Linux et Bash - Script Shell

60 minutes Normal

DataScientest • com

Introduction à Linux

Script Shell

Langage Bash

Bash est un langage installé par défaut sur les machines Linux et permet de piloter la manipulation des fichiers.

Exécuter des scripts Bash

Un script Bash est un simple fichier contenant des lignes de codes écrits en Bash qui peuvent être exécutés. Il contient à sa base, un **shebang**, qui correspond à un indicateur de l'emplacement du shell qui doit être utilisé pour exécuter le code et se présente sous cette forme :

```
1  #!/bin/bash
2
```

C'est quelque chose que vous trouverez dans de nombreux scripts, pas seulement dans les scripts Bash : par exemple, au début d'un script Python, vous pouvez trouver une ligne du type `#!/bin/python3`. Cela permet au fichier de s'exécuter de lui même.

> Ouvrez un fichier nommé `script.sh` avec `nano`. Ajoutez le shebang correspondant à Bash et une autre ligne qui répertorie le contenu du répertoire `/`. Ensuite, quittez l'édition du fichier et enregistrez-le.

Il y a deux façons d'exécuter le fichier :

```
1  bash script.sh
2  ./script.sh
3
```

Dans le premier cas, nous lisons uniquement le contenu du fichier et le transmettons à Bash. Dans le second cas, nous l'exécutons. Par défaut, cela devrait déclencher une erreur car vous n'avez pas le droit d'exécuter ce fichier.

Machine status

Ubuntu Server 18.04 LTS SSD Volume Type 64-bit x86

Online

34.245.135.23

Connect

Reset Stop

Les bases en Bash

Commentaires

Tout d'abord, comme pour tout langage de programmation, vous pouvez commenter des parties de votre code. Pour ce faire, vous pouvez commenter une ligne en utilisant `#` au début de celle-ci.

Définir une variable

Pour définir une variable, nous pouvons utiliser `=`. Par exemple, nous pouvons définir la variable `ma_variable` de la façon suivante :

```
my_variable=hello

```

> Essayez d'utiliser la commande `echo` pour imprimer le contenu de la variable avec :

```
echo my_variable

```

Vous devriez voir que cette commande retourne `ma_variable`. C'est parce que vous devez vous référer à la valeur même de la variable en précédant celle-ci par `$`.

> Essayez la ligne de code suivante pour vous en rendre compte :

```
echo $my_variable

```

Notez que les variables ne sont pas typées et doivent apparaître sous forme de tableaux ou listes de caractères, mais certaines peuvent être utilisées pour l'arithmétique opérations, comparaisons, ...

Guillemets

Si vous souhaitez attribuer une phrase à une variable, vous rencontrerez quelques problèmes :

> Essayez par exemple :

```
my_variable=hello world

```

Nous nous rendons compte que l'espace rompt l'affectation de variable. Vous devez utiliser des guillemets pour correctement définir votre variable :

```
my_variable="hello world"
echo $my_variable

```

Il existe plusieurs types de guillemets : `'`, `"` et `` ` ``.

> Pour comprendre les différences entre les deux premiers, exécutez les lignes suivantes :

```
echo 'the content of my variable is $my_variable'
echo "the content of my variable is $my_variable"

```

Le dernier est utilisé pour affecter le résultat d'une commande Bash à une variable :

Opérations mathématiques

Nous pouvons définir des opérations mathématiques en Bash en utilisant le mot-clé `let` :

```
let "a=1"
let b=2
let "c=b"
let "d = a + b * c"
echo $d

```

Les opérations mathématiques sont assez similaires aux autres langages de programmation : `+`, `-`, `*`, `/`, `**`, ...

