Programmation Shell

Laurent Réveillère

Enseirb-Matmeca
Département Télécommunications

reveillere@enseirb-matmeca.fr
http://uuu.enseirb-matmeca.fr/~reveille/

Shell scripts et fonctions

- Un script
  - fichier de commandes Shell = programme Shell
- Exemples : **`.bash_profile`** et **`.bashrc`**
- Exécution de scripts
  - **`source mon_script`** (déjà vu)
  - **`mon_script`** (à voir)
- En Shell, pas de distinction entre
  - Code exécutable
  - Commande interne du Shell
  - Script Shell

© 2013 L. Réveillère

Exécution d'un script en tant que commande

- Chemin :
  - Script dans un répertoire contenu dans **PATH**
  - Chemin complet du script (par ex. **`./mon_script`**)
- Permission
  - Par défaut

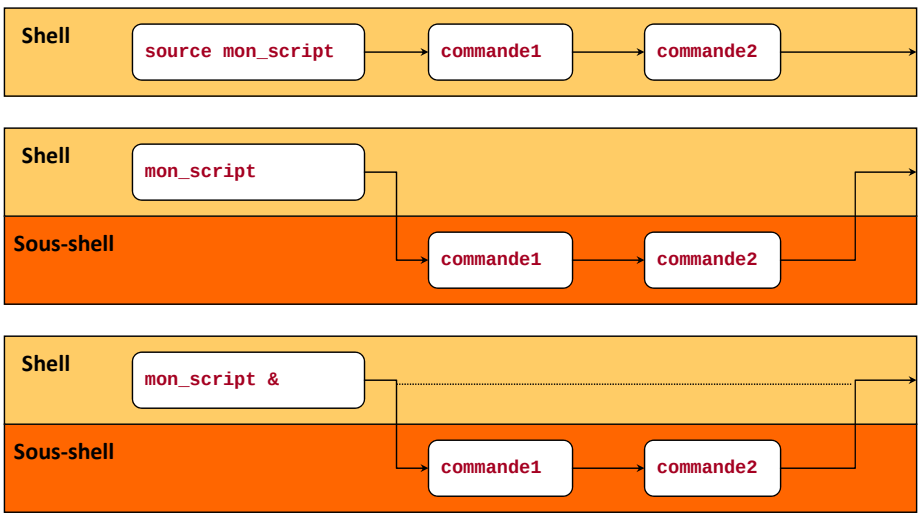
```
$ mon_script
  mon_script: permission denied
```

