

## TD5 : Description du langage shell

Le shell est aussi un langage de programmation qui permet d'écrire de nouvelles commandes appelées **scripts** (commande personnalisées, administration système). Les fichiers de commandes ainsi créés contiennent des commandes UNIX et des structures de contrôle définies par le shell (if, for, while etc...).

### Description du langage shell

- \* Les fichiers de commandes doivent commencer par la ligne:

*#!/bin/sh*

qui indique que les commandes qui suivent doivent être interprétées par le shell sh.

- \* les lignes de commentaires doivent commencer par le caractère #

- \* on met une commande par ligne

- \* le nom de fichier de commande sera le nom de la commande à taper pour exécuter le script.

- \* les scripts sont rendus exécutable par la commande chmod

*ex: chmod u+x script* rend le script exécutable pour le possesseur du fichier

- \* on peut définir des variables. Les variables sont initialisées avec égal: var = bonjour

- \* on accède aux variables en les faisant précéder du caractère \$: echo \$var

- \* par définition, les variables \$1, \$2, ... \$9 correspondent aux arguments donnés avec la commande. La variable \$0 désigne la ligne de commande telle qu'elle a été tapée. La variable \$\* contient la liste des arguments séparés par des blancs. La variable \$# contient le nombre d'arguments donnés.

- \* récupération du résultat d'une commande dans une variable: on met la commande entre back-quotes:

*ex: var = `date`*

### Instructions utilisables dans un script

Toutes les commandes UNIX sont utilisables

- \* **echo**: permet d'afficher un texte sur la sortie standard

*ex: MESSAGE=bonjour*

*echo "salut a tous"*

*echo \$MESSAGE*

- \* **read**: permet de lire un texte sur l'entrée standard

*ex: echo "donnez votre nom "*

*read reponse*

*echo \$reponse*

- \* **La conditionnelle: if ... then... else... fi**

*ex: if [ \$langue = "français" ]*

*then echo bonjour*

*else echo hello!*

*fi*

Le test derrière le if doit être entre [] et il doit y avoir des blancs entre chaque constituants du test.

Les opérateurs sont nombreux :

opérateurs booléens: ! (négation), -a (et), -o (ou), parenthésage (),

opérateurs sur les entiers: -eq (=), -ne (différent), -gt (>), -ge, -lt, -le

prédicats sur les fichiers:

-r : le fichier existe en lecture,

-w : le fichier existe en écriture

-f : l'argument existe et est un fichier,

-d : l'argument existe et est un répertoire,

-s : existe et est de taille non nulle

opérateur sur les chaînes de caractères: =, != (différent), -n (chaîne non vide)

### Exemple d'utilisation:

```
if [ $# -ne 2 ]  
then echo "erreur il faut deux paramètres"  
fi
```

### \* La boucle: for ... in ... do ... done

*ex: for fichier in f1.p f2.p f3.p*

*do*

*echo je compile \$fichier*

*pc \$fichier*

*done*

on peut mettre \* derrière le **in**, la variable prend alors la liste des fichiers du répertoire courant.

*ex: for fich in \**

*do*

*chown giron \$fich*

*done*

*for V in \$\* : V prend alors ses valeurs dans la liste des arguments passés au script.*

### Exemple 1

```
#!/bin/csh -f  
foreach i ($*)  
    echo "Cherche le mot unix dans $i"  
    grep -i unix $i  
end
```

### Exemple 2

```
#!/bin/csh -f  
if (-e $1) then  
    cat $1  
else  
    echo "$1 n'existe pas"  
endif
```

### Exemple 3

```
#!/bin/csh -f  
@n = 5  
while ($n)  
    echo "Hello !"  
    @n--  
end
```

### Exemple 4

```
#!/bin/csh -f  
switch ($#argv)  
case 0 :  
    pwd;  
    breaksw  
case 1 :  
    if (-f $argv[1]) then  
        cat $argv[1]  
    else if (-d $argv[1]) then  
        ls -l $argv[1]  
    else echo "Erreur de parametre"  
    endif  
endif  
breaksw  
default :  
    echo "Trop de parametres"  
endsw
```