Système d'exploitation
Les scripts shell

IUT G.T.R

1ère Année

Sylvain MERCHEZ

PLAN DU COURS

- **L 'exécution**
- Les variables
- Les structures de contrôle

Exécution des commandes

- Enchaînement des commandes (;)
  cd / ; pwd ; ls –al → si une des commandes n'existe pas, l'exécution s'arrête.
- Exécution en arrière plan (&)
  netscape & → l'utilisateur récupère l'interpréteur pour lancer d'autres commandes
- Exécution conditionnelle && ou ||
  cm1 && cm2 → cm2 est exécutée si l'exécution de cm1 a réussi
  cm1 || cm2 → cm2 est exécutée si l'exécution de cm1 a échoué

Exécution d'un programme shell (1)

Commande	Conditions sur le fichier	Effet
bash nom_fichier	Fichier lisible	Interprétation par un sous processus shell
nom_fichier	Fichier exécutable dont la première ligne commence par #! ref_shell	Interprétation par un sous processus ref_shell
source nom_fichier	Fichier lisible	Interprétation dans le shell courant
exec nom_fichier	Fichier lisible et exécutable	Recouvrement du csh courant par un csh interprétant le fichier

Exécution d'un programme shell (2)

Un programme shell est un fichier contenant un ensemble de commandes unix. Cet ensemble est interprétable par le shell.

Exemple : mon_script

#! /bin/bash → interpréteur de commandes
echo "ceci est un script" → instruction à exécuter

Exemples d'exécution :

- bash mon_script argument1 argument2 ….
- source mon_script argument1 argument2 …
- mon_script argument1 argument2 ….

Dans le dernier cas le programme devra :

- commencer par #! Référence d'un shell (ex. #! /bin/bash)
- posséder l'attribut d'exécution (x)

Exécution d'un programme shell (3)

- Récupération des paramètres
  - $0 nom de la commande
  - $n le n-ième paramètre
  - $# le nombre de paramètres
  - $* la liste de tous les paramètres
- Quelques variables spéciales
  - $$ le numéro du processus de la dernière commande
  - $? Status de la dernière commande

Divers

- # commentaires
- (cmde) exécute la commande dans un sous-shell
- read a lecture d 'une entrée pendant l 'exécution
- exit num renvoie le status de la commande
- return num code d 'erreur
- .script exécution du script dans le shell courant
- eval arg interprète arg avant de l 'exécuter
- nom_fonc() {liste_commandes } définition d 'une fonction
- exec arg exécute la commande dans un nouveau shell

PLAN DU COURS

- L 'exécution
- **Les variables**
- Les structures de contrôle

bash : les variables (1)

Le bourne-shell distingue 2 types de variables :

- **les variables locales** : visibles uniquement par le processus qui les a créées. Les variables locales du processus père ne sont pas accessibles par le fils.
- **les variables globales** : les variables du père sont accessibles et modifiables par le processus fils

L'ensemble des variables globales est appelé **environnement** du processus

Par convention :

- Les variables locales sont en minuscules
- Les variables globales sont en majuscules

les variables (2)

- Nom de variable : chaîne de caractères
- Accès au contenu d'une variable ($)
  Exemple : echo $path
- Affectation d'une variable :
  set nomvar = valeur → initialisation d'une variable
- Désallocation d'une variable :
  unset nomvar → désallocation d'une variable
- Listing des variables déclarées :
  set → liste des variables

les variables prédéfinies (1)

- Variables globales
  - HOME : répertoire de connexion de l'utilisateur
  - PATH : liste des répertoires de recherche des commandes
  - PWD : répertoire courant
  - TERM : type de terminal utilisé
  - SHELL : nom du shell utilisé
  - USER : nom de l'utilisateur
  - DISPLAY : localisation de l'affichage

les variables prédéfinies (2)

- Variables locales homonymes
  - status : code de retour de la dernière commande
  - prompt : chaîne de caractères utilisée comme invite
  - home : répertoire de connexion de l'utilisateur
  - path : liste des répertoires de recherche des commandes
  - term : type de terminal utilisé
  - shell : nom du shell utilisé
  - user : nom de l'utilisateur

Toute modification d'une de ces variables entraîne la modification de la variable globale correspondante. Par contre l'inverse n'est pas vrai.

PLAN DU COURS

- L 'exécution
- Les variables
- **Les structures de contrôle**

Interpréteur de commandes : shell

Un shell est un interpréteur de langage de commandes.
Il a 2 rôles :

- Un rôle interactif : l'utilisateur peut exécuter directement des commandes
- Un rôle de langage de programmation : le shell exécute les commandes contenues dans un fichier

Sous unix, il existe plusieurs shells :

- Le C-shell (csh) ou turbo C-shell (tcsh)
- Le Korn-shell (ksh)
- Le Bourne-shell (sh) ou le Bourne-Again-Shell (bash)

La liste des shells disponibles se trouve dans /etc/shells

Utilisation des divers guillemets

- ' : une chaîne de caractères mise entre simple quotes ne sera pas interprétée par le shell
- " : seuls les métacaractères ($, ` et \) sont interprétés, les autres seront pas interprétés.
- ` : un texte entre quotes inversés est considéré comme une commande à exécuter.

