89 65,21
printf "%-15s %-15s %-15s %-15d %-15.2f %5.1f\n" \
"Squirtle" "Aigua" "" 48 65,42 43,78
```

Caràcter d'escapament

Els caràcters `\` al final de cada línia indiquen que la comanda continua a la següent línia. Aquest caràcter s'anomena caràcter d'escapament i es pot utilitzar amb qualsevol comanda.

Això ens permet escriure el codi de forma més llegible.

Fitxers d'inicialització

L'intèrpret de comandes **/bin/bash** fa ús d'una sèrie de fitxers d'inici per establir un entorn propici per a la seva execució. Cada fitxer té una funció específica i pot influir de manera diversa en els entorns interactius dels usuaris. En sistemes basats en el kernel de linux, els fitxers del directori */etc* generalment proporcionen configuracions globals, mentre que si hi ha un equivalent al vostre directori personal, aquest pot prevaldre sobre les configuracions globals.

Quan una shell s'inicia després d'una autenticació correcta al sistema, mitjançant **/bin/login** i llegint el fitxer **/etc/passwd**, l'intèrpret normalment processa **/etc/profile** i el seu equivalent privat, **~/.bash_profile**.

- **/etc/profile**: S'executa quan qualsevol usuari inicia la sessió.
- **/etc/bashrc**: S'executa cada cop que qualsevol usuari executa el programa bash.
- **~/.bash_profile**: S'executa quan l'usuari inicia la sessió.
- **~/.bashrc**: S'executa quan l'usuari executa un programa bash.
- **~/.bash_logout**: S'executa quan l'usuari tanca la sessió.

Exemples de configuració

Normalment a *.bashrc* es defineixen variables d'entorn i al *.bash_profile* es defineixen variables d'usuari.

```
echo "export EDITOR=nano" >> ~/.bashrc
echo "export HISTSIZE=1000" >> ~/.bashrc
```

```
echo "Welcome to debian!" >> ~/.bash_profile
echo "export PATH=$PATH:/home/jordi/bin" >> ~/.bashrc
```

Bash scripting (Bàsic)

Variables

Les variables en la programació Bash s'utilitzen per emmagatzemar i manipular dades. A Bash, les variables es poden assignar valors utilitzant l'operador =, i els seus valors es poden recuperar utilitzant el prefix $.

```
nom="Charmander"
echo $nom
# Sortida: Charmander
```

Els espais en blanc no són vàlids entre el nom de la variable i l'operador =. Si els utilitzes, la comanda no funcionarà correctament.

```
nom = "Pikachu"
bash: nom: no s'ha trobat l'ordre
```

Operació unset

La comanda **unset** s'utilitza per desassignar o eliminar el valor d'una variable. Quan una variable es desassigna, es converteix en buida o indefinida. Aquí tens un exemple:

```
nom="Charmander"
echo $nom
# Sortida: Charmander

unset nom
echo $nom
# Sortida: (buida, ja que la variable s'ha desassignat)
```

Característiques de les variables

Les variables Bash són de tipus dinàmic, el que permet que puguin contenir diferents valors al llarg del temps. Pots assignar un nou valor, com ara *Bulbasaur*, a la mateixa variable **nom**. Novament, fent servir la comanda `echo`, podem mostrar el valor actualitzat de **nom**. La sortida serà *Bulbasaur*.

```
nom=Bulbasaur
echo $nom
# Sortida: Bulbasaur
```

També pots assignar un valor buit a la variable **nom**. La variable **nom** es defineix, però no té cap valor. Quan fem servir la comanda `echo` per mostrar el valor de **nom**, es mostrarà una línia buida.

Interpolació de variables

La interpolació de variables us permet substituir el valor d'una variable dins d'una cadena de text. Per fer-ho heu d'utilitzar cometes dobles i el prefix $.

