Les scripts Bash

Partie 1 - Les fondations

Sommaire

Au menu :

Définition

C'est quoi un script ?

En général

- Fichier texte
- Contient du code
- Écrit dans un langage donné
- Peut-être exécuté via un interprète
- Pour les scripts shell : interpréteur de commande (CLI)
- L'interpréteur lit ligne par ligne dans un environnement non-compilé

Différence entre script et programme

En général

On distingue en général (via des commandes interprétées par le shell) :

- Programme : indique au processeur ce qu'il doit faire
- Script : indique à un OS, ou une application ce qu'ils doivent faire

Objectif des scripts

Mais pourquoi ?

- Reproduction rapide et simple de suites d'actions répétitives
- Anticipation d’actions futures pour une meilleure réactivité (et être plus serein !)
- Fiabilité : tous les contrôles nécessaires sont fait
- Documentation : un autre admin peut consulter les scripts
- Automatisation : déclenchements automatiques

En bref : simplifier la vie de l'administrateur !

Les shell UNIX

The UNIX way

Le shell standard d'UNIX est **sh**.
Bash (et les autres shell) sont compatibles avec sh.
On peut donc écrire des scripts sh et les faire exécuter par bash (ou un autre).
On a donc 2 approches :
Écrire des scripts standard qui s'exécute sur n'importe quel shell.
Écrire des scripts spécifiques en utilisant les ajouts d'un shell particulier.

Mon premier script

Bonjour le
monde !

1. Créer un fichier coucou
2. Insérer un echo pour afficher : Hello World !
3. Lance le script coucou avec bash

Mon premier script (suite)

Édition de fichier

```
wilder@host:~$ touch coucou
wilder@host:~$ nano coucou

# Ajout de :

echo “Hello World !”

# Enregistrer le fichier et quitter l’éditeur

wilder@host:~$ bash coucou
hello world !

wilder@host:~$ cat coucou
echo hello world !
```

Mon premier script (suite)

Tout en CLI

```
wilder@host:~$ echo echo hello world ! > coucou
wilder@host:~$ bash coucou
hello world !
wilder@host:~$ cat coucou
echo hello world !
```

Bonnes pratiques

Un beau script

- Suffixer les noms de scripts par **.sh** (fréquent mais pas essentiel)
- Accorder les droits d'exécution aux scripts (**chmod u+x**)
- Utiliser les commentaires pour expliquer ses scripts
  - bash ignore les lignes qui débutent par **#**
  - Quelques autres recommandations :
    - Sur le Greg's wiki
    - Sur le wiki Bash hackers

Le shebang

Besoin d'un interprète ?

Sur les systèmes d'exploitation de type Unix

- Convention pour les scripts (fichier texte "exécutable")
- La première ligne : **#! <chemin de l'interpreteur>**

Par exemple pour un script bash sur Debian/Ubuntu :

#!/bin/bash

Pour être plus généraliste : **#!/usr/bin/env bash**

Pour l'histoire du nom, voir WikipediA.

Le code de sortie

Terminer en beauté

Toute commande Unix est censée se terminer en fournissant un code

- Ce code est une valeur numérique entre **0** et **255**
- Indique au processus exécuteur la raison de sa fin
- Un script shell est une "commande composite" il doit donc fournir un code de sortie au shell qui l'a invoqué
- **exit** permet de préciser cette valeur

Quelques exemples de code de sortie

Exemples ?

N° de code de sortie	Signification
0	Sortie normale, tout va bien
1	Erreur générale
2	Mauvaise syntaxe de commande
126	Pas de droit d'exécution sur un fichier existant

Mon premier script dans les règles

Bonjour le monde !

