Script shell

Gorgoumack SAMBE

Université Assane Seck de Ziguinchor

Objectifs

A l'issue de ce chapitre, l'apprenant doit être capable de

- implémenter un script shell et de le lancer
- utiliser des expressions et des variables shell
- utiliser des structures conditionnelles shell
- utiliser des structures itératives shell
- utiliser des fonctions shell

Script shell

- Simple **fichier texte**

Script shell

- Simple **fichier texte**
- Permet d'**automatiser une série d'opérations**.

Script shell

- Simple **fichier texte**
- Permet d'**automatiser une série d'opérations**.
- Constitué de :
  - une ou plusieurs commandes
  - variables et expressions
  - structures de contrôles
  - structures itératives
  - fonctions
  - ...

Mon premier script shell (bash)

- Hello World
  1. saisir ce code dans un editeur ($vim[1] hello.sh):

```
#!/bin/bash
# fichier : bonjour.sh
# Affiche un salut a l'utilisateur
echo "Bonjour"
```

  2. enregistrer le fichier sous le nom hello.sh

[1] lancer vimtutor pour s'initier à vim

Mon premier script shell (bash)

- Hello World
  1. saisir ce code dans un editeur ($vim[1] hello.sh):

```
#!/bin/bash
# fichier : bonjour.sh
# Affiche un salut a l'utilisateur
echo "Bonjour"
```

  2. enregistrer le fichier sous le nom hello.sh

- Executer le script

  $bash hello.sh

[1] lancer vimtutor pour s'initier à vim

Variables

- **Déclaration de variables**
  message=Hello World
  echo $message

[2] offre des options intéressantes à voir

Variables

- **Déclaration de variables**
  message=Hello World
  echo $message

- **Tableaux**
  tableau=('valeur0' 'valeur1' 'valeur2')
  Accès à une valeur : $tableau[2]

[2] offre des options intéressantes à voir

Variables

- **Déclaration de variables**
  message=Hello World
  echo $message

- **Tableaux**
  tableau=('valeur0' 'valeur1' 'valeur2')
  Accès à une valeur : $tableau[2]

- **Lecture de données au clavier : read**[2]
  read -p 'Votre Nom' nom
  echo Hello $nom

[2] offre des options intéressantes à voir

Expression arithmétique

- Syntaxe à crochets ou parenthèses[3] :
  $((expression))ou $[expression]

[3] On peut aussi utiliser la commande **let**

Expression arithmétique

- Syntaxe à crochets ou parenthèses[3] :
  $((expression))ou $[expression]

- Exemple :
  $ echo $((3*5))
  $ 15
  $ x=3
  $ y=5
  $ echo $[x*y]
  $ 15

[3] On peut aussi utiliser la commande **let**

If ... then...else

- ```
  if Condition
  then
      Instructions
  fi
  ```
  ```

If ... then...else

- ```
  if Condition
  then
      Instructions
  fi
  ```

- ```
  if Condition
  then
      Instructions
  else
      Instructions Alternatives
  fi
  ```

If ... then...else

- ```
  if Condition
  then
      Instructions
  fi
  ```

- ```
  if Condition
  then
      Instructions
  else
      Instructions Alternatives
  fi
  ```

- ```
  if Condition
  then
      Instructions
  elif Condition
  then
      Instructions
  else
      Instructions Alternatives
  fi
  ```
  ```

Condition

- Primitive sous la forme [[expression]] ou [expression]

Condition

- Primitive sous la forme [[expression]] ou [expression]
- Opérateurs

Quelques opérateurs			
Numériques		Chaines de caractères	
-eq	égalité	-z	chaine vide
-ne	inegalité	-n	chaine non vide
-lt	inférieur	==	chaines identiques
-le	inférieur ou égal	!=	chaines différentes
-gt	supérieur		
-ge	supérieur ou égal		

Condition

- Primitive sous la forme [[expression]] ou [expression]
- Opérateurs

Quelques opérateurs			
Numériques		Chaines de caractères	
-eq	égalité	-z	chaine vide
-ne	inegalité	-n	chaine non vide
-lt	inférieur	==	chaines identiques
-le	inférieur ou égal	!=	chaines différentes
-gt	supérieur		
-ge	supérieur ou égal		

- Exemple :

```
if [[ x -gt 10 && x -lt 20 ]]
then
    echo "Moyenne"
fi
```

Case

- ```
  case EXPRESSION in
      CASE1)  Commandes;;
      CASE2)  Commandes;;
      ...
      CASEn) Commandes;;
  esac
  ```
  ```

Case

- ```
  case EXPRESSION in
      CASE1)  Commandes;;
      CASE2)  Commandes;;
      ...
      CASEn) Commandes;;
  esac
  ```

- Exemple

```
echo -n "Votre réponse :"
read REPONSE
case $REPONSE in
    O* | o*) REPONSE="OUI" ;;
    N* | n*) REPONSE="NON" ;;
    *)  REPONSE="PEUT-ETRE !";;
esac
```
```

While

- ```
  while CONDITION
  do
      COMMANDES
  done
  ```
  ```

While

- ```
  while CONDITION
  do
      COMMANDES
  done
  ```

- Exemple

  ```
  #!/bin/bash
  while [ $reponse != 'oui' ]
  do
          read -p 'Dites oui : ' reponse
  done
  ```
  ```

until

- ```
  until CONDITION
  do
      COMMANDES
  done
  ```
  ```

until

- ```
  until CONDITION
  do
      COMMANDES
  done
  ```

- Exemple

```
#!/bin/bash
until [ $reponse == 'oui' ]
do
        read -p 'Dites oui : ' reponse
done
```
```

for

- Le shell propose un for différent de celui des autres langages

```
for Variable in LIST
do
    COMMANDES
done
```

for

- Le shell propose un for différent de celui des autres langages

```
for Variable in LIST
do
    COMMANDES
done
```

- Exemple

```
#!/bin/bash
for variable in 'valeur1' 'valeur2' 'valeur3'
do
        echo "La variable vaut $variable"
done
```

for

- Le shell propose un for différent de celui des autres langages

```
for Variable in LIST
do
    COMMANDES
done
```

- Exemple

```
#!/bin/bash
for variable in 'valeur1' 'valeur2' 'valeur3'
do
        echo "La variable vaut $variable"
done
```

- Pour simuler un for classique

```
#!/bin/bash
for i in `seq 1 10`;
do
        echo $i
done
```

break/continue

- Commande break
  Permet de quitter la boucle la plus interne

break/continue

- Commande break
  Permet de quitter la boucle la plus interne

- Commande break n
  n: entier supérieur à 0
  Permet de sortir de n boucles imbriquées.

break/continue

- Commande break
  Permet de quitter la boucle la plus interne

- Commande break n
  n: entier supérieur à 0
  Permet de sortir de n boucles imbriquées.

- Commande continue
  Permet d'interrompre l'itération courante d'une boucle et de passer à l'itération suivante.

Fonctions

- Déclaration

```
function nomFonction {
    instructions ....
}
```

Fonctions

- Déclaration

```
function nomFonction {
    instructions ....
}
```

- ou

```
nomFonction {
    instructions ....
}
```

Fonctions

- Déclaration

```
function nomFonction {
    instructions ....
}
```

- ou

```
nomFonction {
    instructions ....
}
```

- appel de fonction
  nomFonction

Paramètres de script et de fonction

- $0
  nom du script
- $1 à $n
  arguments du script ou de la fonction
- $#
  nombre d'arguments
- $*
  tous les arguments
- Exemple