Introduction à l'informatique

Les scripts

D. Buscaldi, J.-C. Dubacq

IUT de Villetaneuse

S1 2016

Les scripts Bash

Introduction
Variables et Paramètres

Rappel

Les interpréteurs

- L'interpréteur parcourt le texte tapé par l'utilisateur, identifie les commandes et les paramètres, et si la syntaxe est correcte, lance un processus.
- Plusieurs interpréteurs existent : csh, tcsh, bash.
- Bash est l'interpréteur du projet GNU. Il est le plus utilisé sous linux. C'est Bash l'interpréteur qu'on utilise dans ce cours.
- L'interpréteur peut lire les commandes à partir d'un fichier, le *script* shell.

Introduction

Structure d'un script Bash

- Un script Bash commence toujours par la ligne # !/bin/bash , suivi par une série d'instructions et commentaires (optionels)
- Un commentaire est une partie rédigée du script qui ne sera pas considérée comme une instruction lors de l'exécution du script. Pour commenter une portion du script on utilise le caractère #. L'ensemble du texte situé sur la même ligne et après le carcactère # sera considéré comme un commentaire et ne sera pas évalué.

Exemple

```
#!/bin/bash
echo Liste des Fichiers:
#affiche la liste
ls
```

Introduction

Execution d'un script

- Un script est un simple fichier texte (habituellement, ils ont l'extension .sh) . Pour l'executer, il faut avant tout le rendre exécutable : `chmod u+x script.sh`
- Maintenant, on peut l'exécuter en faisant : `./script.sh`
- On peut aussi le lancer en appelant explicitement l'interpréteur : `bash script.sh`

Premier script Bash

Q1 Après avoir créé un repertoire nommé `/Intro_Systeme/TP_3/scripts/`, écrivez et exécutez un script `exo_0_script.sh` qui affiche à l'écran le nombre de fichiers contenus dans le repertoire courant, après un message de texte "Nombre de fichiers :"

Exercices

Introduction aux scripts Bash

Q2 Définissez et exécutez un script nommé `exo_1_script.sh` qui réalise la suite de commandes suivante : `echo "Debut"; sleep 2; echo "Apres 2 sec."; sleep 5; echo "Apres 5sec"`

Q3 Que se passe-t-il si vous commentez les lignes commencant par la commande sleep ?

Q4 Définissez un script `exo_2_script.sh` qui affiche "Bonjour", définit le répertoire `/Intro_Systeme/TP_3/scripts/` comme répertoire courant, puis crée dans celui-ci un répertoire `Test`, et finalement copie dans `Test` le fichier `/proc/cpuinfo`.

Q5 Définissez un script nommé `exo_3_script.sh` qui affiche le contenu du répertoire `Test`, puis supprime le fichier `cpuinfo` y contenu (`Test/cpuinfo`), et finalement crée dans `Test` un fichier `infoCPU.txt` composé par les lignes du fichier `/proc/cpuinfo` qui contiennent le mot `'cpu'`.

Les Variables

Les variables en Bash

- Pour affecter une valeur à une variable c'est très simple. Il suffit d'écrire `nom_variable=valeur`
- Pour accéder au contenu d'une variable, il faut utiliser le préfixe $
- On peut accéder aussi aux variables d'environnement, qui ont été définies ailleurs (par exemple `$PATH`)

Exemple

```
MSG=Bonjour
echo $MSG
echo $PATH
```

Les Variables

Q6 Définissez un script nommé `exo_4_script.sh` à partir du script `exo_2_script.sh`, et modifiez-le pour que le nom du répertoire `Test/` soit une variable dans le script.

Les Paramètres

Les paramètres

- Il s'agit d'unes variables spéciales qui contiennent les arguments fournis au script par la ligne de commandes
- **$0** : nom du script
- **$1 $2 ...** : paramètres en position 1, 2, ...
- **$#** : nombre de paramètres positionnels
- **$*** : ensemble des paramètres

Exemple

Soit `arg.sh` le script suivant :

```
#!/bin/bash
echo "Nombre d'argument "$#
echo "Les arguments sont "$*
echo "Le second argument est "$2
```

```
login@host:~$ ./arg.sh A B C
Nombre d'argument 3
Les arguments sont A B C
Le second argument est B
```

Exercices

Introduction aux scripts Bash

Q7 Définissez un script nommé `exo_5_script.sh` à partir du script `exo_2_script.sh`, et modifiez-le pour que le nom du répertoire `Test/` soit passé comme un paramètre du script.

Q8 Rédigez un script recevant 3 paramètres (nom, prénom et serveur) permettant l'affichage d'une adresse mail formatée (nom.prénom@serveur)

Les calculs arithmétiques

`Bash` un langage orienté sur le traitement des chaînes de caractères

Même si ce langage n'est pas fait pour effectuer des opérations de calcul arithmétique il propose des fonctionnalités de base permettant d'effectuer des calculs simples tels que les additions, soustractions, multiplications et divisions.