Exemples :

```
echo ' ceci est une chaine de caractères non interprétée'
echo `pwd`
echo " Le répertoire courant de l'utilisateur $user est `pwd` "
```


Les caractères spéciaux

- \ banalise le caractère suivant
- « … » banalise les caractères sauf \, $, et `
- ' … ' banalise tous les caractères
- ` … ` substitution de commande
- * n 'importe quelle chaîne de caractères
- ? n 'importe quel caractère
- [...] n 'importe quel caractère décrit entre les crochets

Les instructions (1) : commentaire

- Toute chaîne précédée du caractère # est un commentaire.

  Exemple :

  #Ceci est un commentaire

Les instructions (2) : les expressions

- Syntaxe : **expr**

expr oper1 op oper2	exécute les opérations arithmétiques sur les opérandes oper1 et oper2 (+, -, *, /, %, =, \>, \>=, \<, \<=, !=)
expr exp1 | exp2	renvoie *exp1* si l 'expression est non nulle, *exp2* sinon
expr exp1 \& exp2	renvoie *exp1* si exp1 et exp2 sont non nuls, 0 sinon
expr exp1 : exp2	renvoie le nombre de caractères en commun dans les deux arguments
expr length exp	retourne le nombre de caractères de *exp*
expr substr exp n1 n2	retourne la chaîne de caractères *exp* où les caractères *n1* ont été remplacés par *n2*
expr index exp car	retourne la position du caractère car dans la chaîne *exp*

La commande test

La commande permet de valider une condition

- Syntaxe
  - **test express** ou **[express]**

-r fichier	vrai si l'utilisateur a le droit de lecture sur *fichier*
-w fichier	vrai si l 'utilisateur a le droit d 'écriture sur *fichier*
-x fichier	vrai si l 'utilisateur a le droit d 'exécution sur *fichier*
-e fichier	vrai si *fichier* existe
-f fichier	vrai si *fichier* est un fichier régulier
-d fichier	vrai si *fichier* est un répertoire
-s fichier	vrai si la taille de *fichier* est non nulle
-o fichier	vrai si l'utilisateur est propriétaire du *fichier*

La commande test (2)

- **test express** ou **[express]**

S1 = s2	vrai si les deux expressions sont égales
s1 != s2	vrai si les deux expressions sont différentes
s1	vrai si s1 n 'est pas la chaîne nulle
e1 -eq e2	vrai si les deux entiers e1 et e2 sont égaux (autres comparaisons: -ne, -gt, -ge, -lt, -le)
!	Négation unaire
-a	opération binaire ET
-o	opération binaire OU

La commande test (2)

Exemples :

test $val -le 20 -a $val -ge 0

→ la variable val est comprise entre 0 et 20

test "$chaine" != ""

→ la variable chaîne est différente de vide

Remarque : le caractère espace est nécessaire avant et après un des opérateurs ci-dessus

L'instruction conditionnelle (1) : IF

- La plus traditionnelle :

```
if condition
then
   instructions
else
   instructions
fi
```

ou

```
if condition
then
      instructions
fi
```

Exemple :

```
if find / -name main*.c; then
      echo « fonction main trouvée »
else
      echo « fonction main non trouvée »
fi
```


L'instruction conditionnelle (2) : IF

- If en cascade : un seul fi sur la totalité de la structure.

```
if condition
then
   instructions
elif condition
 then
   instructions
...
else
   instructions
fi
```

Remarque : le shell est un langage interprété, il est nécessaire de respecter scrupuleusement la syntaxe (par exemple l 'alignement du mot then)

Les instructions itératives (1) : WHILE

- Syntaxe :

```
while condition
do
      instructions

done
```

Exemple

```
while [ $n -lt 10 ]
do
      echo $n
      n=`expr $n + 1`
done
```

- Les instructions sont exécutées tant que la condition est vraie.
- Remarque : une expression de valeur nulle est considérée comme fausse, tout autre valeur est considérée comme vraie

Les instructions itératives (2) : UNTIL

- Syntaxe :

```
until condition
do
      instructions

done
```

Exemple

```
until [ $n -gt 10 ]
do
      echo $n
      n=`expr $n + 1`
done
```


Les instructions itératives (2) : FOR

- Syntaxe :

```
for param in liste
do
      instructions

done
```

Exemple

```
for i in `ls`
do
      cp $i /dir/$i
      echo « $i copie »
done
```

La variable para prend successivement les valeurs de la liste
Si la liste est omise, para prend alors les valeurs passées en paramètres

L'instruction de choix multiple : case

- Syntaxe

```
case para in
  choix1   )  instructions ;;
  choix2 | choix3 ) instructions ;;
esac
```

- Exemple

```
case $1 in
  -d )   rmdir $dir; exit;;
  -o )   echo « option-o »; exit;;
  * )    echo « reponse incorrecte »;;
esac
```