1. Créer un fichier coucou.sh
2. Ajouter le shebang en première ligne
3. Faire afficher "Hello World !"
4. Terminer en renvoyant 0
5. Rendre le script exécutable
6. L'exécuter directement

Mon premier script dans les règles (suite)

Édition de fichier

```
wilder@host:~$ touch coucou.sh
wilder@host:~$ nano coucou.sh

# Ajout de :

#!/bin/bash
echo “Hello World !”
exit 0

# Enregistrer le fichier et quitter l’editeur

wilder@host:~$ chmod u+x coucou.sh
wilder@host:~$ ./coucou.sh
hello world !
```

Mon premier script dans les règles (suite)

Tout en CLI

```
wilder@host:~$ echo '#!/bin/bash' > coucou.sh
wilder@host:~$ echo 'echo “Hello World !”’  >> coucou.sh
wilder@host:~$ echo 'exit 0' >> coucou.sh
wilder@host:~$ chmod u+x coucou.sh
wilder@host:~$ ./coucou.sh
Hello World !
wilder@host:~$ ls -l coucou.sh
-rwxrw-r-- 1 wilder wilder 65 mai   17 12:00 coucou.sh
wilder@host:~$ cat coucou.sh
#!/bin/bash
echo "Hello World !"
exit 0
```

La base

Parser des lignes

Back to basics

Le shell lit un flux de caractères :

- Entrés au clavier en mode interactif
- Lus dans un fichier en mode script

Pour chaque ligne il analyse la ligne caractère par caractère :

- Reconnaître les mots (analyse lexicale)
- Reconnaître les phrases (analyse syntaxique)
- Exécuter la (ou les) commande(s)

Un ligne se termine par le caractère **newline** (obtenu via touche ⏎)

Les métacaractères

Un autre sens

Un métacaractère est un séparateur de mots pour bash :

- Les "blancs" : **espace** et **tabulation** (⇆)
- La fin de ligne : **newline** (⏎)
- Les autres : **| || & && ; ( ) < > # “ ‘**
  - Forment les opérateurs de contrôle
  - Permettent les séquences de commandes, redirections, pipelines...

Le caractère d'échappement : \

- supprime la fonction particulière du caractère suivant

Les métacaractères (suite)

Suite de commandes

```
wilder@host:~$ echo Bonjour ; echo tout le monde ; echo ""
Bonjour
tout le monde

wilder@host:~$ echo Bonjour && echo tout le monde && echo ""
Bonjour
tout le monde

wilder@host:~$ echo -e "Bonjour \ntout le monde\n"
Bonjour
tout le monde

wilder@host:~$
```

Les métacaractères (suite)

Un exemple plus complexe

```
wilder@host:~$ mkdir main\& && mkdir main\&/sub1 \
> main\&/sub2 \
> main\&/sub3
wilder@host:~$ ll ; ll main\&
total 36K
drwxrwxr-x  4 wilder wilder 4,0K mai   18 10:07  ./
drwxr-x--- 28 wilder wilder 4,0K mai   18 09:06  ../
drwxrwxr-x  5 wilder wilder 4,0K mai   18 10:07 'main&'/
total 20K
drwxrwxr-x 5 wilder wilder 4,0K mai   18 10:07 ./
drwxrwxr-x 4 wilder wilder 4,0K mai   18 10:07 ../
drwxrwxr-x 2 wilder wilder 4,0K mai   18 10:07 sub1/
drwxrwxr-x 2 wilder wilder 4,0K mai   18 10:07 sub2/
drwxrwxr-x 2 wilder wilder 4,0K mai   18 10:07 sub3/
```

Les commandes

Anatomie d'une commande

Commande simple :

- Suite de mots séparés par des "blancs"
- Terminée par **newline** ou un opérateur de contrôle
- Premier mot => nom de la commande. Doit correspondre à :
  - une commande interne (cd, exit, umask...)
  - un chemin (avec des /) => emplacement d'un programme
  - un nom de fonction
  - un nom d'exécutable dans un des dossiers de PATH
- Les autres mots sont les **arguments** (vu plus tard) de la commande

Quotes & Double quotes

Quoting

Bash permet d'encapsuler des caractères :

- *Single quotes* (apostrophes) **'**
  - Aucun métacaractère sauf ' => fin de la chaîne
- *Double quotes* (guillemets doubles) **"**
  - Métacaractères : **$** **`** **"** et ****
  - Attention à ne pas confondre ' et `

Quotes & Double quotes (suite)

Suite de commandes

```
wilder@host:~$ nom="Wilder"
wilder@host:~$ echo 'Bonjour $nom'
Bonjour $nom
wilder@host:~$ echo "Bonjour $nom"
Bonjour Wilder
```

Les variables

C'est quoi une variable ?

