ASR2 – USI 2004

Programmer avec les shell-scripts
Le langage BASH

1. Shell scripts

- Langage permettant de mélanger
  - Des commandes (lancement de programmes)
  - Des structures de contrôle pour les enchaîner automatiquement
  - Des variables
  - etc.

Pourquoi bash ?

- Familles de langages de script : sh, csh etc.
- En commun : script : commandes + structures de contrôle
- Le shell historique d'UNIX est **sh** (Bourne shell)
- Bash : Version évoluée de **sh** (Bourne again shell)

Pourquoi programmer en shell ?

- Automatisation facile des tâches répétitives
- Travaux courants et/ou complexes
- Scripts de démarrage

Exemple de script

```
#!/bin/bash
D='paul ringo'
VER=3.0
DIR=/home/mp3/maquette-$VER
TAR=/tmp/maquette-$VER.tgz
tar cvzf $TAR $DIR
mutt -a $TAR -s "maquette $VER"  $D
  <<XX
Ci-joint la version $VER de la maquette
XX
```

2. Les variables

- Une variable contient une chaîne de caractères. Affectation

```
titre='les shell scripts'
soustitre="les variables"
```

- Expansion

```
echo Cours intitulé $titre
echo ${version}bis
```

Quotes

- Les apostrophes (*quotes*) empêchent l'expansion

```
echo 'prix = 35$US'
```

- Le caractère d'échappement \

```
echo "Prix = $prix\$US"
echo "Il dit \"c'est l'heure\""
echo 'Il dit "c\'est l\'heure"'
```

read

- L'instruction **read**
  - lit une ligne de texte (liste de mots) sur l'entrée standard
  - la place dans des variables
- Exemple

  read prenom nom
- Un mot par variable, le reste dans la dernière

Export

- Un processus hérite des **variables d'environnement** de son père
- Marquage d'une variable du shell pour l'export (vers processus fils)

```
TEMPERATURE=18
export TEMPERATURE
```

- ou

```
export TEMPERATURE=18
```

Paramètres positionnels

- Les paramètres formels (**arguments**) d'un script sont $1 $2 $3... ce sont des paramètres positionnels
- $# = le nombre effectif d'arguments
- $0 = le chemin d'accès du script
- $* = la liste des arguments

Redirections

- Depuis un fichier

  mail sophie < lettre.txt

- Depuis le script

  mail godot <<XXX

  je crois que quelqu'un

  vous attend en salle $SALLE

  XXX

Redirections (suite)

- Vers un fichier

  >f >>f 2>f 2>>f

- Entre processus
  - ls -lR ~ | gzip -c > liste.tgz

Expansion de commande

- La sortie d'une commande peut être redirigée vers une chaine de caractères

```
MESSAGE="Il est $(date +%T)"
```

- Autre notation

```
MESSAGE="Il est `date +%T`"
```

« compatible sh », mais moins lisible.

Arithmétique

- Calculs en nombres entiers

```
let prix=quantite*unitaire
let stock-=quantite
```

- Expansion arithmétique

```
echo vous devez $((total-acompte)) Eur
```

3. Code de retour et conditions

- Un processus s'achève renvoie un **code de retour (status)**
- Par convention, **0 = OK**
- En bash, la variable **$?** contient le status de la dernière commande exécutée

Les conditions

- Les conditions sont des **processus**
- Condition vraie si et seulement si le processus s'est terminé avec le status 0

```
if  grep $NOM annuaire.txt
then
   echo "$NOM est dans l'annuaire"
fi
```

Le programme « test »

- Fournit les conditions usuelles
  - Existence fichier :

    test -f fichier [-f fichier]

  - égalité de chaine

    test $reponse = 'oui' [$reponse = 'oui']

  - Chaîne vide

    test -z $reponse [-z $reponse]

4 .Structures de contrôle if-then-else

- Exécution conditionnelle
- Syntaxe

```
if condition1
then
    action1
[ elif condition2
    action2 ]
...
[ else
    action n ]
fi
```

Enchaînements de commande

- Enchaînement simple

  date ; ls

- Enchainement si succès

  [$reponse = '42'] && echo bravo

- Enchainement si échec

  [$reponse = '42'] || echo perdu

Boucle for

- Parcourt une liste
- Syntaxe

  for *variable* **in** *liste*

  do

  Actions

  done
- Exemple

  for f in *.cc ; **do** gzip $f ; **done**

Boucle while

- Syntaxe

  while condition

  do

  actions

  done

Lecture fichier

- Une tournure fréquente : while-read
- Exemple

```
while read NOM PRENOM TEL ADRESSE
do
   echo $NOM $ADRESSE
done < annuaire.txt
```

case

- Syntaxe

case mot **in**

motif-a1 **|** *motif-a2* .. **)**

action_A **;;**

motif-b1 **|** *motif-b2* ..)

action_B **;;**

...

esac

exemple

```
case $reponse in
oui | yes | ja )
    reply=yes ;;
no | nein | non )
    reply=no ;;
* )
    reply=invalid ;;
esac
```

case (suite)

```
case $NOMFICHIER in
*.c | *.C )         LANGAGE=C ;;
*.p | *.pas )       LANGAGE=Pascal ;;
*.cc | *.CC )       LANGAGE=C++ ;;
*)                  LANGAGE=inconnu;;
esac
```

5. fonctions

- Permettent une décomposition des tâches
- Admettent des paramètres positionnels
- Syntaxe

```
nom_fonction () {
   actions
}
```

Exemple fonction

```
lister () {
   for F in $*
   do
      case $F in
      *.tgz | *.tar.gz ) OPT=z ;;
      *.tar.bz2 ) OPT=j ;;
      esac
      tar t${OPT}f $F
   done
}
```