  - Changement de protection

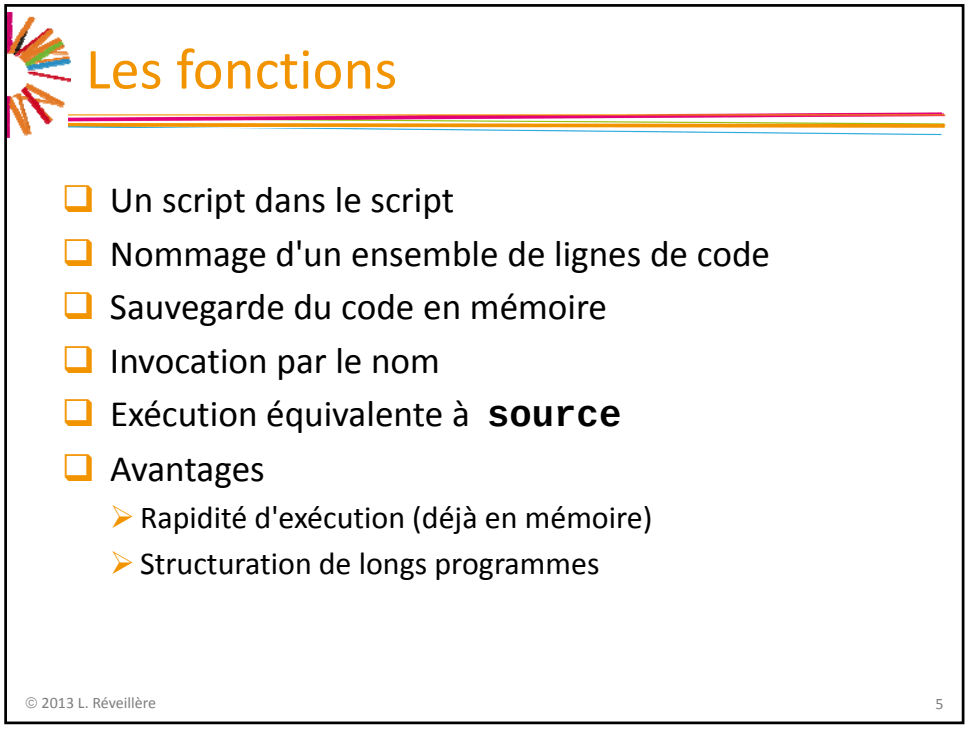
```
$ chmod u+x mon_script
```

© 2013 L. Réveillère

Méthodes d'exécution d'un script



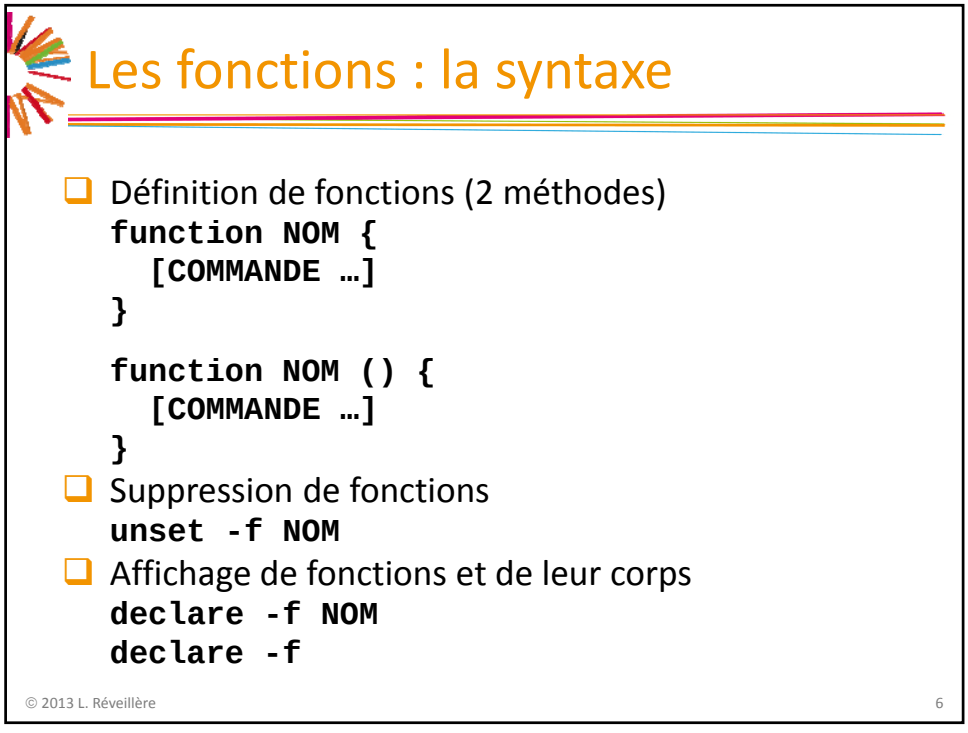
© 2013 L. Réveillère



Les fonctions

- Un script dans le script
- Nommage d'un ensemble de lignes de code
- Sauvegarde du code en mémoire
- Invocation par le nom
- Exécution équivalente à **`source`**
- Avantages
  - Rapidité d'exécution (déjà en mémoire)
  - Structuration de longs programmes

© 2013 L. Réveillère



Les fonctions : la syntaxe

- Définition de fonctions (2 méthodes)

```
function NOM {
  [COMMANDE …]
}

function NOM () {
  [COMMANDE …]
}
```

- Suppression de fonctions

```
unset -f NOM
```

- Affichage de fonctions et de leur corps

```
declare -f NOM
declare -f
```

© 2013 L. Réveillère

Précédence des commandes

- Alias
- Mots-clés (**`function`**, **`if`** …)
- Fonctions
- Commandes internes (**`cd`**, **`type`** …)
- Scripts et programmes exécutables -- ordre défini par **PATH**

