



DIRO  
IFT 1215

# INTRODUCTION AUX SYSTÈMES INFORMATIQUES

## SCRIPTS SHELL

*Max Mignotte*

Département d'Informatique et de Recherche Opérationnelle  
Http: [//www.iro.umontreal.ca/~mignotte/](http://www.iro.umontreal.ca/~mignotte/)  
*E-mail: [mignotte@iro.umontreal.ca](mailto:mignotte@iro.umontreal.ca)*

# SCRIPTS SHELL

## SOMMAIRE

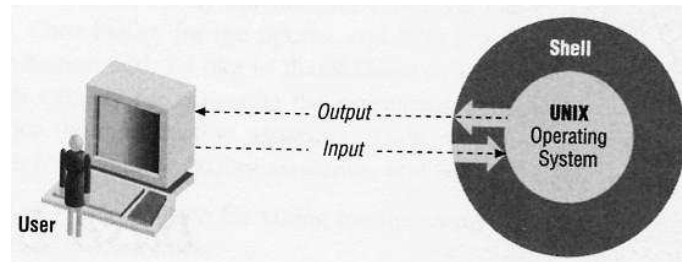
Introduction .....	2
Historique .....	4
Fondamentales .....	5
Redirection & Pipeline .....	8
Background, help, groupement .....	9
Variables .....	11
Variables arguments .....	14
Exercice .....	16
Conditions Tests .....	17
Boucles .....	22
Variables avancées .....	26
Exemple de Script .....	27
Livre de Référence .....	29

# SCRIPTS SHELL

## INTRODUCTION

### Introduction

**Shell** : Un shell est une interface utilisateur pour le système d'exploitation UNIX (ou LINUX), i.e., un programme qui prend les commandes d'un utilisateur et les traduit en instructions que l'OS peut comprendre



*La responsabilité du Shell est de*

- Fournir une **interface de ligne de commande** (interpréteur)
- **Langage de programmation** qui lui permet d'exécuter des Scripts (programme shell)
  - Faire une redirection des I/O
  - Faire une substitution de `[pathname]`, paramètre, variable
  - ...

### **Exemple**

```
sort -n phonelist > phonelist.sorted
```

- Divise la ligne de commande en “*mots*” (sort, -n, phonelist, >, phonelist.sorted)

# SCRIPTS SHELL

## INTRODUCTION

- Détermine le but de chaque mot [sort] ▷ commande, [-n, phonelist] ▷ arguments, [>, phoneliste.sorted] ▷ instructions I/O
- Organise les I/O (espace mémoire pour fichier, etc.)
- Trouve le prog. associé à la commande sort et l'exécute avec l'option *n*

### Shell est indépendant de UNIX

#### Scripts

**Scripts** : Ensemble de commandes système, mis ensemble dans un fichier texte et combiné entre eux par un langage de programmation destiné à être interprété (Shell)

#### Shell et Administrateur Système

[*SysAdmin*] utilise le shell pour communiquer avec le [*Kernel*] du système

- Configurer le système, les services du réseau, variables d'environnement etc.
- La plupart de l'administration d'un système est contrôlé par des shell scripts

[*SysAdmin*] utilise aussi le langage PERL (Pattern Extraction Language) -un langage dérivé du langage SHELL, C et d'autres langages

# SCRIPTS SHELL

## HISTORIQUE

### Historique

L'indépendance du Shell avec l'OS a conduit au développement de douzaine de shells différents (très similaire mais avec leur propre sémantique et syntaxe)

- **Bourne Shell** (*Steven Bourne*)

**sh**, Inclu dans la première version de UNIX en 1979  
Beaucoup de Scripts aidant les SysAdmin à configurer un système sont écrit en Bourne Shell

- **C Shell**

**cs**, Structure syntaxique très similaire au langage C

- **Korne Shell**

**ksh**, Extension du Bourne Shell, produit commercial d'IBM & AT&T pour la version d'UNIX d'IBM (AIX), incorporant les meilleurs caractéristiques des 2 premiers

- **Bash Shell & TcShell**

**bash**, **tcsh**, Similaire au Korne Shell mais gratuit, bash shell est le shell standart pour LINUX (bash-2.05b)

- ...

