Introduction *shell scripting*

Aloÿs Dufour

ATER, LIPN équipe LoCal
Université Paris-Nord XIII

6 janvier 2026

Introduction au script shell

Imaginez un mini-langage de programmation intégré à Linux. Ce n'est pas un langage aussi complet que peuvent l'être le C, le C++ ou le Java par exemple, mais cela permet d'automatiser la plupart de vos tâches. Voici un aperçu de ce qu'on peut faire avec :

- Sauvegarde de vos données
- Surveillance de la charge de votre machine
- Système de gestion personnalisé de vos téléchargements
- ...etc

Introduction au script shell

Pourquoi pas le C ? Le gros avantage des scripts shell, c'est qu'ils sont totalement intégrés à Linux : il n'y a rien à installer et rien à compiler. Et surtout : vous avez trés peu de nouvelles choses à apprendre. En effet, toutes les commandes que l'on utilise dans les scripts shells sont des commandes du système que vous connaissez déjà : `ls`, `cut`, `grep`, `sort`, ... etc.

Introduction au script shell

- Créer un nouveau fichier pour le script : `gedit essai.sh` $\Longrightarrow$ fichier vide
- La première chose à faire dans un script shell est d'indiquer ... quel shell est utilisé :
- Rajouter dans `essai.sh` la ligne `#!/bin/bash`
- le `#!` est appelé le sha-bang
- Après le sha-bang, nous pouvons commencer à coder.
- Le principe : Ecrire les commandes que vous souhaitez exécuter. Ce sont les mêmes que celles que vous tapiez dans l'invite de commandes! Exemple :

```
#!/bin/bash
ls
```

Introduction au script shell

- Donner les droits d'exec au script
  `chmod +x essai.sh`
- Exécuter le script
  - en tapant `./` devant le nom du script : `./essai.sh`
  - en l'appelant à l'aide du shell : `sh essai.sh`

Les variables

- Un nom
- Une valeur
- Exemple `message='Bonjour tout le monde'` Rq :Pas d'espace autour de "="
- `echo` : afficher une variable
  - `echo "Salut tout le monde"`
  - `echo -e "Message\n Autre ligne"`

Les quotes

- Les apostrophes ' '

```
message='Bonjour tout le monde'
echo 'Le message est : $message'
```

la variable n'est pas analysé et le $ est affiché tel quel.

- Les guillemets " "

```
message='Bonjour tout le monde'
echo "Le message est : $message"
```

la variable est analysée et son contenu affiché.

- Les accents graves

```
message=`pwd`
echo "Le message est: $message"
```

les back quotes demandent à bash d'exécuter ce qui se trouve à l'intérieur

READ

- Demander au user de saisir du texte avec la commande `read`.
- La façon la plus simple de l'utiliser est d'indiquer le nom de la variable dans laquelle le message saisi sera stocké :

```
read nom prenom numero
echo "Bonjour $nom $prenom $numero!"
```

- La commande `read` propose plusieurs options intéressantes.
  - -p : afficher un message de prompt
  - -n : limiter le nombre de caractères
  - -s : ne pas afficher le texte saisi

Les arguments d'un script

- Les scripts bash acceptent des paramètres `./varparam.sh param1 param2 param3`
  - `$#` : contient le nombre de param.
  - `$0` : contient le nom du script exécuté
  - `$n` : contient nième param.
- Exp :

```
#!/bin/bash
echo "Vous avez lance $0, il y a $# parametres"
echo "Le parametre 1 est $1"
```

Test

- ```
  if [ test ]
  then
      echo "true"
   else
      echo "false"
  fi
  ```
  ```

Test

```
if [ test ]
then
    echo "premier test a ete verif"
elif [ autre_test ]
    echo "second test a ete verif"
elif [ encore_autre_test ]
    echo "troisieme test a ete verif"
else
    echo "Aucun des tests prec. n'a ete verifie"
fi
```

Test

- 3 types de tests différents en bash :
  1. Tests sur des chaînes de caractères
  2. Tests sur des nombres
  3. Tests sur des fichiers
- Effectuer plusieurs tests à la fois (et : &&, ou : ||) : encadrer chaque condition par des crochets

Tests sur les chaînes

- `$chaine1 = $chaine2` teste si 2 chaînes sont identiques (sensible à la casse...)
- `$chaine1 != $chaine2` teste si 2 chaînes sont $\neq$
- `$chaine` teste si 1 chaîne est vide
- `-n$chaine` teste si 1 chaînes est non vide

```
if [ $1 != $2 ]
then
echo "Les 2 parametres sont differents !"
else
echo "Les 2 parametres sont identiques !"
fi
```

Tests sur les nombres

- `$num1 -eq $num2` teste si les nombres sont égaux.
- `$num1 -ne $num2` teste si les nombres sont diff.
- `$num1 -lt $num2` teste si `num1 < num2`.
- `$num1 -le $num2` teste si `num1 <= num2`
- `$num1 -gt $num2` teste si `num1 > num2`
- `$num1 -ge $num2` teste si `num1 >= num2`

```
#!/bin/bash
if [ $1 -ge 20 ]
then
    echo "Vous avez envoye 20 ou plus"
else
    echo "Vous avez envoye moins de 20"
fi
```

Tests sur les fichiers

- `-e $nomfich` Teste si le fich. existe
- `-d $nomfich` teste si le fich. est un rep.
- `-f $nomfich` teste si le fich. est un... fich. Un vrai fich. pas un dossier.
- `-L $nomfich` teste si fich est un lien symbolique
- `-r $nomfich` teste si fich est lisible (r)
- `$fich1 -nt $fich2` teste si fich1 est plus récent que fich2 (newer than)
- `$fich1 -ot $fich2` (older than)

Les boucles

- `while [ test ]do  echo 'Action en boucle'done`
- Exemple :

```
#!/bin/bash
while [ -z $reponse ] || [ $reponse != 'salade' ]
do
   read -p 'Dites salade : ' reponse
done
```

Les boucles

- ```
  #!/bin/bash
  for etudiant in 'Vincent' 'Thomas' 'Maxime' 'Melody' 'Philippe'
  do
      echo "Moi $etudiant  adore le cours SE"
  done
  ```
- La liste des valeurs n'a pas besoin d'être définie directement dans le code :

  ```
  #!/bin/bash
  liste_fichiers=`ls`
  for fichier in $liste_fichiers
  do
  echo "Fichier trouve : $fichier"
  done
  ```
  ```

Les boucles

- Un cas plus classique du for

```
for i in `seq 1 10`; do echo $i; done
```

- Pour faire des sauts de 2 faire `for i in `seq 1 2 10``