© 2013 L. Réveillère

Information sur les commandes

```
$ type -all foo
  foo is aliased to 'echo "foo bar"'
  foo is a function
  foo () {
    echo "foo bar la fonction"
  }
  foo is ./foo

$ type -type function
  keyword
```

© 2013 L. Réveillère

Les variables

- Concept fondamental de Bash
- Nombreux types de variables
- Nombreuses opérations sur les variables

© 2013 L. Réveillère

Paramètres positionnels

- Arguments passés aux scripts
- Nommage des paramètres
  $*i* ⌫ valeur du *i*ème paramètre
- Nom de la commande : **$0**
- Nombre de paramètres : **$#**
- Valeur de tous les paramètres :
  - **$*** : chaîne de caractères unique
  - **$@** : N chaînes de caractères
    - » N nombres de paramètres

  Différence : traitement des espaces passés en paramètre

© 2013 L. Réveillère

Exemple de script

- Script **alice**

```
echo "alice: $@"
echo "$0: $1 $2 $3 $4 $5 $6"
echo "$# arguments"
```

- Execution

```
$ ./alice au pays des merveilles
  alice: au pays des merveilles
  ./alice: au pays des merveilles
  4 arguments
```

© 2013 L. Réveillère

Variables globales et paramètres positionnels

- Script **ma_cmde**

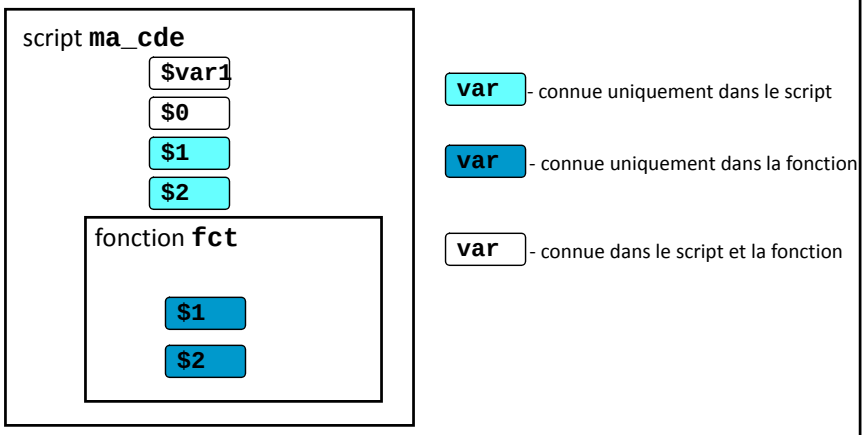
```
function fct {
 echo dans fct: $0 $1 $2
 var1="dans fct"
 echo var1: $var1
}

var1="hors de fct"
echo var1: $var1
echo $0: $1 $2
fct fa1 fa2
echo var1: $var1
echo $0: $1 $2
```

- Exécution

```
$ ma_cmde arg1 arg2
var1: hors de fct
./ma_cde: arg1 arg2
dans fct: ./ma_cde fa1 fa2
var1: dans fct
var1: dans fct
./ma_cde: arg1 arg2
```

© 2013 L. Réveillère

Variables globales et paramètres positionnels



© 2013 L. Réveillère

Variables locales

- Définition de variables locales à une fonction
- Avantages :
  - Plus de structuration
  - Plus d'indépendance
- Limitée à une fonction

© 2013 L. Réveillère

Variables globales et paramètres positionnels

- Script **ma_cde**

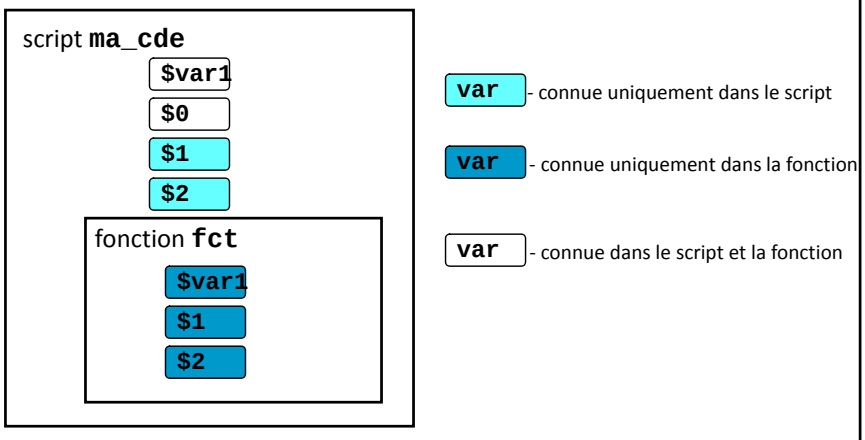
```
function fct {
 local var1
 echo dans fct: $0 $1 $2
 var1="dans fct"
 echo var1: $var1
}
var1="hors de fct"
echo var1: $var1
echo $0: $1 $2
fct fa1 fa2
echo var1: $var1
echo $0: $1 $2
```