Pour savoir quel shell vous utilisez par défaut

```
echo $SHELL
```

```
> /bin/bash
```

Pour savoir la version de votre shell

```
echo $BASHVERSION
```

```
> 5.1.8(1)-release$
```

Pour savoir quels sont les shells valides sur votre machine

```
more /etc/shells
```

## SCRIPTS SHELL FONDAMENTALES

Tous les fichier script exécute par le bourne shell ont l'extension .sh (myscript.sh) et doivent commencé par

**#! /bin/bash**

Le [Kernel] exécutera /bin/bash/myscript.sh

Nota : si on utilise le shell bash ▷ le fichier myscript.bash doit comporter **#! /bin/bash** (sinon execution du bash par défaut)

et doit être un fichier rendu **exécutable**

**chmod +x myscript.sh**

### Caractères Spéciales

Caractère	Sens
;	Sépare deux instructions
~	[ <i>Home directory</i> ]
#	Commentaire
\$	Expression d'une variable
&	Pour exécuter en arrière plan [ <i>background</i> ]
*	N'importe quel chaîne de caractères
\	Affiche le prochain caractère
	[ <i>Pipe</i> ]
[... - ...]	Commence/termine un ensemble de caractère
{ ... }	Commence/termine un bloc de commande
'	Fort guillemet
"	Faible guillemet
<	Re-dirige l'entrée
>	Re-dirige la sortie
/	Séparateur de [ <i>directory</i> ] de [ <i>pathname</i> ]
?	N'importe quel unique caractère
!	Pipeline avec NON logique

## SCRIPTS SHELL FONDAMENTALES

### Exemples

Suppose que tu as les fichiers *bob*, *darlene*, *dave*,  
*ed*, *frank*, *fred* dans ton répertoire courant

Expression	Désigne
<i>fr*</i>	<i>frank, fred</i>
<i>*ed</i>	<i>ed, fred</i>
<i>b*</i>	<i>bob</i>
<i>*e*</i>	<i>darlene, dave, ed, fred</i>
<i>*r*</i>	<i>darlene, frank, fred</i>
<i>*</i>	<i>bob, darlene, dave, ed, frank, fred</i>
<i>d*e</i>	<i>darlene, dave</i>

Expression	Correspond
<i>[abc]</i>	<i>a, b, c</i>
<i>[-_]</i>	<i>-, _</i>
<i>[a - c]</i>	<i>a, b, c</i>
<i>[a - z]</i>	<i>Toute les lettres minuscules</i>
<i>[!0 - 9]</i>	<i>Tous ce qui n'est pas chiffre</i>
<i>[0 - 9!]</i>	<i>Tous les chiffres et !</i>
<i>[a - zA - Z]</i>	<i>Toute les lettres majuscule/minuscules</i>

Pour lister tous les fichier .c .h .o

**ls \*.*[cho]***

Pour lister tous les fichier commençant par "b" et "e"  
de tous les utilisateurs (usr1, usr2,etc.)

**ls /usr\*/*[be]*\***

## SCRIPTS SHELL FONDAMENTALES

Pour afficher tous les noms de fichier commençant par  
code. et finissant par un caractère au plus

```
ls code.?
```

### Accolade

Pour afficher *beds, bolts, bars*

```
echo b{ed,olt,ar}s
```

Pour afficher *bards, barns, barks, beds*

```
echo b{ar{d,n,k},ed}s
```

```
ls *.{c,h,o}
```

### Guillemet

Les caractères spéciales (slide 5) doivent être *protégés*  
si on désire les afficher, les rechercher, etc.

```
echo "2 * 3 > 5 est vrai"  
echo 2 \* 3 \> est vrai
```

Exemple de commande qui utilise souvent  
des caractères spéciales

- echo
- find  
find -name \\*.jpg
- grep  
grep image \*.tex



## SCRIPTS SHELL

### REDIRECTION ET PIPELINE

Une des forces du **Bash** est de pouvoir contrôler précisément d'où viennent et où vont les entrées et sorties d'un programme

#### I/O Streams

*Il y a trois fichiers standards associés à un programme*

- Standard Input (stdin): normalement le clavier
- Standard Output (stdout): normalement l'écran
- La sortie Erreur (stderr): normalement l'écran aussi

*Un programme prend son entrée au clavier  
et écrit ses résultats à l'écran  
Or Shell permet de changer ce comportement*

Exemple	Description
cmd1   cmd2	stdout de cmd1 ▷ stdin de cmd2
cmd1 < file	Lit le stdin à partir d'un fichier
cmd » file	Ecrire stdout à la fin d'un fichier
cmd < f1 > f2	Stdout de cmd < f1 dans fichier f2

