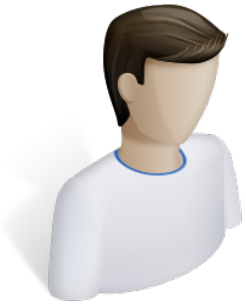
CS230 - Programmation shell sous Unix

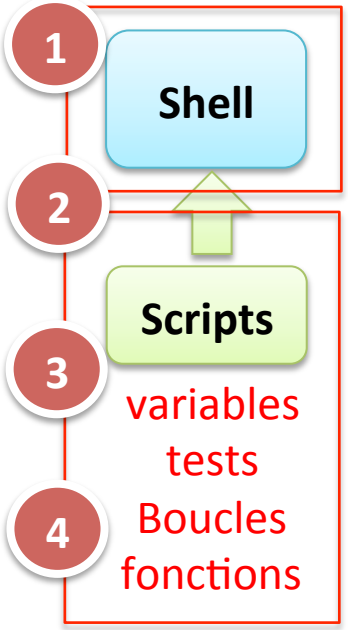
Cours 3 – Les scripts shell – la suite

Dr. Bassem DEBBABI

Objectif du cours 4



Utilisateur



Outils

Binaires exécutables

Appels système

Système de fichiers

Système d'exploitation UNIX

Plan du cours 4

- Manipulation de chaînes de caractères
- Expressions arithmétiques
- Exécution conditionnelle
- Instruction "case"
- Les boucles
- Les fonctions

Plan du cours 4

- Manipulation de chaînes de caractères
- Expressions arithmétiques
- Exécution conditionnelle
- Instruction "case"
- Les boucles
- Les fonctions

Manipulation de chaînes de caractères

- ''

  délimitent une chaîne de caractères. A l'intérieur, tous les métacaractères perdent leur signification.

- " "

  délimitent une chaîne de caractères. A l'intérieur, tous les métacaractères perdent leur signification, à l'exception des métacaractères `, $ et \

- \

  protège le caractère qui suit, que ce soit un caractère normal ou un métacaractère du shell (sauf à l'intérieur d'une chaîne délimitée par des ').

```
nom='Dupont'
echo 'Mr $nom'
echo "Mr $nom"
echo "Mr \"$nom\""
```

- Pour connaître **la taille** (longueur) d'une chaîne de caractères, utilisez la syntaxe suivante :

```
${#ma_variable}
```

  - Exemple

```
nom='Dupont'
echo ${#nom}
#affiche 6
```

Manipulation de chaînes de caractères

- Modificateurs de chaînes
  - Suppression de la plus **courte** sous-chaînes à **gauche**
    - ${paramètre#modèle}
  - Suppression de la plus **longue** sous-chaînes à **gauche**
    - ${paramètre##modèle}
  - Suppression de la plus **courte** sous-chaîne à **droite**
    - ${paramètre%modèle}
  - Suppression de la plus **longue** sous-chaîne à **droite**
    - ${paramètre%%modèle}

Manipulation de chaînes de caractères

- Extraction de sous-chaînes
  - ${variable:n}
    - retourne le contenu de la variable en supprimant les "n" premiers caractères
  - ${variable:n:long}
    - extrait de la variable la chaîne de caractères commençant au "n-ième" caractère et de longueur "long"

Manipulation de chaînes de caractères

- Remplacement de sous-chaînes
  - ${variable/motif/chaîne}
    - permet de faire une substitution dans la "variable" en remplaçant le "motif" par la "chaîne" de caractères

Manipulation de chaînes de caractères

- Exemples
  - var='/home/debbabi' ; echo ${var#*/}
    - home/debbabi
  - var='/home/debbabi' ; echo ${var##*/}
    - debbabi
  - var='/home/debbabi' ; echo ${var%/*}
    - /home
  - var='CS230::13:15::15:00' ; echo ${var%%:*}
    - CS230
  - var='Bonjour tout le monde'; echo ${var:8}
    - tout le monde
  - var='Bonjour tout le monde'; echo ${var:0:7}
    - Bonjour
  - var='Bonjour tout le monde'; echo ${var/Bonjour/Bonsoir}
    - Bonsoir tout le monde

Plan du cours 4

- Manipulation de chaînes de caractères
- Expressions arithmétiques
- Exécution conditionnelle
- Instruction "case"
- Les boucles
- Les fonctions

Expressions arithmétiques

- Les variables dans bash sont par défaut des chaînes de caractères.

```
n=6/3
echo $n
# affiche 6/3
```

- On doit explicitement déclarer une variable de type *entier*

```
declare -i n
n=6/3
echo $n
# affiche 2
```