Syntaxe

`$((` *expression_arithmétique* `))`

Exemples

```
login@host:~$ total=$(( 5 + 3 ))
login@host:~$ echo $total
8
login@host:~$ echo $(( 5 - 3 ))
2
login@host:~$ echo $(( 5 * 3 ))
15
login@host:~$ echo $(( 5 / 3 ))
1
```


Structures de contrôle en BASH

Les calculs arithmétiques
La boucle for
Les branchements conditionnels if

Exercices

Les calculs arithmétiques

Q9 Proposez une suite de 2 commandes affectant à une variable `res` le résultat des opérations arithmétiques suivantes et affichant le résultat contenu dans cette variable : $5 + 7$ et $3 * 2$

Q10 Proposez une suite de 3 commandes permettant :

- d'affecter à une variable `res` la valeur `3`,
- d'ajouter `13` à la variable `res`,
- d'afficher le résultat de l'addition stockée dans la variable `res`.

for

for Boucle itérative

- permet de répéter l'évaluation d'une ou plusieurs instructions,
- à chaque tour de boucle une variable appelée itérateur change de valeur,
- la sortie de boucle s'effectue lorsque l'itérateur atteint une certaine valeurs.

Syntaxe #1

```
for (( init ; test ; incr )) ; do
    expr_1
    expr_2
    ...
done
```

Ici, la condition d'arrêt est sur la valeur numérique de l'itérateur.

Exemple #1

test_for_loop_1.bash

```
#!/bin/bash

echo "test #1"
for (( i = 0; i < 3; i++
));do
   echo '$i = '$i
done
```

```
login@host:~$
./test_for_loop_1.bash
test #1
$i = 0
$i = 1
$i = 2
```


for

for Boucle itérative

- permet de répéter l'évaluation d'une instruction,
- à chaque tour de boucle une variable appelée itérateur change de valeur,
- la sortie de boucle s'effectue lorsque l'itérateur a parcouru toute la liste.

Syntaxe #2

```
for var in val_1 val_2 ... ; do
    expr_1
    expr_2
    ...
done
```

Ici, la boucle s'arrête lorsque toute la liste des valeurs a été parcourue.

Exemple #2

test_for_loop_2.bash

```
#!/bin/bash

echo "test #2"
for i in hello la terre;do
   echo '$i = '$i
done
```

```
login@host:~$
./test_for_loop_2.bash
test #2
$i = hello
$i = la
$i = terre
```


Exercices

La boucle for

Q11 Dans le cours nous avons vu plusieurs syntaxes possibles pour la boucle for. Soit le script suivant :

```
#!/bin/bash
# affiche les 10 premiers entiers pairs
for int in 2 4 6 8 10 12 14 16 18 20
do
echo $int
done
```

Q12 Modifiez ce script pour remplacer la liste de valeurs par une expression arithmétique

if

Branchements conditionnels

- Le `if` permet de mettre en place des alternatives.
- Un test (dont le résultat est Vrai ou Faux) permet de conditionner les expressions qui seront évaluées.

Syntaxe #1

```
if test
then
    expr_1
    expr_2
    ...
fi
```

Comportement

- Ici, les expressions ne sont évaluées que si le test renvoie la valeur Vrai.
- Aucune des expressions ne sont évaluées si le test renvoie la valeur Faux.

if

Syntaxe #2

```
if test
then
        expr_1
else
        expr_2
fi
```

Comportement

- Si le test renvoie la valeur Vrai l'expression `expr_1` est évaluée, et
- sinon le test renvoie la valeur Faux c'est l'expression `expr_2` qui est évaluée.

Syntaxe #3

```
if test_1
then
        expr_1
elif test_2
then
        expr_2
elif test_3
then
        expr_3
else
```

Comportement

- Si `test_1` renvoie la valeur Vrai l'expression `expr_1` est évaluée,
- si `test_2` renvoie la valeur Vrai l'expression `expr_2` est évaluée,
- si `test_3` renvoie la valeur Vrai l'expression `expr_3` est évaluée, et
- si aucun des tests ne renvoie la valeur Vrai alors c'est l'expression

Les tests

Les tests peuvent prendre plusieurs formes

Il peuvent porter sur :

- l'arborescence (présence, absence, permission sur les répertoires et fichiers),
- les chaînes de caractères,
- les valeurs numériques.

Tests de l'arborescence

Syntaxe	Valeur
[-d fichier]	Vrai si fichier est un nom de répertoire valide (si il existe).
[-f fichier]	Vrai si fichier est un nom de fichier valide (si il existe).
[-r fichier]	Vrai si il y a le droit de lecture sur le fichier.
[-w fichier]	Vrai si il y a le droit d'écriture sur le fichier.
[-x fichier]	Vrai si il y a le droit d'exécution sur le fichier.