- Exécution

```
$ ma_cde arg1 arg2
var1: hors de fct
./ma_cde: arg1 arg2
dans fct: ./ma_cde fa1 fa2
var1: dans fct
var1: hors de fct
./ma_cde: arg1 arg2
```

© 2013 L. Réveillère

Variables globales, variables locales et paramètres positionnels



© 2013 L. Réveillère

Syntaxe des variables

- Raccourci
  `$nom` ⌫ **`${nom}`**
- Concaténation
  `Nom:${USERNAME}` ⌫ **`Nom:alice`**
- Longueur de la valeur d'une variable
  `${#fichier}` ⌫ **`8`**
  si **`fichier`** a pour valeur *albert.c*

© 2013 L. Réveillère

Manipulation de chaînes de caractères

Manipulation de valeurs des variables

- Vérification variable définie et valeur non-nulle
- Affectation de valeurs par défaut
- Gestion des erreurs résultant de variables non initialisées
- Suppression des portions de valeur de variables en fonction d'un modèle

© 2013 L. Réveillère

Opérateurs de substitution de chaînes (1)

- **`${var:-mot}`**

  Si **`var`** existe et n'est pas nulle, retourne sa valeur, sinon retourne **`mot`**.

  - Emploi : valeur par défaut
  - Exemple : **`${compte:-0}`**

- **`${var:=mot}`**

  Si **`var`** existe et n'est pas nulle, retourne sa valeur, sinon lui attribuer celle de **`mot`**, et retourner cette valeur. Ne s'applique pas aux paramètres positionnels et spéciaux.

  - Emploi : valeur par défaut
  - Exemple : **`${compte:=0}`**

© 2013 L. Réveillère

Opérateurs de substitution de chaînes (2)

- **`${var:?message}`**

  Si **`var`** existe et n'est pas nulle, retourne sa valeur, sinon affiche « **`var`**: » suivi de **`message`**, et abandonne la commande ou le script en cours.

  - Emploi : traitement des erreurs de variables non définies
  - Exemple : **`${compte:?"non defini!"}`**

- **`${var:+mot}`**

  Si **`var`** existe et n'est pas nulle, retourne **`mot`**, sinon retourne **`null`**.

  - Emploi : tester l'existence d'une variable
  - Exemple : **`${compte:+1}`**

© 2013 L. Réveillère

Problème

- Fichier contenant le nombre d'albums par artiste

  5 Serge Gainsbourg
  6 Léo Ferré
  1 Jean Yanne
  …

  Ecrire un programme qui affiche les N artistes dont vous possédez le plus de disques. Par défaut, N est égale à 10.
  - **`sort -nr [fichier]`**
    - **`n`** valeur numérique
    - **`r`** tri par ordre décroissant
  - **`head -`***`N`* **`[fichier]`**
    - *N* : premières lignes affichées

© 2013 L. Réveillère

Modèles, comparaisons et correspondance

- Comparaison des portions de la valeur d'une variable avec un modèle
- Modèle composé de caractères génériques
- Applications :
  - Suppression de chemins d'accès
  - Suppression de suffixes de noms de fichier
- Exemple

  `/home/rene/livre/un_grand_fichier`

© 2013 L. Réveillère

Opérateurs de comparaison de modèles (1)

- **`${var#modele}`**

  Si **`modele`** correspond au début de la valeur de la variable, supprime la plus petite partie correspondante et retourne ce qui reste.

  ```
  ${chemin#/*/}
  rene/livre/un_grand_fichier
  ```