Expressions arithmétiques

- Utilisation de la **commande externe** *expr*
  - Utilisation de la commande *expr* avec une substitution de commande pour avoir le résultat:

```
z=5
z=$(expr $z + 1)
echo $z
# affiche 6
```

```
z=5
z=$(expr $z+1)
echo $z
# affiche 5+1
```

Expressions arithmétiques

- Utilisation de la **commande interne** *let*
  - Pas besoin d'une substitution de commande
  - Pas besoin de référencer les variables avec le $
  - Pas d'espace entre les expressions

```
let z=5
echo $z
# affiche 5

let z=$z+1
echo $z
# affiche: 6

let z=$z + 1
# -bash: let: +: syntax error: operand expected (error token is "+")

let z=z+1
echo $z
# affiche 7
```

Expressions arithmétiques

- Une autre forme alternative de *let* est d'envelopper l'ensemble de l'expression entre un double parenthèse
  - Cette forme est plus tolérante concernant les espaces

```
((e=5))
echo $e
# affiche 5

(( e = e + 3 ))
echo $e
# affiche: 8

(( e=e+4 ))
echo $e
# affiche 12
```

- On pourra affecter la valeur de l'expression à une variable

```
(( w = 6 + 2 ))
echo $w
# affiche: 8
```

```
w=$(( 6 + 2 ))
echo $w
# affiche: 8
```

Expressions arithmétiques

- La liste des opérations est la suivante :
  - **+ , - , *, /** : addition, soustraction, multiplication, division
  - **%** : reste de la division entière
  - ****** : exponentiel

```
#!/bin/bash
a=2
b=3
c=$(( a**b / 3 ))
echo $c
# affiche 2
```

- Toutes les opérations se font sur des *entiers* seulement

Expressions arithmétiques

- Utilisation de la **commande externe** *bc* (calculatrice)
  - dispose d'un grand nombre de fonctions mathématiques complexes
  - Support les nombres réels

```
r=$( echo "7/3" | bc )
echo $r
# affiche 2
```

```
r=$( echo "scale=2; 7/3" | bc )
echo $r
# affiche 2.33
```

```
r=5.7
t=$( echo "$r + 2.2" | bc )
echo $t
# affiche 7.9
```

Plan du cours 4

- Manipulation de chaînes de caractères
- Expressions arithmétiques
- Exécution conditionnelle
- Instruction "case"
- Les boucles
- Les fonctions

Exécution conditionnelle

- il n'éxiste pas de type Booléen(**True/False** ou encore **0/1**) en Bash, ni d'autres types d'ailleurs, car Bash est un langage non-typé
- Ce sont en fait des 'commandes' qui jouent ce rôle
  - Bash considère qu'une commande s'est bien déroulée lorsqu'elle reçoit comme valeur de sortie '0' (zéro),
  - toute autre valeur correspondant à une exécution non réussie (entièrement ou partiellement).
  - si, dans la structure de commande, la commande qui joue le rôle de condition renvoie '0', alors cela correspond à un True dans le type Booléen, et vice-versa.
- Toute condition est analysée par la commande **test**

```
test expression
# ou
[ expression ]
```

Exécution conditionnelle

Comparaisons de chaînes de caractères

Syntaxe	Description	Exemple
-z *chaine*	vrai si la *chaine* est vide	`[ -z "$VAR" ]`
-n *chaine*	vrai si la longueur de la *chaine* n'est pas nulle	`[ -n "$VAR" ]`
chaine		`[ "$VAR" ]`
chaine1 == *chaine2*	vrai si les deux chaînes sont égales	`[ "$VAR" == "toto" ]`
chaine1 != *chaine2*	vrai si les deux chaînes sont différentes	`[ "$VAR" != "toto" ]`
chaine1 < *chaine2*	vrai si *chaine1* vient avant *chaine2* en basant sur la position de ses caractères sur la table d'ascii	`[ "abcd" \< "abyz" ]`
chaine1 > *chaine2*	l'inverse que celle d'avant	`[ "abcd" \> "abyz" ]`

Exécution conditionnelle

Comparaisons arithmétiques

Syntaxe	Description	Exemple
num1 -eq *num2*	égalité	`[ $nombre -eq 17 ]`
num1 -ne *num2*	inégalité	`[ $nombre -ne 17 ]`
num1 -lt *num2*	inférieur (<)	`[ $nombre -lt 17 ]`
num1 -le *num2*	inférieur ou égal (<=)	`[ $nombre -le 17 ]`
num1 -gt *num2*	supérieur (>)	`[ $nombre -gt 17 ]`
num1 -ge *num2*	supérieur ou égal (>=)	`[ $nombre -ge 17 ]`

Exécution conditionnelle

Examiner le statut d'un fichier

Syntaxe	Description
-c *fichier*	Vrai si le *fichier* existe et est spécial en mode caractère.
-d *fichier*	Vrai si le *fichier* existe et est un répertoire.
-e *fichier*	Vrai si le *fichier* existe.
-f *fichier*	Vrai si le *fichier* existe et est un fichier ordinaire.
-w *fichier*	Vrai si le *fichier* existe et est accessible en écriture.
-x *fichier*	Vrai si le *fichier* existe et est exécutable.
fichier1 -**nt** *fichier2*	Vrai si *fichier1* est plus récent (d'après les dates de modification) que *fichier2*.

