420-436-SH Développement de scripts

P2 – Scripts Bash (partie 2)

cégep de Sherbrooke

Plan

- Caractères de substitution (*Wildcards*)
- Scripts interactifs
- Codes de sortie de commandes et de scripts
- Redirections
- Pipes

Caractères de substitution (*Wildcards*)

- *Wilcards* utilisés par des commandes comme ls ou find :

Wildcard	Correspondance
*	Zéro caractères ou plus
?	Un caractère
[abcde]	Un caractère (faisant partie de la liste **abcde**)
[a-e]	Un caractère (se trouvant dans la plage spécifiée de **a** jusqu'à **e**)
[!abcde]	Un caractère (ne faisant pas partie de la liste)
[!a-e]	Un caractère (ne se trouvant pas dans la plage spécifiée)
{Québec,Canada}	Une chaîne de caractères (faisant partie de la liste)

- *Wilcards* utilisés pour évaluer des expressions régulières ou par des commandes comme grep :

Wildcard	Correspondance
^	Le début d'une ligne ou texte
$	La fin d'une ligne ou texte
.	N'importe quel caractère
?	L'expression indiquée se trouve **0** fois ou **une** fois
*	L'expression indiquée se trouve **0** fois ou **plusieurs** fois
+	L'expression indiquée se trouve **une** fois ou **plusieurs** fois

Caractères de substitution (*Wildcards*)

- Le *Wilcard* * :

```
ls cv*
ls *txt
ls *Pierre*
ls cv*-*txt
```

- Le *Wilcard* ? :

```
ls ?ocs
ls ???ocs
ls *?ocs
```

Caractères de substitution (*Wildcards*)

- Le *Wilcard* […] :

```
ls *[cx]
ls cv_[d-l]*
```

- Le *Wilcard* [!…] :

```
ls *[!cx]
```
Lister les fichiers qui ne se terminent pas par c ni par x

```
ls [!cv_]*
```
Lister les fichiers qui ne commencent pas par c ni par v ni par _

```
ls !cv_*
```
Lister les fichiers qui ne commencent pas par cv_

Commande grep

- Permet d'effectuer des recherches avancées dans un fichier, dans un texte ou dans le résultat d'une commande. Ex : Soit un fichier texte contenant une liste de pays :

```
grep 'ana' pays.txt → retourne tous les pays contenant la chaîne « ana » dans son nom
```

```
grep '^B' pays.txt → retourne tous les pays dont le nom commence par « B »
```

```
grep 'i$' pays.txt → retourne tous les pays dont le nom se termine par « i »
```

```
grep '\<B.*i\>' pays.txt → retourne tous les pays dont le nom commence par « B » et se termine par « i »
```

```
grep '[CN]' pays.txt → retourne tous les pays contenant la lettre « C » ou la lettre « N »
```

```
grep '^[CN]' pays.txt → retourne tous les pays dont le nom commence par la lettre « C » ou la lettre « N »
```

Expression régulières (RegEx)

- Pour comparer si une variable contient un chiffre entre 0 et 9 :

```
if [[ $var =~ [0-9] ]]; then
    echo "La variable contient au moins un chiffre entre 0 et 9"
else
    echo "oops"
fi
```

- Pour comparer si une variable est un chiffre entre 0 et 9 :

```
if [[ $digit =~ ^[0-9]$ ]]; then
    echo "La variable est un chiffre entre 0 et 9"
else
    echo "oops"
fi
```

- Pour comparer si une variable est une séquence de chiffres :

```
if [[ $digit =~ ^[0-9]+$ ]]; then
    echo " La variable est une séquence de chiffres entre 0 et 9"
else
    echo "oops"
fi
```

Scripts interactifs

- Scripts que demandent des entrées à l'utilisateur

Lecture d'une entrée par clavier

```
1 #!/bin/bash
2
3 clear
4
5 # Affichage d'un message
6 echo -e "***** Exemple de lecture d'entrée par clavier \n"
7
8 # Affichage d'un message qui demande à l'utilisateur d'entrer son nom
9 echo "Veuillez entrer votre nom : "
10
11 # Lecture du clavier et assignation de la lecture à la variable 'name'
12 read name
13
14 # Affichage du contenu de la variable 'name' sur écran
15 echo "Bonjour $name, je te souhaite une belle journée !"
```

La commande *read* permet de lire le clavier et de stocker les données entrée dans la variable '*name*'

Résultat de l'exécution du script

```
***** Exemple de lecture d'entrée par clavier

Veuillez entrer votre nom :
John Doe
Bonjour John Doe, je te souhaite une belle journée !
```

Scripts interactifs

- Vérification du type d'entrée par clavier

```
 9 # Affichage d'un message qui demande à l'utilisateur d'entrer une valeur
10 echo "Veuillez entrer une valeur : "
11
12 # Lecture du clavier et assignation de la lecture à la variable 'input'
13 read input
14
15 if [[ $input =~ ^[+-]?[0-9]+$ ]]; then
16   echo "Input est un entier."
17 elif [[ $input =~ ^[+-]?[0-9]+\.$ ]]; then
18   echo "Input es un texte."
19 elif [[ $input =~ ^[+-]?[0-9]+\.?[0-9]+$ ]]; then
20   echo "Input es un float."
21 else
22   echo "Input est un texte"
23 fi
```