Tableaux

Nous pouvons définir des objets similaires à des listes **tableaux** (*array*), pour stocker plusieurs valeurs à la fois :

```
my_array=(hello world)

```

Pour accéder aux différentes valeurs du tableau, nous pouvons utiliser la syntaxe suivante, en tenant compte du fait que l'indexation d'un tableau commence à partir de 0 :

```
echo ${my_array[0]}

```

Pour lui attribuer de nouvelles valeurs, nous pouvons utiliser :

```
my_array[0]=Hi
echo ${my_array[0]}

```

Et pour renvoyer le tableau complet, nous remplaçons simplement l'index par `*` .

```
echo ${my_array[*]}

```

Pour ajouter des éléments à la liste, nous attribuons simplement des valeurs aux indices qui ne sont pas encore attribués dans le tableau :

```
my_array[2]=or
my_array[4]=hello
my_array[1000]=world

echo ${my_array[*]}

```

Notez que l'index n'a pas besoin d'être complet en ce sens que les indices n'ont pas besoin de se suivre.

De plus, les tableaux présentent deux fonctionnalités intéressantes :

- `$ {! my_array [*]}` renvoie les indices des éléments.
- `$ {# my_array [*]}` renvoie le nombre d'éléments.

Boucles et conditions

pouvons utiliser une structure **if-then-fi** :

```
prenom="Daniel"
if [ $prenom = "Daniel" ]
then
echo "Salut Daniel !"
fi

```

Si vous souhaitez ajouter une instruction `else`, il est possible de le faire avec :

```
prenom="Daniel"
if [ $prenom = "Daniel" ]
then
echo "Salut Daniel !"
else
echo "Bonjour" + $prenom +"!"
fi

```

Si nous voulons enchaîner les conditions pour vérifier plusieurs cas :

```
prenom="Diane"
if [ $prenom = "Daniel" ]
then
echo "Salut Daniel ! "
elif [ $prenom = "Diane" ]
then
echo "Salut Diane"
else
echo "Bonjour" + $prenom +"!"
fi
```

Nous pouvons faire appel à de nombreux éléments pour créer une condition :

- `$var1 = $var2` teste l'égalité des tableaux de caractères ;
- `$var1 != $var2` teste l'inégalité des tableaux de caractères ;
- `-z $variable` teste si le tableau de caractères est vide ;
- `-n $variable` teste si le tableau de caractères n'est pas vide ;
- `$var1 -eq $var2` teste l'égalité de valeurs numériques ;
- `$var1 -ne $var2` teste l'inégalité de valeurs numériques ;
- `$var1 -gt $var2` teste `var1 > var2` ;
- `$var1 -lt $var2` teste `var1 < var2` ;
- `$var1 -ge $var2` teste `var1 >= var2` ;
- `$var1 -le $var2` teste `var1 <= var2` .

Pour combiner deux conditions, nous pouvons utiliser `&&` (ET) si nous souhaitons que les deux conditions soient vérifiées et `||` (OU) si nous voulons qu'au moins une des deux conditions le soit :

```
prenom="Diane"
nom="Datascientest"
if [ prenom="Daniel" ] && [ nom="Datascientest" ]
then
echo "Bonjour Daniel Data"
else
echo "Bonjour" + $prenom + $nom
fi

```

While

Pour effectuer une boucle **while** (tant que...), la syntaxe est la suivante :

```
while [ $i -lt 10 ]
do
let "i=i+1"
done
```

For

Pour effectuer une boucle **for** (pour...), la syntaxe est la suivante :

```
for x in '1st iteration' '2nd iteration' '3rd itera
do
echo $x
done

```

Une fonction intéressante pour ces boucles est `seq`. Cette fonction fournit une suite d'entiers d'un entier de départ à un entier de fin comme suit :

Exécutez le code suivant :

```
seq 3 22

```

Fonctions

Il y a deux façons pour définir une fonction :

```
my_function () {
echo "Nous pouvons faire quelque chose ici"
}

```

Ou

```
function my_function {
echo "Nous pouvons faire quelque chose ici"
}

```

Les arguments à passer à la fonction sont donnés par leur numéro. Par exemple une fonction dans laquelle on affiche le premier puis le second argument ressemblera à ceci :

```
function my_function {
echo "Premier argument"
echo $1
echo "Second argument"
echo $2
}

```

Enfin pour appeler ma fonction, il suffit simplement d'écrire :

```
my_function "Daniel" "24"

```

Validated