Exécution conditionnelle

Combinaisons

Syntaxe	Description	Exemple
! *expression*	Vrai si expression est fausse.	`[ ! $nombre -eq 17 ]`
expr1 -a *expr2*	Vrai si *expr1* et *expr2* sont vrai	`[ $foo -ge 3 -a $foo -lt 10 ]`
expr1 -o *expr2*	Vrai si *expr1* ou *expr2* est vrai	`[ $foo -lt 3 -o $foo -ge 10 ]`
(expression)	vrai si *expression* est vrai.	`[ ($nombre -le 17) ]`

Autres syntaxes pour les conditions

- Double crochets [[expr]]
  - Une version améliorée du syntaxe simple crochets.
  - Utilisation des caractères génériques
    - [["$stringvar" == [sS]tring*]]
  - La séparation des mots est évitée
    - [[$stringvarwithspaces != foo]]
  - Autres combinaisons : "&&" et "||"
    - [[$num -eq 3 && "$stringvar" == foo]]
  - Expression régulières
    - [["$email" =~ "\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}\b"]]

Autres syntaxes pour les conditions

- Double parenthèses ((expr))
  - Seulement pour les comparaisons arithmétiques
    - (($num <= 5))
  - Elle support les opérateurs classiques de comparaison
    - "==", "!=", ">", ">=", "<", "<="
  - Mais aussi des combinaisons logiques
    - "&&", "||"

Exécution conditionnelle

- Plus couramment, il sera intéressant d'utiliser les connecteurs logiques pour lancer des commandes. Ainsi :
  - **cmd_1 && cmd_2** : "cmd_2" ne sera exécuté que si "cmd_1" se termine avec succès.
    - [$foo -ge 3] && echo true
  - **cmd_1 || cmd_2** : "cmd_2" ne sera exécuté que si "cmd_1" se termine par un échec.
  - **cmd_1 | cmd_2** : l'entrée de "cmd_2" est la sortie de "cmd_1"

Exécution conditionnelle (if .. elif .. else .. fi)

- L'instruction if permet d'exécuter des instructions si une condition est vraie. Sa syntaxe est la suivante :

- if [condition] ; then
  action
  fi

- if [condition]
  then
  action1
  else
  action2
  fi

```
#!/bin/bash
nombre_1=25
nombre_2=390
if (( nombre_1 >= nombre_2 )); then
   echo "$nombre_1 est superieur ou egale a $nombre_2"
fi
```

- if [condition1] then action1 elif [condition2] then action2 elif [condition3] then action3 else action4 fi

- Remarque
  - action est une suite de commandes quelconques.

Plan du cours 4

- Manipulation de chaînes de caractères
- Expressions arithmétiques
- Exécution conditionnelle
- Instruction "case"
- Les boucles
- Les fonctions

Instruction "case"

- Choix multiple

```
case mot in
modèle1 ) COMMANDES1 ;;
modèle2 ) COMMANDES2 ;;
esac
```

  - Le modèle peut être construit à l'aide des caractères et expressions génériques de bash.

```
#!/bin/bash
read –p "Entrez votre réponse : " rep
case rep in
   o|O )    echo "OUI" ;;
   *)       echo "Indefini"
esac
```

- Il y a deux modèles dans l'exemple :
  - Soit "o" ou "O"
  - Sinon n'importe quel autre chaine de caractères.

Plan du cours 4

- Manipulation de chaînes de caractères
- Expressions arithmétiques
- Exécution conditionnelle
- Instruction "case"
- Les boucles
- Les fonctions

La boucle *for*

- La boucle for est intéressante quand vous avez une liste d'items à parcourir.
- L'itération for possède plusieurs syntaxes:

  - Première forme :

```
for var
do
  COMMANDES;
done
```

    - "*var*" prend successivement la valeur de chaque paramètre de position initialisé

  - Deuxième forme :

```
for var in liste_mots
do
  COMMANDES;
done
```

    - "*var*" prend successivement la valeur de chaque mot de "*liste_mots*"

