

# Programmation shell

- ensemble de commandes dans un fichier ayant éventuellement des paramètres
- nom de la commande = nom du fichier = `script`  
`shell`
- paramètres repérés par leur position quand on appelle la commande
- les commandes sont regroupées par une syntaxe de *langage de commandes*
- plusieurs langages → plusieurs syntaxes regroupées en 2 familles
  - les Bourne Shells (`sh`, `ksh`, `bash`)
  - les C-Shells (`csh`, `tcsh`)

# Ordre des opérations

1. Ouvrir un fichier texte et mettre en **1<sup>ere</sup> ligne** le shell choisi

```
#!/usr/bin/sh
```

2. Ecrire la commande
3. La rendre exécutable :  
`chmod a+x` *votre commande*

# Exemple

```
$cat cmd1.sh
#!/usr/bin/sh
date
echo "Catalogue accueil      =" $HOME
echo "Catalogue de travail  =" \c" ; pwd
echo "Nombre de fichiers    =" " ; ls | wc -l

$chmod a+x cmd1.sh

$cmd1.sh
```

# Variables utilisateurs

- On peut créer des variables  $v$  par affectation et les utiliser avec  $\$ v$
- Manipulation des expressions arithmétiques avec  $expr$

```
#!/usr/bin/sh
C="abc"
N=12
echo "variable C = " ${C}
echo "variable N = " ${N}
echo "variable N+1 = \c" ; expr ${N} + 1
N=`expr ${N} + 10`
echo "variable N apres N=N+10= " ${N}
```

# Variables prédéfinies

- `$*`: liste des paramètres
- `$#`: nb de paramètres  
(sans le nom de la commande)
- `$$`: n<sup>o</sup> processus en cours
- `$!`: n<sup>o</sup> du dernier processus en background
- `$?`: code retour de la dernière commande exécutée

# Exemples

```
$cat cmd2.sh
```

```
#!/usr/bin/sh
```

```
echo "Nombre de parametres = " $#
```

```
echo "Parametre numero 0 = " $0
```

```
echo "Parametre numero 1 (nom du catalogue) = " $1
```

```
echo
```

```
echo "No du processus en cours ($0) = $$"
```

```
pwd &
```

```
echo "No du dernier processus en background (pwd)= $!"
```

```
echo
```

```
date
```

```
echo "Nombre de fichiers dans le catalogue $1 = " ; ls $1
```

# Exemples

```
$cat cmd3.sh
```

```
#!/usr/bin/sh
```

```
ls
```

```
echo "exemple code de retour OK (ls)= $? "
```

```
ls toto
```

```
echo "exemple code de retour KO = $? "
```

# Guillemets, apostrophes and co

- `` exp `` : *exp* est évaluée dans un **sous-shell** (ou processus fils)
- `' exp '` : protège *exp* d'une évaluation
- ``` exp1 exp2 ``` : protège *exp1 exp2* d'une évaluation et les considère comme un seul argument

# Exemples

- COMMANDE ``echo a b`` → 2 arguments: a et b
- COMMANDE `"`echo a b`"` → 1 argument : "a b"
- COMMANDE ``echo`` → pas d'argument
- COMMANDE `"`echo`"` → un argument vide

# La commande test

Syntaxe `test exp` ou `[exp]` ;

Elle permet :

- de réaliser des tests
- de restituer le résultat sous la forme d'un code retour (=0 si OK, ≠0 sinon)

Quelques exemples de `exp` :

- `-r fichier` : vrai si on peut lire dans fichier
- `-x fichier` : vrai si on peut l'exécuter
- `-d fichier` : vrai si répertoire
- `-n c1` : vrai chaîne non nulle
- `c1 = c2` : vrai si  $c1 = c2$
- `c1 != c2` : vrai si  $c1 \neq c2$

# Entrée standard

Par la commande

```
read variable
```

Exemples :

```
#!/usr/bin/sh
```

```
read v
```

```
echo $v
```

# Commandes `exec` et `eval`

En `sh`

- `exec commande`

- `eval commande`

**exec** : *commande* s'exécute en remplaçant le processus courant

**eval** : *commande* est interprétée puis le résultat de l'interprétation est exécuté

# Exemple exec

```
#!/usr/bin/sh
```

```
read cmd
```

```
exec $cmd
```

```
# Tout ce qui suit ne sera pas exécuté
```

```
# car le code de $cmd a remplacé celui ci
```

```
echo ligne 1
```

```
echo ligne 2
```

```
echo ligne 3
```

# Exemple eval

```
#!/usr/bin/sh

read cmd
# supposons que l'utilisateur saisisse "cmd1.sh"

# Exemple
# 1) interpretation de "file $cmd" --> file cmd1.sh
# 2) execution de file cmd1.sh
eval "file $cmd"

# Ce qui suit est exécuté
echo ligne 1
echo ligne 2
echo ligne 3
```

# Sélection

```
if ...then ...else ...fi
```

Syntaxe :