Définition

Un contenant nommé pour une valeur :

- Nom choisi
- Permet de stocker une valeur => chaque nouvelle écriture d'une valeur remplace la précédente
- Permet de récupérer la dernière valeur stockée autant de fois qu'on le souhaite

Identifiant de variable

Nommer ses variables

Un identifiant de variable :

- Commence par **une lettre** ou un **_**
- Est constitué uniquement de **lettres**, **chiffres** et **_**
- Doit être unique et ne pas être un mot clé du langage
- Est sensible à la casse

Quels sont les noms de variables valides ?

Quizz

✅ someVariable
❌ some-variable
✅ some_variable
✅ variable1
❌ 1variable
❌ my@mail
❌ importantVariable!

Convention de nom

Lisibilité (part 2)

En plus des règles imposées par le langage :

- Donner des noms signifiants (et éviter les abréviations)
- En anglais (dans l'éventualité où vos scripts seront relus)
- En minuscule pour les différencier des variables prédéfinies qui sont en majuscules (**PATH, SHELL, HOME, USER, PS1...**)
- Les shell est sensible à la casse, donc “VARIABLE1” est différent de “variable1”

Convention de nommage pour les scripts bash

Lisibilité (part 2)

En règle générale, on utilise la convention de nommage snake_case :

- Tout en minuscule
- Les mots sont séparés par _

Exemple :

- create_user_account
- function_add_users

Autres conventions de nommage

Lisibilité (part 2)

- camelCase :
  - Lettre majuscule au début de chaque mot sauf le premier
  - Tous les mots sont attachés
  - Ex : createUserAccount
- PascalCase :
  - Comme le camelCase mais chaque mot commence par une majuscule
  - Ex : CreateUserAccount

Utiliser des variables

Comment ça marche ?

- Syntaxe : **nomVariable=valeur** (Attention : pas d'espaces !)
- Par défaut : **string**
- Accède à leur valeur avec **$**
- Détruire une variable : **unset**

```
wilder@host:~$ greetings="Coucou"
wilder@host:~$ echo "$greetings"
Coucou
wilder@host:~$ greetings="Bonjour"
wilder@host:~$ echo "$greetings"
Bonjour
wilder@host:~$ myDirectory="MonDossier"
wilder@host:~$ mkdir $myDirectory
wilder@host:~$ ls | grep "Mon"
MonDossier
wilder@host:~$ unset greetings
wilder@host:~$ echo "$greetings"
```

Un petit exercice

À vous de jouer !

1. Déclarer une variable commande ayant pour valeur 'whoami'
2. Afficher la valeur de la variable
3. Exécuter whoami en utilisant la variable

```
wilder@host:~$ commande='whoami'
wilder@host:~$ echo $commande
whoami
wilder@host:~$ $commande
wilder
```

Arguments et paramètres

Anatomie d'une commande

Les **arguments** d’un script sont des valeurs passées à un script lors de son exécution.

- Ils permettent de donner des données au code
- Ils ne modifient pas le corps du script

Cela permet d'exécuter plusieurs fois un script avec des données différentes.
Dans le script, la valeur des arguments (qui deviennent des **paramètres**) est récupérée dans l’ordre avec les variables **$1**, **$2**, **$3**, etc.

Arguments et paramètres (suite)

Vs

Différence :

Argument (en dehors du script) :
Valeur réelle passée au script lors de son exécution.

Paramètre (à l'intérieur du script) :
Nom utilisé dans le script pour recevoir une valeur.

Arguments et paramètres (suite)

À vous de jouer !

Écrit le script suivant.

Exécute le 2 fois :

- La première fois sans argument
- La seconde avec “Bob” et “Alice”

```
#!/bin/bash

echo "Bonjour $1 !"
echo "Je cherche $2, l’as-tu vu ?"
exit 0

wilder@host:~$ ./script.sh
Bonjour  !
Je cherche , l’as-tu vu ?
wilder@host:~$ ./script.sh Bob Alice
Bonjour Bob !
Je cherche Alice, l’as-tu vu ?
```

Un autre exercice

À vous de jouer !

notify-send sert à envoyer une *desktop notification*

1. Consulter le **man** de **notify-send** ou au moins la syntaxe avec **notify-send --help**
2. Déclarer une variable **notify** ayant pour valeur **notify-send**
3. Afficher la notification de résumé **'Plop'** avec le texte : **'Message envoyé via notify-send'** en utilisant la variable

```
wilder@host:~$ notify='notify-send'
wilder@host:~$ $notify Plop "Message envoyé via $notify"
```