- **`${var##modele}`**

  Si **`modele`** correspond au début de la valeur de la variable, supprime la plus longue partie correspondante et retourne ce qui reste.

  ```
  ${chemin##/*/}
  un_grand_fichier
  ```

© 2013 L. Réveillère

Opérateurs de comparaison de modèles (2)

- **`${var%modele}`**

  Si **`modele`** correspond à la fin de la valeur de la variable, supprime la plus petite partie correspondante et retourne ce qui reste.

  ```
  ${chemin%_*}
  /home/rene/livre/un_grand
  ```

- **`${var%%modele}`**

  Si **`modele`** correspond à la fin de la valeur de la variable, supprime la plus longue partie correspondante et retourne ce qui reste.

  ```
  ${chemin%%_*}
  /home/rene/livre/un
  ```

© 2013 L. Réveillère

Problèmes

- Écrire un programme qui change l'extension d'un fichier, c'est-à-dire
  `fichier.c` → `fichier.o`
  - Si le fichier ne se termine pas par **`.c`** ?
- Écrire un programme qui supprime le chemin d'accès à un fichier, c'est-à-dire
  `chemin/fichier` → `fichier`
  - S'il n'y a pas fichier ?

© 2013 L. Réveillère

Substitution de commande

- Substitution d'une commande par sa sortie standard
- Syntaxe
  `$(COMMANDE)`
- Imbrication des substitutions
- Exemples de valeurs
  - **`$(pwd)`** équivalent à **`$PWD`**
  - **`$(ls $HOME)`**
  - **`$(ls $(pwd))`**
  - **`ls -l $(type -path -all cde)`**
  - **`"dans $(pwd) maintenant"`**

© 2013 L. Réveillère

Problème

- Ecrire les fonctions pushd et popd qui mettent en œuvre une pile de répertoires qui permettent de se déplacer temporairement, le Shell mémorisant alors où vous vous trouviez auparavant.

Commande	Contenu de la pile	Répertoire résultant
`pushd doc`	*/home/luc/doc /home/luc*	*/home/luc/doc*
`pushd /etc`	*/etc /home/luc/doc /home/luc*	*/etc*
`popd`	*/home/luc/doc /home/luc*	*/home/luc/doc*
`popd`	*/home/luc*	*/home/luc*
`popd`	Vide (erreur)	*/home/luc*

© 2013 L. Réveillère

Structures de contrôle

Contrôle du déroulement de l'exécution du programme (control flow)

- Exécution de certaines parties du programme
- Répétition, sous certaines conditions, de l'exécution de certaines parties du programme
- **`if`**, **`for`**, **`while`**, …

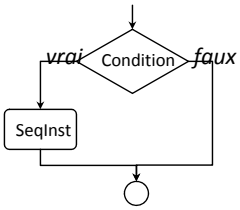
© 2013 L. Réveillère

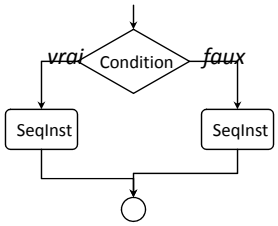
Les conditionnelles

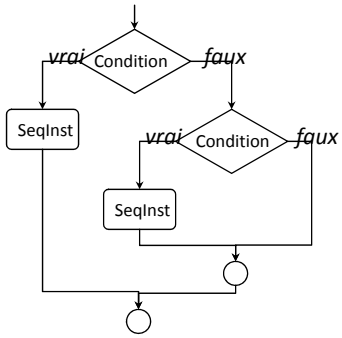
```
if Condition
then
  SeqInst
[elif Condition
   then SeqInst …]
[else
   SeqInst ]
fi
```

Conditions

- Variables Shell
- Caractéristiques de fichiers
- Résultat d'une commande
- etc.







© 2013 L. Réveillère

Statut de retour et de sortie

- Condition = liste d'instructions
  (Pas une expression booléenne comme autres langages)
- Vérification de condition
  - statut de sortie de la commande
- Toute commande Unix = code de retour
  (fonction Shell, script, code C ou autre langage)
- Code/valeur de retour ou statut de fin
  - Nombre entier
    - Valeur 0 → succès/exécution correcte
    - Autre valeur (entre 1 et 255) échec/erreur

© 2013 L. Réveillère

Forme symbolique et exemple

```
si (if) la commande s'est exécutée correctement
alors (then)
   traitement normal
sinon (else)
   gestion de l'erreur
finsi (fi)
```

```
mycd() {
 if cd $1
 then
  echo "Répertoire courant: $1"
 else
  echo "Répertoire erroné: $1"
 fi
}
```

© 2013 L. Réveillère

Gestion de la valeur de sortie : accès

- Variable spéciale « **?** » (valeur **$?**)
- Exemple

```
cd mauvais_repertoire
echo $?
```

- Quelle est la valeur de retour de **mycd** ?