```
name="Bulbasaur"
echo "El meu nom és $name"
# Output: El meu nom és Bulbasaur
```

En escenaris més complexos, podeu utilitzar les claus {} per delimitar el nom de la variable.

```
suffix="saur"
echo "Bulba${suffix}"
# Output: Bulbasaur
echo "Ivy${suffix}"
# Output: Ivysaur
echo "Venu${suffix}"
# Output: Venusaur
```

Espais en blanc

Considereu l'exemple següent: la variable nom rep el valor Ash Ketchum, que inclou diversos espais consecutius. Quan utilitzeu l'ordre echo $nom, la variable s'expandeix dins de les cometes dobles. Bash realitza una divisió de paraules i tracta els espais consecutius com a delimitador, de manera que la sortida és Ash Ketchum amb els espais addicionals col·lapsats en un.

```
nom="Ash          Ketchum"
echo $nom
# Sortida: Ash Ketchum
echo "$nom"
# Sortida: Ash          Ketchum
```

Enters (`let`)

`let` us permet realitzar operacions aritmètiques i assignacions dins dels scripts de Bash. S'utilitza habitualment per a expressions aritmètiques senzilles. Suposem que tenim dos Pokémon al nostre equip, Pikachu i Charmander, amb els valors de poder de combat (CP) respectius de 500 i 300. Volem calcular el CP total quan el CP de Pikachu es duplica i s'afegeix al CP de Charmander.

```
#!/bin/bash
pikachu_cp=500
charmander_cp=300

let total_cp=pikachu_cp*2+charmander_cp

echo "CP Total: $total_cp"
# Sortida: CP Total: 1300
```

Enters (`expr`)

L'ordre expr avalua i realitza operacions aritmètiques en expressions proporcionades com a arguments de línia de comandes o dins de scripts. Calculem la mitjana dels punts d'experiència (EXP) bàsics de tres Pokémon: Bulbasaur (64), Squirtle (63) i Charmander (62).

```
#!/bin/bash
bulbasaur_exp=64
squirtle_exp=63
charmander_exp=62

mitjana_exp=$(expr \( $bulbasaur_exp + $squirtle_exp + $charmander_exp \) / 3)

echo "Mitjana EXP: $mitjana_exp"
# Sortida: Mitjana EXP: 63
```

Enters (`$(())`)

La sintaxi `$(( ))` us permet realitzar operacions aritmètiques directament dins de la consola o en la substitució de comandes. Calculem la suma dels valors de les estadístiques bàsiques (HP, Atac, Defensa, Atac Especial, Defensa Especial i Velocitat) d'un Pikachu.

```
pikachu_hp=35
pikachu_atac=55
pikachu_defensa=40
pikachu_atac_especial=50
pikachu_defensa_especial=50
pikachu_velocitat=90

estadistiques_totals=$((pikachu_hp + pikachu_atac + pikachu_defensa +
pikachu_atac_especial + pikachu_defensa_especial + pikachu_velocitat))

echo "Estadístiques Totals: $estadistiques_totals"
# Sortida: Estadístiques Totals: 320
```

Operadors aritmètics

Operador	Descripció
+,-,*,/,%	Suma,Resta,Multiplicació,Divisió,Mòdul
++,–	Increment,Decrement
+=,-=,*=,/=,%=	Assignació amb operació aritmètica
**	Exponenciació
«,»	Desplaçament de bits
&&,	
!	Operador lògic NOT
>,<,>=,<=	Operadors de comparació
==,!=	Operadors de comparació

Arrays

Els arrays s'utilitzen per emmagatzemar múltiples valors en una única variable. Els arrays de Bash poden contenir elements de diferents tipus de dades. Per exemple, un array pot contenir cadenes de text, enters. Els arrays de Bash són de tipus dinàmic, el que permet que puguin contenir diferents valors al llarg del temps.