La boucle *for*

- Exemples :

```
#!/bin/bash
for i do
   echo $i
   echo "Passage a l'argument suivant ..."
done
```

```
#!/bin/bash

for var in one two three
   echo $var
done
```

```
#!/bin/bash
set $(date)
for i do
   echo $i
done
```

```
#!/bin/bash
valeurs="one two three"
for var in $valeurs
   echo $var
done
```

```
#!/bin/bash
for ((i=1; i<=10; i++))
   echo $i
done
```

```
#!/bin/bash
Fichiers=$(ls *.sh)
for f in $Fichiers do
   echo $f
done
```

La boucle *for*

- La boucle for et les tableaux
  - Rappel :
    - tab=(un deux trois quatre) # déclaration d'un tableau
    - echo ${tab[2]} # affiche trois
    - echo $tab # affiche que le premier élément
    - echo ${tab[*]} # affiche tous les éléments du tableau
    - echo ${#tab[*]} # affiche le nombre d'éléments initialisés

```
#!/bin/bash
tab=(un deux trois quatre)
for t in ${tab[*]}
do
   echo $t
done
```

```
#!/bin/bash
tab=(un deux trois quatre)
long=${#tab[*]}
for ((i=0; i<long; i++))
do
   echo ${tab[$i]}
done
```

La boucle *while*

- La boucle while est intéressante quand vous n'avez pas une liste d'items à parcourir.
- Syntaxe :

```
while suite_cmd1
do
  suite_cmd2
done
```

  - La suite de commandes *suite_cmd1* est exécutée;
    - Si son code de retour est égale à 0 :
      - La suite de commande *suite_cmd2* est exécutée, puis *suite_cmd2* est ré-exécutée.
    - Sinon, l'itération se termine.

La boucle *while*

- Exemples :

```
#!/bin/bash
i=1
while [ $i -lt 5 ]
do
  echo `date`
  sleep 1
  let i=$i+1
done > partitions.txt
```

```
#!/bin/bash
while read myline
do
 if [ "$myline" = "continue" ]; then
   continue;
 elif [  "$myline" = "break" ]; then
   break;
 fi
 echo "vous avez entré : $myline"
done
```

```
#!/bin/bash
a=0
while [ $a -lt 10 ]
do
    echo $a
    if [ $a -eq 5 ]
    then
        break
    fi
    a=`expr $a + 1`
done
```

- continue
  - Passer à l'itération suivante
- break
  - Sortir de la boucle

La boucle *select*

- La commande select affiche un menu qui vous propose le choix parmi une liste définie.
  - Chaque ligne est numérotée en partant de 1
  - Select affiche une phrase qui vous demande d'entrer votre choix. Cette phrase est contenu dans la variable PS3.
  - Une fois que vous avez validé votre réponse, elle est mémorisée dans la variable REPLY
- Syntaxe :

```
select item in list_valeurs_possibles
do
  COMMANDES;
done
```

La boucle *select*

- Exemple :

```
#!/bin/bash
PS3="> selectionnez un plat : " # definie l'invite du menu
echo " -- menu du jour -- " # affiche un titre
select choix in cassoulet pizza "salade du chef" "quitter (q|Q)";
Do
  case $REPLY in
   1) echo "Voici votre $choix."
      echo "Desirez-vous autre chose ?" ;;
   2) echo "Une pizza ? Excellent choix !"
      echo "Desirez-vous autre chose ?";;
   3) echo "Et une $choix, une !"
      echo "Desirez-vous autre chose ?";;
   4|q*|Q*) echo "Au revoir"
      break; ;;
   *) echo "Je n'ai pas compris votre commande. Veuillez repeter svp.";;
  esac
done
```

```
./script.sh
 -- menu du jour --
1) cassoulet
2) pizza
3) salade du chef
4) quitter (q|Q)
> selectionnez un plat : 2
Une pizza ? Excellent choix !
Desirez-vous autre chose ?
> selectionnez un plat : 4
Au revoir
```

Plan du cours 4

- Manipulation de chaînes de caractères
- Expressions arithmétiques
- Exécution conditionnelle
- Instruction "case"
- Les boucles
- Les fonctions

Les fonctions

- Il est souvent utile de définir des fonctions.
- Syntaxe de définition :

```
function nom_fonction { COMMANDES; }
```

```
nom_fonction () { COMMANDES; }
```

- Syntaxe d'appel :

```
nom_fonction
```

```
nom_fonction  param1 param2
```

Les fonctions

- Exemples :

```
#!/bin/bash
ma_function ()
{
    echo "Hello World!"
}

ma_function
```

```
#!/bin/bash
max ()
{
    if [ $1 -gt $2 ]
    then
        echo $1
    else
        echo $2
    fi
}
VAR=$(max 17 11)
```

```
#!/bin/bash
# recursive
factorial ()
{
    if [ "$1" -gt "1" ]; then
        previous=$(($1 – 1))
        parent=$(factorial $previous)
        result=$(($1 * $parent))
        echo $result
    else
        echo 1
    fi
}

factorial $1
```

Fin.