Exercices

Tests de l'arborescence

Q13 Créez un script `ico_existe.sh`, qui teste si un fichier `ico` est présent dans le répertoire courant. Si le fichier existe, le script affiche le message d'avertissement suivant ($PWD sera remplacé lors de l'exécution par la valeur de la variable d'environnement) :

```
Attention: le fichier $PWD/ico existe
```

Q14 Modifiez le script pour qu'il supprime le fichier `ico` si celui-ci existe et affiche un message d'avertissement indiquant que le fichier est supprimé. Les affichages seront alors les suivants :

```
Attention: le fichier $PWD/ico existe
Le Fichier $PWD/ico est supprime
```

Q15 Modifiez ce script pour qu'il teste en plus si le répertoire courant contient un répertoire nommé `ico/`. Si il ne contient pas de répertoire `ico/`, le script crée ce répertoire.

Les tests

Tests sur les chaînes de caractères

Syntaxe	Valeur
[chaine_1 = chaine_2]	Vrai si les 2 chaînes sont identiques.
[chaine_1 != chaine_2]	Vrai si les 2 chaînes sont différentes.
[-n chaine]	Vrai si la chaîne est non vide.
[-z chaine]	Vrai si la chaîne est vide.

Tests sur les chaînes

Q16 Définissez un script `testPWD.sh` qui prend en paramètre une chaîne de caractères et la compare avec la variable d'environnement `$PWD`, il doit afficher 'OK' si le paramètre correspond à la valeur de la variable, 'Non' en cas contraire.

Les tests

Tests sur les valeurs numériques

Syntaxe	Valeur
[nb_1 -eq nb_2]	Vrai si nb_1 = nb_2 (eq pour equal).
[nb_1 -ne nb_2]	Vrai si nb_1 $\neq$ nb_2 (ne pour not equal).
[nb_1 -gt nb_2]	Vrai si nb_1 $>$ nb_2 (gt pour greater than).
[nb_1 -ge nb_2]	Vrai si nb_1 $\geq$ nb_2 (ge pour greater or equal).
[nb_1 -lt nb_2]	Vrai si nb_1 $<$ nb_2 (ge pour lower than).
[nb_1 -le nb_2]	Vrai si nb_1 $\leq$ nb_2 (ge pour lower or equal).

Les tests

Opérateurs booléens

Syntaxe	Valeur
! [test]	NOT : Vrai si le test renvoie Faux (négation).
[test_1] || [test_2]	OU logique.
[test_1] && [test_2]	ET logique.

Tables de vérité

ET (&&)	Vrai	Faux
Vrai	Vrai	Faux
Faux	Faux	Faux

OU (| |)	Vrai	Faux
Vrai	Vrai	Vrai
Faux	Vrai	Faux

NOT (!)	Vrai	Faux
	Faux	Vra

Exercices

Tests sur les valeurs numériques

Q17 Définissez un script `testTemp.sh` qui prend en paramètre une valeur numérique et une lettre ('C' ou 'F'). Si la lettre choisie est 'C', le script doit afficher 'chaud' si le paramètre numérique est plus grand que `25`, 'froid' si est moins que `10`, 'normal' dans les autres cas. Si la lettre choisie est 'F', il affiche 'chaud' si le paramètre numérique est plus grand que `78` et 'froid' si le paramètre numérique est inférieur à `50`, 'normal' dans les autres cas. Si la lettre n'est pas 'C' ou 'F', il affiche un message d'erreur.

Substitution de commande

Un moyen de composer les instructions

La syntaxe `$(commande avec des arguments)` est remplacée à l'exécution par le résultat de l'exécution dans un sous-shell de la commande `commande avec des arguments`.
Cette fonctionnalité très puissante permet d'utiliser des commandes pour les affecter dans des variables et ensuite s'en servir dans le script.
C'est une substitution

Exemple

```
#!/bin/bash
TITLE="En ce jour du $(date -I)"
MOTS=$(grep cool /usr/share/dict/words)
for i in $MOTS; do
  echo "$TITLE, $i est un mot cool"
done
```

Exercices

Archiveur

Faites un script qui a les actions suivantes si on lui donne en argument un répertoire (par exemple `~/M1101/TD6`) :

Q18 S'arrête si la cible n'est pas un répertoire

Q19 Définit une variable BACKUPDIR qui vaut le chemin du répertoire du dessus suivi du mot `sauvegarde` (ici `~/M1101/sauvegarde`) en utilisant la commande `dirname`

Q20 Crée le répertoire s'il n'existe pas

Q21 Définir une variable faite avec la date du jour et le nom du répertoire (par exemple 2014-10-31-TD6) en utilisante les commandes `basename` et `date`.

Q22 Crée une archive compressée du répertoire (ici en exécutant `tar czf ~/M1101/sauvegarde/2014-10-31-TD6.tgz ~/M1101/TD6`)

On pourra affiner en s'arrêtant si une archive existe déjà sous ce nom avant de la créer (ou proposer de l'effacer en utilisant la commande `read x` pour lire une variable depuis le terminal).