

Les bases du langage Bash

Un script *Bash* peut automatiser vos tâches répétitives sur Unix. Apprenez à l'utiliser pour améliorer votre flux de travail efficacement.

- [Perfectionnement des systèmes open source - Bash](#)

Perfectionnement des systèmes open source - Bash

author: Dw4rF

2. Basic scripting

2.1. Conventions

- `#! <bin_path>`
-

2. Basic scripting

2.2. Variables d'entrée

- `$$` => PID
 - `$0` => Nom du script
 - `$n` => \$n^{ième} argument
 - `$#` => Nombre d'argument fourni
-

2. Basic scripting

2.3. Arguments

```
while [[ "$1" =~ ^- && ! "$1" == "--" ]]; do case $1 in
-v | --version )
    echo "$version"
    exit
    ;;
-s | --string )
    shift; string=$1
    ;;
-f | --flag )
    flag=1
    ;;
esac; shift; done
if [[ "$1" == '--' ]]; then shift; fi
```

2. Basic scripting

2.4. Conditions

2.4.1. If - Structure (1/2)

```
if [[ $condition1 ]];then
[]..
elif [[ $condition2 ]];then
[]..
else
[]..
```

2. Basic scripting

2.4. Conditions

2.4.1. If - Structure (2/2)

- `[]` => Retourne un booléen
- `!` => Inverse
- `&&` => Et
- `||` => Ou

2. Basic scripting

2.4. Conditions

2.4.2. If - Variable (1/5)

- `-z $strToTest` => Vérifie si la variable est vide
- `-n $strToTest` => Vérifie si la variable n'est pas vide

2. Basic scripting

2.4. Conditions

2.4.2. If - Variable (2/5)

- `$variable1 == $variable2` => `$variable1` égal à `$variable2` (*str*)
 - `$variable1 -eq $variable2` => `$variable1` égal à `$variable2` (*int*) [**e**qual]
 - `$variable1 != $variable2` => `$variable1` pas égal à `$variable2` (*str*)
 - `$variable1 -ne $variable2` => `$variable1` pas égal à `$variable2` (*int*) [**n**ot **e**qual]
-

2. Basic scripting

2.4. Conditions

2.4.2. If - Variable (3/5)

- `$variable1 -lt $variable2` => `$variable1` inférieur à `$variable2` (*int*) [**l**ess **t**han]
 - `$variable1 -le $variable2` => `$variable1` inférieur ou égal à `$variable2` (*int*) [**l**ess than or **e**qual]
-

2. Basic scripting

2.4. Conditions

2.4.2. If - Variable (4/5)

- `$variable1 -gt $variable2` => `$variable1` supérieur à `$variable2` (*int*) [**g**reater **t**han]
 - `$variable1 -ge $variable2` => `$variable1` supérieur ou égal à `$variable2` (*int*) [**g**reater than or **e**qual]
-

2. Basic scripting

2.4. Conditions

2.4.2. If - Variable (5/5)

- `$variable1 =~ ${regex}` => `$variable1` match avec la regex (*str*)
 - `(($variable1 < $variable2))` => Similaire à `$variable1 -lt $variable2` (*int*)
-

2. Basic scripting

2.4. Conditions

2.4.3. If - Fichier (1/4)

- `-e $file` => Vérifie si le fichier **e**xiste
 - `-s $file` => Vérifie si le fichier n'est pas vide
-

2. Basic scripting

2.4. Conditions

2.4.3. If - Fichier (2/4)

- `-f $file` => Vérifie si le nom de fichier fourni correspond à un **f**ichier
- `-d $file` => Vérifie si le nom de fichier fourni correspond à un **d**ossier

- `-h $file` => Vérifie si le nom de fichier fourni correspond à un lien symbolique [**h**yperlink]
-

2. Basic scripting

2.4. Conditions

2.4.3. If - Fichier (3/4)

- `-r $file` => Vérifie si le fichier fourni peut être lu (**r**ead)
 - `-w $file` => Vérifie si le fichier fourni peut être écrit (**w**rite)
 - `-x $file` => Vérifie si le fichier fourni peut être exécuté (**e**xecutable)
-

2. Basic scripting

2.4. Conditions

2.4.3. If - Fichier (4/4)

- `$file1 -nt $file2` => Vérifie si le fichier `$file1` est plus récent que le fichier `$file2` [**n**ewer **t**han]
 - `$file1 -ot $file2` => Vérifie si le fichier `$file1` est plus vieux que le fichier `$file2` [**o**lder **t**han]
 - `$file1 -ef $file2` => Vérifie si le fichier `$file1` est pareil que le fichier `$file2` [**e**qual **f**ile]
-