Résultat de l'exécution du script

```
[alexj@localhost scripts]$ ./script_course2.sh
Veuillez entrer une valeur :
23
Input est un entier.
[alexj@localhost scripts]$ ./script_course2.sh
Veuillez entrer une valeur :
3.
Input es un texte.
[alexj@localhost scripts]$ ./script_course2.sh
Veuillez entrer une valeur :
3.1415
Input es un float.
[alexj@localhost scripts]$ ./script_course2.sh
Veuillez entrer une valeur :
3.45rr
Input est un texte
[alexj@localhost scripts]$ ./script_course2.sh
Veuillez entrer une valeur :
r23
Input est un texte
```

Codes de sortie de commandes et de scripts

- Code de sortie
  - Une valeur (un code) retournée qu'indique si la commande ou le script s'est exécuté sans erreurs
  - Le code (la valeur) 0 (zéro) est retourné si l'exécution s'est bien passée.
  - Le code est compris entre 1 et 255 s'il y a eu un problème.
  - La variable $? donne ce code après l'exécution d'une commande ou scripts.
  - Exemple :

```
[alexj@localhost ~]$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[alexj@localhost ~]$ echo $?
0
```

```
[alexj@localhost ~]$ ls musique
ls: cannot access 'musique': No such file or directory
[alexj@localhost ~]$ echo $?
2
```

Codes de sortie de commandes et de scripts

- Script avec codes de sortie

```
 9 # Affichage d'un message qui demande à l'utilisateur d'entrer un chiffre entier
10 echo "Veuillez entrer une un chiffre : "
11
12 # Lecture du clavier et assignation de la lecture à la variable 'input'
13 read input
14
15 if [[ $input =~ ^[+-]?[0-9]+$ ]]; then
16   echo "Vous avez entré $input. Merci !"
17   exit 0
18 else
19   echo "L'entrée n'est pas un chiffre entier"
20   exit 2
21 fi
```

```
[alexj@localhost scripts]$ ./script_code_retour.sh
Veuillez entrer une un chiffre :
12
Vous avez entré 12. Merci !
[alexj@localhost scripts]$ echo $?
0
[alexj@localhost scripts]$ ./script_code_retour.sh
Veuillez entrer une un chiffre :
Hi
L'entrée n'est pas un chiffre entier
[alexj@localhost scripts]$ echo $?
2
```

Flots d'entrée/sortie et redirection

- Exécution de commandes/scripts
  - Les données entrées et produites par les commandes transitent par des canaux spécifiques
  - Canaux standard
    - **stdin (0)** : canal d'entrée standard. L'entrée standard par défaut est le clavier.
    - **stdout (1)** : canal de sortie standard. La sortie standard par défaut est l'écran.
    - **stderr (2)** : canal d'erreur standard. La sortie d'erreur standard par défaut est l'écran.
- Redirection
  - Il est possible d'indiquer à une commande d'utiliser d'autres canaux pour ses entrées / sorties
  - Redirection de la sortie standard

`ls -l` → Le résultat de la commande est envoyé à la sortie standard (l'écran)

`ls -l > fichier1.txt` → Le résultat de la commande est redirigé ver le fichier *fichier1.txt*

`ls -l > fichier1.txt` ↔ `ls -l 1> fichier1.txt`

Flots d'entrée/sortie et redirection

- Redirection de la sortie d'erreur standard (*stderr*)

`ls -l fichier_abc.txt` (Dans cet exemple. le fichier fichier_abc.txt n'existe pas)

Résultat :

```
ls: cannot access fichier_abc.txt: No such file or directory
```

`ls -l fichier_abc.txt 2> log.txt` → Le message d'erreur est redirigé vers le fichier *log.txt*

- Il est possible de combiner plusieurs redirections

```
ls -l fichier_abc.txt 1> resultat.txt 2> log.txt
```

Flots d'entrée/sortie et redirection

- Redirection de l'entrée standard (*stderr*) vers un fichier

`read var1` → Cette commande reçoit une entrée par clavier et la stocke dans la variable *var1*.

`Salut` → L'utilisateur entre le texte « Salut » par clavier

`echo $var1` → Le contenu de la variable var1 est affiché sur écran

Résultat :
`Salut`

`read < fichier2.txt var1` → L'entrée de la commande *read* n'est plus le clavier mais le fichier « fichier2.txt »

`read < fichier2.txt var1` ↔ `read 0< fichier2.txt var1`

Pipes (Tubes)

- Pipe : |
- Permettent de rediriger la sortie d'une commande vers l'entrée d'une autre commande

`ls -l /bin | more` → Affiche le contenu du répertoire « /bin », écran par écran

`ls -l /bin | grep 'sh$'` → Affiche les fichiers dont le nom finit par « sh » dans le répertoire « /bin »

`ls | wc -l` → Affiche le nombre de lignes du résultat de la commande ls

`cat pays.txt | sort` → Affiche les lignes (les pays) du fichier pays.txt, ordonnés en ordre croissant

`cat pays.txt | sort | uniq` → Exemple précédent + élimine les résultats répétés

`ip route | grep default | awk {'print $3'}` → Affiche la passerelle par défaut

Références intéressantes

- Scripts Linux.pdf (Notes de cours)

- Bash Beginners Guide

  https://tldp.org/LDP/Bash-Beginners-Guide/html/chap_01.html

- Shell Scripting Tutorial

  https://www.shellscript.sh/first.html