```
# Array indexat (Vector)
tipus=("Aigua" "Foc" "Verí")

# Array associatiu (Diccionari)
declare -A pokedex
pokedex=([Bulbasaur]="1" [Charmander]="4" [Squirtle]="7")
```

Un **array indexat** és un array en què cada element té un índex numèric únic. Els índexs d'un array indexat comencen amb 0 i van fins a n-1, on n és el nombre d'elements de l'array. Un **array associatiu** és un array en què cada element té una clau única. Les claus d'un array associatiu poden ser cadenes de text o enters.

Operadors d'arrays (Accedir als elements)

```
# Array indexat
echo ${tipus[2]}
# Sortida: Verí

# Array associatiu
echo ${pokedex["Bulbasaur"]}
# Sortida: 1
```

Operadors d'arrays (Actualitzar els elements)

```
# Array indexat
tipus[1]="Elèctric"
echo ${tipus[1]}
# Sortida: Elèctric

# Array associatiu
pokedex["Charmander"]="5"
echo ${pokedex["Charmander"]}
# Sortida: 5
```

Operadors d'arrays (Afegir elements)

```
# Array indexat
tipus[3]="Planta"
echo ${tipus[3]}
# Sortida: Planta

# Array associatiu
pokedex["Pikachu"]=25
echo ${pokedex["Pikachu"]}
# Sortida: 25
```

Operadors d'arrays (Eliminar els elements)

```
# Array indexat
unset tipus[2]
echo ${tipus[2]}
# Sortida: (buit)

# Array associatiu
unset pokedex["Charmander"]
echo ${pokedex["Charmander"]}
# Sortida: (buit)
```

Operadors d'arrays (Obtenir la longitud)

```
# Array indexat
echo ${#tipus[@]}
# Sortida: 3

# Array associatiu
echo ${#pokedex[@]}
# Sortida: 3
```

Exemple de l'ús d'arrays (Pokèdex)

```bash
pokedex=("Bulbasaur" "Charmander" "Squirtle")
echo "Entrades de la Pokédex:"
for pokemon in "${pokedex[@]}"
do
    echo "- $pokemon"
done

# Sortida:
# Entrades de la Pokédex:
# - Bulbasaur
# - Charmander
# - Squirtle
```

Constants

Podeu declarar variables com a de només lectura per evitar que els seus valors es modifiquin posteriorment al script. Això és útil quan voleu definir constants o configuracions que haurien de romandre inalterades. Podeu declarar una variable com a de només lectura utilitzant la comanda **readonly** o la comanda declare amb l'opció **-r**.

```
readonly TIPUS1='foc'
declare -r TIPUS2='aigua'
echo "Tipus1: $TIPUS1" i "Tipus2: $TIPUS2"
TIPUS1='planta'
# Sortida: bash: TIPUS1: variable de només lectura
TIPUS2='planta'
# Sortida: bash: TIPUS2: variable de només lectura
```

Les constants són variables que no poden canviar el seu valor durant l'execució d'un script. Per convenció, els noms de les constants es defineixen en majúscules.

Tipat de variables

Bash no és un llenguatge de programació tipat. No obstant això, `declare` pot ser utilitzat amb atributs i tipus de dades específics per proporcionar més control sobre la creació de variables en comparació amb la simple assignació utilitzant l'operador =.

```
declare name="Charmander"
declare -i id=1
declare -a skills=("Fire Blast" "Ember" "Flamethrower")
declare -r type="fire"
echo "Nom: $name"
 # Sortida: Charmander
echo "Id: $id"
# Sortida: 1
echo "Habilitats: ${skills[@]}"
# Sortida: Fire Blast Ember Flamethrower
echo "Tipus: $type"
# Sortida: fire
```

Àmbit de les variables

De forma predeterminada, les variables tenen un àmbit restringit dins de la consola, la qual cosa significa que només són accessibles dins del context del script o la sessió de consola on estan definides. No obstant això, hi ha maneres de fer que les variables siguin accessibles als processos posteriors que s'executen a la mateixa consola.