© 2013 L. Réveillère

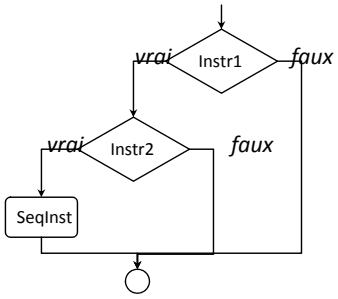
Gestion de la valeur de sortie : création

- Instruction **`return`**
  - Utilisation : fonction (ou script avec **`source`**)
  - Effet : termine la fonction
  - Options :
    - » Argument = valeur de retour
    - » Sans argument = valeur par défaut
      (statut dernière commande)
- Instruction **`exit`**
  - Utilisation : fonction ou script
  - Effet : termine le script
  - Options : *(identique à* **`return`***)*

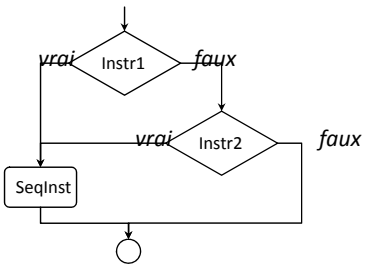
© 2013 L. Réveillère

Combinaison de valeurs de sortie

```
if Instr1 && Instr2
then
  SeqInst
fi
```




```
if Instr1 || Instr2
then
  SeqInst
fi
```



© 2013 L. Réveillère

Négation de la valeur de sortie

```
if ! Instr
then
  SeqInst
fi
```

© 2013 L. Réveillère

Exemples de combinaisons

```
fichier=$1
mot1=$2
mot2=$3
if grep $mot1 $fichier && grep $mot2 $fichier
then
  echo "$mot1  et  $mot2 dans $fichier"
fi
```

ou

© 2013 L. Réveillère

Tests conditionnels

- Autres conditions
  - Comparaisons de chaînes de caractères
  - Tests d'attributs de fichiers
  - Comparaisons de nombres entiers
- Syntaxe :
  `[ condition ]`
- Combinaison :

```
if [ cond1 ] && [ cond2 ]
then
  ...
fi
```

- Valeur de sortie : condition vraie ou fausse
- Commande **`[`** ➔ commande **`test`**

© 2013 L. Réveillère

Comparaisons de chaînes de caractères

Opérateur	Vrai si…
chaîne1 = chaîne2	chaîne1 et chaîne2 identiques
chaîne1 != chaîne2	chaîne1 et chaîne2 différentes
-n chaîne1	chaîne1 non nulle
-z chaîne1	chaîne1 nulle

```
if [ -z $1 ]
then
  echo 'ma_cde: argument inexistant'
fi
```

© 2013 L. Réveillère

Tests d'attributs de fichiers

Opérateur	Vrai si…
-d fichier	fichier existe et est un répertoire
-e fichier	fichier existe
-f fichier	fichier existe et est *ordinaire*
-r fichier	permission de lire
-s fichier	existe et n'est pas vide
-w fichier	permission d'écrire
-x fichier	permission d'exécuter
-O fichier	propriétaire du fichier
-G fichier	groupe propriétaire du fichier
fichier1 –nt fichier2	fichier1 plus récent fichier2
fichier1 –ot fichier2	fichier1 plus ancien fichier2

© 2013 L. Réveillère

Problèmes

- Modifications de
  - **popd** : cas de la pile vide.
  - **pushd** : cas du répertoire destination erroné.
- Ecriture d'une version verbeuse de la commande **`ls`**. Par défaut, liste toutes les informations sur un fichier.