2. Basic scripting

2.5. Boucles

2.5.1. For (1/2)

```
for <variable> in <condition>;do  
  ...  
done
```

2. Basic scripting

2.5. Boucles

2.5.1. For (2/2)

- `<path/to/folder>` => Pour chaque fichier
 - `{x..y}` => Intervalle `x` à `y`
 - `{x..y..z}` => Intervalle `x` à `y` avec un pas de `z`
-

2. Basic scripting

2.5. Boucles

2.5.2. While (1/2)

```
while <condition>; do  
  ...  
done
```

2. Basic scripting

2.5. Boucles

2.5.2. While (2/2)

Lecture de fichier ligne par ligne:

```
while read -r ligne; do
  ...
done <fichier.txt
```

2. Basic scripting

2.6. Variables

2.6.1. Types - Simple

Type	Déclaration
integer	<code>variable=1</code>
boolean	<code>variable=true</code> ou <code>variable=false</code>
string	<code>variable="value"</code>

Utilisation : `echo $variable # Affiche le contenu de la variable`

2. Basic scripting

2.6. Variables

2.6.2. Types - Tableau (1/2)

Déclaration d'un `array` :

```
variable=('value1' 'value2' 'value3')
```

2. Basic scripting

2.6. Variables

2.6.2. Types - Tableau (2/2)

Utilisation d'un `array` :

```
echo ${variable[0]} # Affiche value1  
echo ${variable[1]} # Affiche value2  
echo ${variable[2]} # Affiche value3
```

2. Basic scripting

2.6. Variables

2.6.3. Types - Liste (1/2)

Déclaration d'un `list` :

```
declare -A variable  
  
variable['key1']="value1"  
variable['key2']="value2"  
variable['key3']="value3"
```

2. Basic scripting

2.6. Variables

2.6.3. Types - Liste (2/2)

Utilisation d'un `list` :

```
echo ${variable['key1']} # Affiche value1  
echo ${variable['key2']} # Affiche value2  
echo ${variable['key3']} # Affiche value3
```

2. Basic scripting

2.6. Variables

2.6.4. Tricks - Variables (1/3)

- `${variable:x:y}` => Renvoie les `y` caractères depuis le caractère `x`

- `${variable%suf}` => Renvoie le contenu de `$variable` sans le suffixe (`suf`) indiqué
 - `${variable#pre}` => Renvoie le contenu de `$variable` sans le préfixe (`pre`) indiqué
-

2. Basic scripting

2.6. Variables

2.6.4. Tricks - Variables (2/3)

- `${variable/toChange/toPut}` => Équivalent `s/toChange/toPut/` de *vim*
 - `${variable//toChange/toPut}` => Équivalent `s/toChange/toPut/g` de *vim*
 - `${variable/#toChange/toPut}` => Équivalent `s/toChange/toPut/` de *vim* **sur le préfixe**
 - `${variable/%toChange/toPut}` => Équivalent `s/toChange/toPut/` de *vim* **sur le suffixe**
-

2. Basic scripting

2.6. Variables

2.6.4. Tricks - Variables (3/3)

- `${variable:-defaultValue}` => Si `$variable` est vide, retourne `defaultValue`
 - `${variable:=defaultValue}` => Si `$variable` est vide, définit `$variable` à `defaultValue`
 - `${variable:+defaultValue}` => Si `$variable` n'est pas vide, retourne `defaultValue`
 - `${variable:?errorMessage}` => Si `$variable` est vide, affiche `errorMessage` et arrête l'exécution
-

2. Basic scripting

2.7. Petit rappel

- `{file1,file2}` => Retourne `file1 file2`
 - `{file1,file2}.php` => Retourne `file1.php file2.php`
 - `{1..3}` => Retourne `1 2 3`
 - `{2..10..2}` => Retourne `2 4 6 8 10`
 - `$(<cmd>)` => Retourne la sortie de commande de `cmd`
-

2. Basic scripting

2.8. Fonctions

Déclaration :

```
maFonction(){  
  ...  
}
```

2. Basic scripting

2.8. Fonctions

Utilisation :

```
maFonction
```

2. Basic scripting

2.9. Fin

- `exit` => Arrêt simple du programme
- `exit 0` => Arrêt du programme en "mode" *sans erreur*
- `exit 1` => Arrêt du programme en "mode" *erreur*
- `exit n` => Arrête du programme en "mode" *erreur* avec code particulier (cf: `/usr/include/syssexits.h`)