```
nom="Charmander"
export tipus="foc"
echo "nom: $nom"
# Sortida: Charmander
echo "tipus: $tipus"
# Sortida: foc
```

En aquest exemple, definim dues variables: **nom** i **tipus**. **Nom** té un àmbit restringit i només es pot accedir dins del script o de la sessió de consola on està definit. D'altra banda, la variable **tipus**, que s'exporta, es converteix en una variable d'entorn i és accessible pels processos posteriors dins de la mateixa sessió de consola.

Exemple de l'àmbit de les variables

```
#!/bin/bash
nom="Charmander"
export tipus="foc"
echo "nom: $nom"
# Sortida: Charmander
echo "tipus: $tipus"
# Sortida: foc
bash
echo "nom: $nom"
# Sortida: (buit, ja que la variable està sense definir)
echo "tipus: $tipus"
# Sortida: foc (definit a l'entorn)
```

Com podeu veure, la variable **tipus** és accessible després de `bash` perquè es va exportar i el procés fill creat amb `bash` coneix el seu valor. No obstant això, la variable **nom** no és accessible perquè té un àmbit restringit i està limitada al context del procés pare.

Variables d'entorn

Les variables d'entorn es defineixen a l'entorn de la consola i són accessibles per a tots els programes i scripts que s'executen en aquest entorn. Depenen del context i són heretades pels processos fills del procés pare. Les variables d'entorn també es defineixen en majúscules i s'utilitzen per configurar el comportament de la consola i altres programes.

En els sistemes Linux, hi ha algunes variables ambientals comunes:

- **PATH**: Especifica els directoris on el sistema operatiu busca programes executables. Per exemple: /usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
- **HOME**: Apunta al directori d'inici de l'usuari actual. Per exemple: /home/jordi
- **USER**: Emmagatzema el nom d'usuari de l'usuari actualment identificat. Per exemple: jordi
- **PWD**: Representa el directori de treball actual. Per exemple: /home/jordi
- **LANG**: Determina la llengua i les configuracions de localització. Per exemple: es_CA.UTF-8

Variables d'entorn (II)

```
echo $SHELL
# Sortida: /bin/bash
echo $PWD
# Sortida: /home/jordi
```

Per veure totes les variables definides en un sistema basat en Linux, podeu utilitzar la comanda `set` o `printenv`(debian). La comanda `set` sense arguments mostra totes les variables, incloses les variables d'entorn i les variables definides per l'usuari, juntament amb els seus valors. Si voleu filtrar i mostrar només les variables d'entorn, podeu utilitzar la comanda `printenv`. Aquests comanda es troba disponbile a sistemes debian i permet ignora les variables definides per l'usuari i es centra només en les variables que s'han exportat i són accessibles pels processos.

Variables especials de Només Lectura

Hi ha diverses variables que proporcionen informació sobre els arguments de la línia de comandes i el procés actual. Aquestes variables són accessibles però no es poden modificar.

- $0: Representa el nom del script o de la mateixa consola.
- $1-$9: Corresponen als arguments d'entrada proporcionats al script o la comanda, amb $1 que representa el primer argument, $2 que representa el segon argument, i així successivament fins a $9. ${10}, ${11}, i així successivament: Representa els arguments d'entrada més grans que 9. Per accedir als arguments amb índexs més grans que 9, heu d'utilitzar claus {}.
- $*: Representa tots els arguments de la línia de comandes com una única cadena de text.
- $@: Representa tots els arguments de la línia de comandes com cadenes de text separades.
- # : Representa el nombre d'arguments de la línia de comandes.
- $!: Representa la identificació del procés del procés en segon pla més recent.
- $?: Representa l'estat de sortida de la comanda o script més recent. Un valor de 0 indica normalment èxit, mentre que un valor no nul indica un error o fallada.

Variables especials de Només Lectura (II)