```
if <liste de commandes 1> then
    <liste de commandes 2>
else
    <liste de commandes 3>
fi
```

# Exemples

```
$cat cmd6.sh
```

```
#!/usr/bin/sh
```

```
# if ( test $# = 0 ) then ou
```

```
if [ $# = 0 ]; then
```

```
    echo "Il n'y a pas de parametres"
```

```
else
```

```
    echo "il y a $# parametres"
```

```
fi
```

# Aiguillages

```
case ...in ...esac
```

Syntaxe :

```
case <chaine> in
    <motif> ) <liste de commandes> ;;
    <motif> ) <liste de commandes> ;;
    . . . . .
    . . . . .
    . . . . .
esac
```

# Exemples

```
$cat cmd7.sh
#!/usr/bin/sh
case $# in
    0) pwd ;;
    1) if( test -f $1 ) then
        cat $1
        else
        echo "fichier $1 inexistant"
        fi ;;
    2) if( test -f $2 ) then
        echo "fichier $2 existant"
        else
        cat $1 > $2
        fi ;;
    *) echo "2 parametres attendus (au lieu de $# [${*}])" ;;
esac
```

# Itération bornée

```
for in/foreach ...do ...done/end
```

Syntaxe :

- en `sh`, `ksh`

```
for variable [ in variableliste ... ] ; do actions ; done
```

- en `csh`, `tcsh`

```
foreach variable ( variableliste ) end
```

# Examples

## En sh, bash

```
#!/usr/bin/sh  
for file in * ; do file $file ; done
```

## En csh, tcsh

```
#!/usr/bin/csh  
foreach file ( * )  
file $file  
end
```

# Itération non bornée 1/2

```
while ...do ...done/end
```

Syntaxe :

- en `sh`, `ksh`

```
while [ conditions ] ; do actions ; done
until [ conditions ] ; do actions ; done
```

- en `csh`, `tcsh`

```
while ( conditions )
[ ... ] # do actions
end
```

# Examples

```
#!/usr/bin/sh
```

```
i=1
```

```
while (test `expr $i \<= $#` -eq 1 ) ; do
```

```
    eval "echo parametre $i = \$$i " ;
```

```
    i=`expr $i + 1` ;
```

```
done
```

# Opérateur shift

Décalage logique à gauche des paramètres

On s'en sert souvent pour parcourir la liste des paramètres

## Exemples

```
#!/usr/bin/sh
```

```
while ( test `expr $#` -gt 0 ) ; do  
    echo "_____" $1  
    shift  
done
```

# La récursion

Les procédures shell peuvent s'appeler elles-mêmes;  
**Exemple:**

```
$cat cmd13.sh
```

```
#!/usr/bin/sh
```

```
fic=$1
```

```
dir=$2
```

```
cd $dir
```

```
if( test -f $fic ) then
```

```
    echo "Le fichier est dans le repertoire \c" ; pwd
```

```
    exit 0
```

```
fi
```

```
for i in * ; do
```

```
    if( test -d $i ) then
```

```
        $HOME/cmd13.sh $fic $i ;
```

```
    fi ;
```

```
done
```

# Les fonctions

Syntaxe:

```
la-fonction () { ... }
```

- Appel comme une commande shell : `la-fonction arg1 arg2`
- Accès aux paramètres par \$1 à \$9
- Retour de résultat
  - `return [valeur numérique]`
  - accès par \$? ou comme retour de fonction standard
- Peut redéfinir les commandes Unix sauf les commande spéciales  
`echo exit export pwd read test...`

# Exemple

```
#!/usr/bin/sh
```

```
arguments()
```

```
{
```

```
    echo " fonction arguments : $*"
```

```
}
```

```
echo " dans le shell : $*"
```

```
arguments "hello world"
```

# Variables globales

Par la commande `export`, on peut rendre des variables name accessibles aux sous-shells

## Exemple

- Dans la le prg appelant :

```
#!/usr/bin/sh
var1=coucou
echo " "
echo "Dans cmd15_1.sh"
echo $var1
export var1
cmd15_2.sh
```

- Dans le prg appelé

```
#!/usr/bin/sh
echo "Dans cmd15_2.sh"
echo $var1
```

# Tableaux en bash

Dans le shelle `bash` on peut utiliser des variables

- de type tableau
- à 1 dimension
- de chaînes de caractères
- 1<sup>er</sup> élément commence à 0
- l'indice `*` donne tous les éléments

# Exemple

```
#!/usr/bin/bash
declare -a Tab #facultatif

Tab=(il fait beau)

echo $Tab[1]
echo ${Tab[1]}
echo ${Tab[*]}
echo ${#Tab[*]}
```