Substitution de commandes

Récupérer le résultat d'une commande

Récupérer le résultat d'une commande au lieu de l'afficher :

- Syntaxe : **$(commande)**

Utilisation :

- Stocker dans une variable
- Utiliser dans une autre commande
- ...

```
wilder@host:~$ id -u
1000
wilder@host:~$ myUID=$(id -u)
wilder@host:~$ echo $myUID
1000
```

Substitution arithmétique

Faire des calculs

Effectuer un calcul

- Syntaxe : **$((<operation>))**

```
wilder@host:~$ echo $(( 12 * 6 ))
72
wilder@host:~$ total=$(( 7 + 3 ))
wilder@host:~$ echo $(( $total * 2 + 1
))
21
```

Variables spéciales

Variables prédéfinies

- **$0** : Nom du script invoqué
- **$#** : Nombre d'arguments du script
- **$*** : les arguments du script en un seul mot
- **$@** : les arguments du script en mots séparés
- **$1** : Le premier argument
- **$2** : Le deuxième argument ...
- **$?** : Le code de sortie de la dernière commande
- **$$** : Le process ID du shell
- **$!** : Le process ID du dernier job en arrière plan

Shell et variables

Une parenthèse

Lorsqu'on exécute un script via un shell

- ce script est exécuté dans un shell fils du shell courant
- ce shell se termine à la fin du script

Les variables déclarées dans un shell ne sont pas accessibles dans les autres (même les shell fils)

Pour exécuter un script dans le shell courant : **source <script.sh>**

Environnement

Une parenthèse

Certaines variables sont dites d'environnement

- Ces variables sont copiées dans chaque shell fils
- Mettre une variable dans l'environnement : **export <variable>**
- **env** permet de récupérer la liste des variables d'environnement

Un petit exercice

À vous de jouer !

Créer un script qui :

- affiche le contenu de la variable **variable**
- met la valeur **‘modified’** dans la variable **variable**
- affiche le contenu de **variable**

Exécuter ce script

Dans un shell, donner à **variable** la valeur **‘initial’**

Ré-exécuter le script

```
#!/bin/bash

echo $variable
variable=modified
echo $variable
exit 0

wilder@host:~$ variable=initial
wilder@host:~$ echo $variable
initial
wilder@host:~$ ./script.sh

modified
wilder@host:~$ echo $variable
initial
```

Suite

À vous de jouer !

Dans un shell :

- donner à **variable** la valeur **initial**
- exporter **variable**
- Ré-exécuter le script
- afficher **variable**

Dans un autre shell distinct :

- afficher **variable**

```
#!/bin/bash

echo $variable
variable=modified
echo $variable
exit 0

wilder@host:~$ variable=initial
wilder@host:~$ export variable
wilder@host:~$ ./script.sh
initial
modified
wilder@host:~$ echo $variable
initial
```

```
wilder@host:~$ echo $variable
```

Et fin

À vous de jouer !

Supprimer le **exit** du script

Dans un nouveau shell :

- donner à **variable** la valeur **initial**
- Exécuter le script avec **source**
- afficher **variable**

```
wilder@host:~$ head -$(($(wc -l < script.sh)-1)) script.sh > scriptnoexit.sh && chmod u+x scriptnoexit.sh
wilder@host:~$ variable=initial
wilder@host:~$ echo $variable
initial
wilder@host:~$ source ./scriptnoexit.sh
initial
modified
wilder@host:~$ echo $variable
modified
```

Synthèse sur les variables

Pour résumer

source <script.sh> :

- Exécution du script dans le shell courant
- La modification des variables dans le script affecte le shell courant => le script agit sur le shell courant

export <variable> :

- La variable est disponible dans l'environnement du shell courant
- Un script exécuté dans ce shell hérite de cette variable => le shell parent prépare les variables pour les shell enfants

Références

La doc officielle
Le wikibooks : Programmation Bash
Le Wiki Bash Hackers
Le Bash Guide de Greg
ExplainShell

Conclusion

Détails complet sur les variables