```bash
#!/bin/bash
# variables_especials.sh
# bash variables_especials.sh arg1 arg2 arg3
echo "Nom del script: $0"
echo "Nombre d'arguments: $#"
echo "Els arguments són: $@"
ls -lh
echo "Codi de sortida de la darrera comanda: $?"
cat wololo
echo "Codi de sortida de la darrera comanda: $?"
echo "El meu PID: $$" # $$ indica el PID del procés actual
top &
echo "PID de la darrera comanda: $!"
kill -9 $!
echo "Codi de sortida de la darrera comanda: $?"
exit 0
```

Manipulació de cadenes de text

```
string="Hello, world!"

# Longitud de la cadena
length=${#string}
echo "Longitud de la cadena: $length"
# Sortida: 13

# Extreure una subcadena des d'una posició
substring_pos=${string:7}
echo "Subcadena a partir de la posició 7: $substring_pos"
# Sortida: world!

# Extreure una subcadena amb una longitud específica a partir d'una posició
substring_length=${string:7:2}
echo "Subcadena de longitud 2 a partir de la posició 7: $substring_length"
# Sortida: wo
```

Manipulació de cadenes de text (II)

```
string="Hello, world!"

# Reemplaçar la primera coincidència de la subcadena amb una altra cadena
replace_first=${string/world/Earth}
echo "Reemplaçar la primera coincidència de 'world' amb 'Earth': $replace_first"
# Sortida: Hello, Earth!

# Reemplaçar totes les coincidències de la subcadena amb una altra cadena
replace_all=${string//l/L}
echo "Reemplaçar totes les coincidències de 'l' amb 'L': $replace_all"
# Sortida: HeLLo, worLd!

# Eliminar coincidències de la subcadena al principi de la cadena
remove_beginning=${string#Hello, }
echo "Eliminar 'Hello, ' del principi de la cadena: $remove_beginning"
# Sortida: world!
```

Manipulació de cadenes de text (III)

```
string="Hello, hello, world!"
# Eliminar la coincidència més curta del patró al davant de la cadena
strip_shortest_front=${string#H*e}
echo "Eliminar la coincidència més curta del patró al davant: $strip_shortest_front"
# Sortida: llo, hello, world!
# Eliminar la coincidència més llarga del patró al davant de la cadena
strip_longest_front=${string##H*e}
echo "Eliminar la coincidència més llarga del patró al davant: $strip_longest_front"
# Sortida: llo, world!
# Eliminar la coincidència més curta del patró al final de la cadena
strip_shortest_back=${string%l*}
echo "Eliminar la coincidència més curta del patró al final: $strip_shortest_back"
# Sortida: Hello, hello, wor
# Eliminar la coincidència més llarga del patró al final de la cadena
strip_longest_back=${string%%l*}
echo "Eliminar la coincidència més llarga del patró al final: $strip_longest_back"
# Sortida: He
```

Substitució de comandes

La substitució de comandes permet utilitzar la sortida d'una comanda com a entrada per a una altra comanda. Hi ha dues formes de substitució de comandes: substitució de comandes amb parèntesis i substitució de comandes amb cometes inverses.

```
# Substitució de comandes amb parèntesis
echo "La data actual és $(date)"
# Substitució de comandes amb cometes inverses
echo "La data actual és `date`"
```

```
dir_list=`ls -l`
echo $dir_list  #No conserva els salts de línia
echo "$dir_list" #Conserva els salts de línia
```

Comanda `exec`

La comanda `exec` permet executar una comanda en el mateix procés. Això significa que la comanda actual es substitueix per la nova comanda i el procés no es crea. La comanda `exec` és útil per reemplaçar el procés actual per un nou procés. Per exemple, si volem que el procés actual sigui reemplaçat per un procés de bash, podem utilitzar la comanda `exec bash`.