Fonction du nom de **`lsv`**

© 2013 L. Réveillère

Comparaisons de nombres entiers

Opérateur	Comparaison
-lt	inférieur à
-le	inférieur ou égal
-eq	égal
-ge	supérieur ou égal
-gt	supérieur à
-ne	différent de

© 2013 L. Réveillère

La boucle `for`

- Répéter le même code pour plusieurs valeurs
- Liste fixe de valeurs pour la boucle
- Chaque itération prend une valeur
- Syntaxe :

```
for nom [in liste]
do
   instructions avec $nom
done
```

© 2013 L. Réveillère

La liste de valeurs du `for`

- **`liste`** = liste de noms
- Si absence de « **`in liste`** », par défaut **`"$@"`**

```
for arg in "$@"
do
   echo $arg
done
```

© 2013 L. Réveillère

Problème

- Ecrire un programme Shell, nommé **`lf`**, qui reçoit en argument un répertoire et qui n'affiche que les fichiers ordinaires présents dans le répertoire.

© 2013 L. Réveillère

Problèmes

- Vérifier que les chemins de la variable **PATH** sont valides (utilisation de la variable **IFS**)

© 2013 L. Réveillère

Problème

- Ecrire une fonction qui effectue un affichage formaté d'une arborescence

```
progs
    proj1
        f.c
docs
    man1
        command.txt
```

  - Informations
    - **echo -e "\t" truc**
    - **Script tracerep**

```
fichier=$1
echo $fichier
if [ -d $fichier ]; then
  cd $fichier
  tracerep $(ls)
  cd ..
fi
exit
```

© 2013 L. Réveillère

La condition **case**

- Syntaxe :

```
case expr in
  mod1 )
    seqInst1 ;;
  mod2 )
    seqInst2 ;;
  ...
esac
```

- Action (**seqInst**) peut être vide
- Cas par défaut : modèle ‘ * ’
- Combinaison de modèles : **mod1 | mod2**

© 2013 L. Réveillère

Exemple de la condition **case**

```
for fichier in *
do
  case $fichier in
  *.c )
     echo "$fichier: Source C";;
  *.p )
     echo "$fichier: Source Pascal";;
  *.* )
     echo "$fichier: type inconnu"
     echo "Ajouter le cas ${fichier##*.}";;
  * ) ;;
  esac
done
```

© 2013 L. Réveillère

La construction `select`

- Syntaxe :

```
select nom [in liste]
do
  seqInst avec $nom
done
```

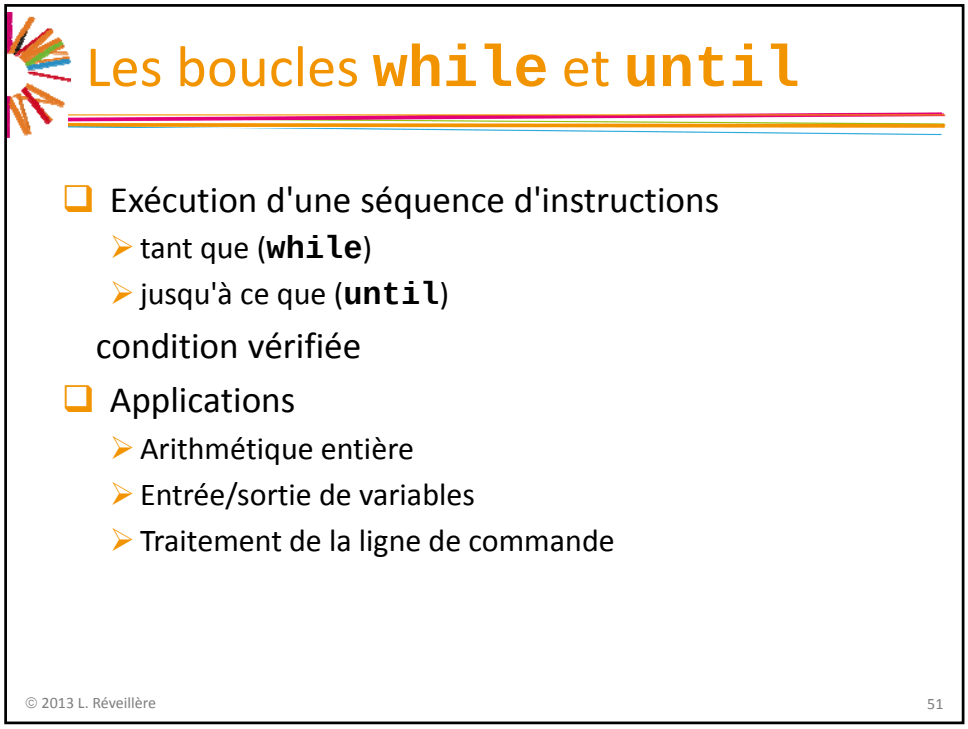
- Effet :
  - Génération d'un menu avec numéros
  - Demande de saisie d'un numéro
  - Enregistrement du choix dans **nom**
  - Exécution des instructions **seqInst**
  - Répétition à l'infini