```
#!/bin/bash
echo "Abans de l'execució de la comanda exec"
exec bash
echo "Després de l'execució de la comanda exec"
```

Comanda `read`

La comanda `read` permet llegir l'entrada de l'usuari i assignar-la a una variable. La comanda `read` pot llegir l'entrada de l'usuari des de l'entrada estàndard (stdin) o des d'un fitxer.

```
#!/bin/bash
echo "Quin és el teu nom i cognom?"
read nom cognom
echo "Hola $nom $cognom!"
```

La comanda `set`

La comanda `set` s'utilitza per modificar l'entorn d'execució de la consola. Aquesta comanda té diverses opcions que permeten modificar el comportament de la consola. Per exemple, la comanda `set -u` permet que la consola mostri un missatge d'error quan s'intenta utilitzar una variable no definida.

```
#!/bin/bash
set hell to highway $(date)
echo $3 $2 $1
echo $@
# Sortida: highway to hell
# Sortida: hell to highway <data>
```

Control de flux

If-then-else

```
#!/bin/bash
if [ $# -eq 0 ]; then
    echo "No s'ha proporcionat cap argument"
elif [ $# -eq 1 ]; then
    echo "S'ha proporcionat un argument"
else
    echo "S'han proporcionat $# arguments"
fi
```

- **Nota 1**: La comanda `test` és equivalent a `[ ]`. Per exemple, `test $# -eq 0` és equivalent a `[ $# -eq 0 ]`.
- **Nota 2**: Els espais en blanc són importants. Per exemple, `[ $# -eq 0 ]` és correcte, però `[$# -eq 0]` no ho és.

Operadors de comparació

Operador	Descripció
-eq	Igual que
-ne	Diferent que
-gt	Més gran que
-ge	Més gran o igual que
-lt	Menys gran que
-le	Menys gran o igual que
-z	Cadena buida
-n	Cadena no buida
=	Igual que
!=	Diferent que

Exemple de l'ús de comparacions

```bash
#!/bin/bash
declare -a pokemon_ids=(1 4 7)
declare -a pokemon_names=("Bulbasaur" "Charmander" "Squirtle")
declare -a pokemon_types=("grass" "fire" "water")
declare -a pokemon_cp=(100 200 300)

if [ $# -eq 0 ]; then
    echo "No s'ha proporcionat cap argument"
elif [ $# -eq 1 ]; then
    id_index=-1
    for i in "${!pokemon_ids[@]}"; do
        if [ "${pokemon_ids[i]}" -eq "$1" ]; then
            id_index="$i"
            break
        fi
    done

    if [ "$id_index" -ge 0 ] && [ -n "${pokemon_names[id_index]}" ]; then
        echo "Informació del Pokémon amb ID $1:"
        echo "Nom: ${pokemon_names[id_index]}"
        echo "Tipus: ${pokemon_types[id_index]}"
        echo "Punts de combat (CP): ${pokemon_cp[id_index]}"
    else
        echo "No s'ha trobat cap Pokémon amb l'ID $1"
    fi
else
    echo "Només es permet un argument (ID del Pokémon)"
fi
```

Comparacions amb fitxers

Operador	Descripció
-e	Comprova si el fitxer existeix
-r	Comprova si el fitxer existeix i té permisos de lectura
-w	Comprova si el fitxer existeix i té permisos d'escriptura
-x	Comprova si el fitxer existeix i té permisos d'execució
-f	Comprova si el fitxer existeix i és un fitxer regular
-d	Comprova si el fitxer existeix i és un directori
-h	Comprova si el fitxer existeix i és un enllaç simbòlic
-p	Comprova si el fitxer és una **pipe**

Exemple d'operadors de comparació amb fitxers

```
touch fitxer_regular.txt; ln -s fitxer_regular.txt enllac_simbolic.txt
if [ -e fitxer_regular.txt ]; then
    echo "El fitxer fitxer_regular.txt existeix"
        if [ -f fitxer_regular.txt ]; then
            echo "El fitxer fitxer_regular.txt és un fitxer regular"
        fi
    else
        echo "El fitxer fitxer_regular.txt no existeix"
fi
if [ -e enllac_simbolic.txt ]; then
    echo "El fitxer enllac_simbolic.txt existeix"
        if [ -h enllac_simbolic.txt ]; then
            echo "El fitxer enllac_simbolic.txt és un enllaç simbòlic"
        fi
    else
        echo "El fitxer enllac_simbolic.txt no existeix"
fi
```

Exemple d'operadors de comparació amb fitxers (II)

```
chmod u-wx fitxer_regular.txt
mkfifo pipe.txt

if [ -r fitxer_regular.txt ]; then
    echo "El fitxer fitxer_regular.txt té permisos de lectura"
fi

if [ ! -w fitxer_regular.txt ]; then
    echo "El fitxer fitxer_regular.txt no té permisos d'escriptura"
fi

if [ -p pipe.txt ] && [ ! -p fitxer_regular.txt ]; then
    echo "El fitxer pipe.txt és una pipe i fitxer_regular.txt no"
fi
```

Switch-case

```
case <variable>
 in
 pattern1)
 commands1
 ;;
 pattern2)
 commands2
 ;;
 *)
 default_commands
 ;;
esac
```

Exemple de l'ús de switch-case

```
#!/bin/bash
echo "d: data actual"
echo "l: llista de fitxers del directori actual"
echo "q: sortir"
echo "Elegeix una opció: "
read option
case "$option" in
 d|D) date
 ;;
 l|L) ls
 ;;
 q|Q) exit 0
 ;;
 *) echo "Opció incorrecta"
 exit 1
 ;;
esac
```

For

```
for <variable> in <list>
do
 commands
done
```

```
for ((initialization; end_condition; increment))
do
 commands
done
```

Exemple de l'ús de for (I)

```
#!/bin/bash
echo "Prepara't per a una batalla Pokémon!"
for i in 3 2 1
do
  echo "Un Pokémon salvatge apareix! Prepara't per capturar-lo en $i..."
  sleep 1
done
echo "Vés-hi, Pikachu!"
echo "Llançant una Pokéball..."
sleep 2
catch_success=$((RANDOM % 2))
if [ $catch_success -eq 1 ]; then
  echo "Felicitats! Has capturat el Pokémon!"
else
  echo "Oh no! El Pokémon s'ha escapat! Està fugint!"
fi
```

Exemple de l'ús de for (II)

```
echo "Reapareixeràs al Centre Pokémon en 5 segons..."
for ((i=5; i>0; i--))
do
  echo "Reapareixent en $i..."
  sleep 1  # Afegeix un retard per simular la compte enrere
done
echo "Benvingut de nou al Centre Pokémon!."
exit 0
```

While

```
while <expressio>
do
    comandes
done
```

- La comanda `break` serveix per sortir del bucle.
- La comanda `continue` acaba la iteració actual.

Exemple d'ús de `while` (I)

```
#!/bin/bash
cat << EOF > pokemon.txt
Bulbasaur
Charmander
Squirtle
EOF
while read pokemon
do
    echo "Pokémon: $pokemon"
done < pokemon.txt
```

Exemple d'ús de `while` (II)

```
n=0
while [ $n -lt 3 ]
do
  echo "Un Pokémon salvatge apareix!"
  echo "Llançant una Pokéball..."

  prob=$((RANDOM % 2))
  if [ $prob -eq 1 ]; then
    echo "Felicitats! Has capturat un Pokémon!"
    ((nombre_pokemon++))
  else
    echo "Oh no! El Pokémon s'ha escapat! Està fugint!"
    echo "Torna-ho a intentar..."
    continue  # Salta la resta del bucle i comença una nova iteració
  fi

  echo "Has capturat $n Pokémon fins ara."
  if [ $n -eq 3 ]; then
    echo "Has arribat al nombre màxim de Pokémon que pots portar."
    break  # Surts del bucle quan es compleix el màxim
  fi
done
```

Until