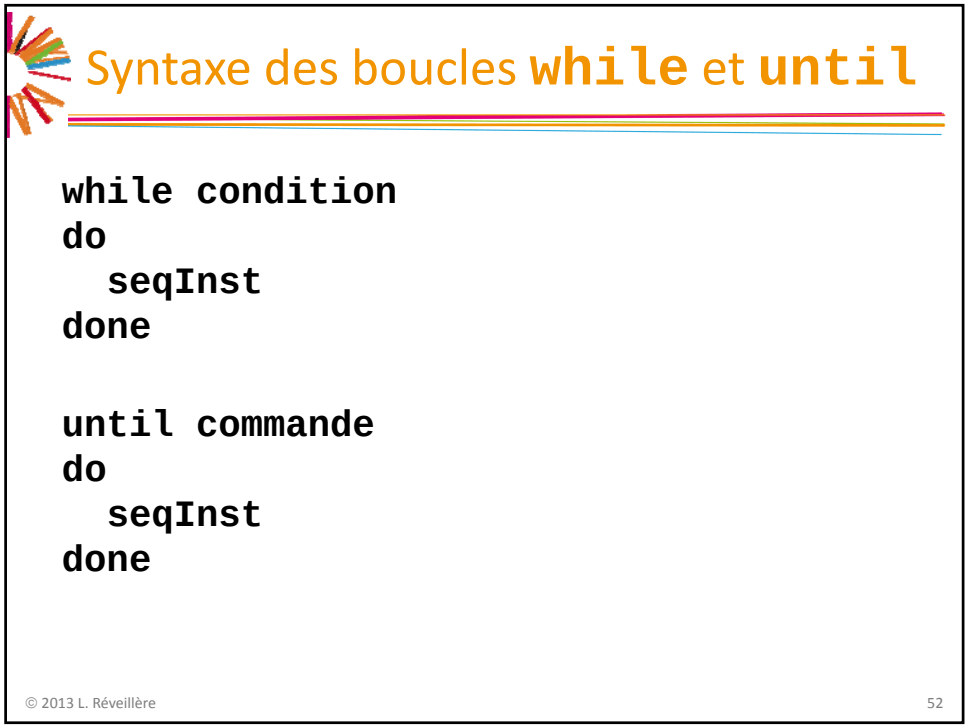
© 2013 L. Réveillère

Exemple de la construction `select`

```
PS3='votre choix ?'
select rep in albert rene luc
do
  if [ -z $rep ]
  then
    echo 'Choix non valide'
  else
    break
  fi
done
```

© 2013 L. Réveillère


Les boucles while et until
Exécution d'une séquence d'instructions
tant que (while)
jusqu'à ce que (until)
condition vérifiée
Applications
Arithmétique entière
Entrée/sortie de variables
Traitement de la ligne de commande
© 2013 L. Réveillère
51


Syntaxe des boucles while et until
while condition
do
seqInst
done
until commande
do
seqInst
done
© 2013 L. Réveillère
52

Exemple de boucle `while`

```
path=${PATH}:
while [ $path ]
do
  ls -ld ${path%%:*}
  path=${path#*:}
done
```

© 2013 L. Réveillère

Exemple de boucle `until`

```
until cp $1 $2
do
  echo -n '.'
  sleep 5
done
```

© 2013 L. Réveillère

Problème

- Ecrire une version modifiée de **cd** qui permet de tester chaque répertoire inclus dans le chemin cible.

```
$ vcd /usr/locol/bin
 vcd: locol: no such directory
```

© 2013 L. Réveillère