```
if [ $# -eq 0 ]; then
 echo "error - number missing"; exit 1
fi
echo “Try to guess the number between 1 and 10”
n=$RANDOM
let "n=(n%10) + 1"
tries=0; guess=0
until [ "$guess" -eq "$n" -o "$tries" -ge $1 ]; do
 echo -n "Write a number: "; read guess
 let "tries=$tries+1"
done
if [ "$n" -eq "$guess" ]
 then echo "Victory ;-)"
 else echo "Defeat ;-("
fi
```

Funcions

Funcions (I)

En bash, les funcions són blocs de codi que s'executen quan es criden. Les funcions són útils per organitzar el codi i evitar la repetició de codi. Les funcions es defineixen utilitzant la sintaxi següent:

```
#!/bin/bash
function sayMyName {
 echo $#  # Nombre d'arguments
 echo $1
 shift
 echo $1
}
sayMyName "My name is" "Ash" "Ketchum"
# Sortida: 3
# Sortida: My name is
# Sortida: Ash
```

Funcions (II)

Les variables definides dins i fora d'una funció són variables i són accessibles.

```
#!/bin/bash
pikachu_hp=35

function reduce_hp {
  damage=$1  # Variable local
  let "pikachu_hp -= damage"
}

echo "Pikachu HP: $pikachu_hp"
echo "Pikachu és atacat!"
reduce_hp 10
echo "damage: $damage"
echo "Pikachu HP: $pikachu_hp"
```

Funcions Variables Globals

Les variables definides dins i fora d'una funció són variables i són accessibles.

```
#!/bin/bash
pikachu_hp=35

function reduce_hp {
  damage=$1  # Variable local
  let "pikachu_hp -= damage"
}

echo "Pikachu HP: $pikachu_hp"
echo "Pikachu és atacat!"
reduce_hp 10
echo "damage: $damage"
echo "Pikachu HP: $pikachu_hp"
```

Funcions Variables Locals

```bash
#!/bin/bash
pikachu_hp=35

function reduce_hp {
  local damage=$1  # Variable local
  let "pikachu_hp -= damage"
}

echo "Pikachu HP: $pikachu_hp"
echo "Pikachu és atacat!"
reduce_hp 10
echo "damage: $damage"
echo "Pikachu HP: $pikachu_hp"
```

Funcions - Valor de Retorn

```bash
#!/bin/bash
pikachu_hp=35

function reduce_hp {
  local hp=$1 # Variable local
  local damage=$2 # Variable local
  let "hp=hp-damage"
  return $hp # Retorn de la funció
}

echo "Pikachu HP: $pikachu_hp"
echo "Pikachu és atacat!"
reduce_hp $pikachu_hp 10
pikachu_hp=$? # Assignació del valor retornat
echo "Pikachu HP: $pikachu_hp"
```

Això és tot per avui

PREGUNTES?

Materials del curs

- **Organització** — OS-GEI-IGUALADA-2425
- **Materials** — Materials del curs
- **Laboratoris** — Laboratoris
- **Recursos** — Campus Virtual

TAKE HOME MESSAGE: Els scripts de bash són útils per automatitzar tasques repetitives i per crear programes senzills.

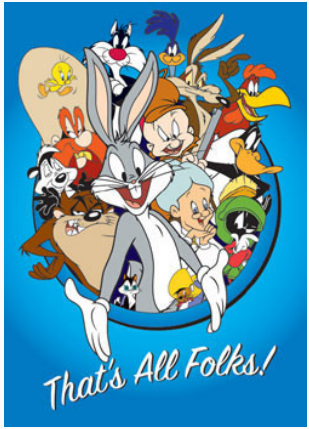

Figura 1: Això és tot per avui