Équivalent de cp f1 f2

**cat < f1 > f2**

Affichage du listing d'un répertoire page/page

**ls -l | more**

Affichage avec trie alphabétique du 1er champ de myfile

**cut -d: -f1 myfile | sort**

## SCRIPTS SHELL

### BACKGROUND, HELP, GROUPEMENT

#### Background

Si tu veux exécuter une commande qui ne requiert pas d'entrée de l'utilisateur et que tu veux faire autre chose pendant que la commande s'exécute, met un & après la commande

Décompresser un gros fichier gc.tar.Z en tache de fond

**uncompress gcc.tar &**

[1]+ Done uncompress gcc.tar

Différence de deux fichiers et redirection dans filedif.txt

**diff file1.txt file1.txt.old > filedif &**

#### Help

Pour avoir de l'aide/info sur une commande

**man diff**

#### Groupement de commande

Exemple	Description
cmd1 ; cmd2	Execute cmd1 et cmd2
cmd1 && cmd2	Execute cmd2 ssi cmd1 a réussi
cmd1    cmd2	Execute cmd2 ssi cmd1 a échoue

**date; pwd; ls**

**date; pwd > oufile**

## **SCRIPTS SHELL**

### **BACKGROUND, HELP, GROUPEMENT**

Chercher la chaîne de caractères "error" et affichage si elle a été trouvée

```
grep "error" file.txt && lpr file.txt
```

Chercher la chaîne de caractères "error" et affichage pas d'erreur si elle n'a pas été trouvée

```
grep "error" file.txt | | echo "pas d'erreur"
```

## SCRIPTS SHELL VARIABLES

### Définition

**var=value**

Doit être mis entre guillemet si value contient des espaces ou des caractères spéciales

**hi="Hello World"**

### Substitution de variables

Pour utiliser la valeur d'une variable

**echo \$hi**

> Hello World

**echo \${hi}s**

> Hello Worlds

**cmd=echo  
\$cmd \${hi}s**

> Hello Worlds

**moi=max**

**echo Mon nom est \$moi**

**echo Mon nom est \ \$moi**

**echo "Mon nom est \$moi"**

**echo "Mon nom est \ \$moi"**

**echo Mon nom est \$Moi**

Pour effacer une définition de variable

**unset moi**

**echo Mon nom est \$moi**

> Mon nom est

## SCRIPTS SHELL VARIABLES

```
i=0 j=10 k=100
echo $i $j
> 0 10

echo la valeur de \ $j est \"$j\"
> la valeur de $j est "10"

echo ${j}234
> 10234
```

Erreur ▷ **i = 10**

### Variables prédéfinies

VARIABLES	SENS
USER	Utilisateur
HOME	répertoire où est exécuté le shell
BASH	nom du shell
BASH_VERSION	version du shell
BASH_VERSINFO	+ info sur le shell
PWD	Chemin du répertoire
SECONDS	Nb de secondes depuis que le shell est invoqué

```
echo $USER $BASH $BASH_VERSION $BASH_VERSINFO
> mignotte /bin/bash 5.1.8(1)-release
```

## SCRIPTS SHELL VARIABLES

### Assignement Arithmétique

```
x=1+4
```

```
echo $x
```

```
> 1+4
```

```
let x=1+4
```

```
echo $x
```

```
> 5
```

```
let x="1+4" ou let "x=1+4"
```

```
echo $x
```

```
> 5
```

```
let x=2+3*5
```

```
let x+=5
```

```
let x=x+5
```

```
echo $x
```

```
> 27
```

```
let x=2+3*5
```

```
let x+=5
```

```
x=x+5
```

```
echo $x
```

```
> x+5
```

```
let x=2+3*5
```

```
b=5
```

```
x=$x+$b
```

```
echo $x
```

```
> 17+5
```

```
let x=2+3*5
```

```
b=5
```

```
x=$((x+$b))
```

```
echo $x
```

```
> 22
```

Remarque: "let" n'existe pas dans le shell sh

## SCRIPTS SHELL VARIABLES ARGUMENTS

### Variables arguments

*variables dédiées aux arguments du programme script*

VARIABLES	SENS - VALEUR
#	Nombre d'arguments
0	Nom du script
<i>n</i>	(nb 1-9) Nième arguments
*	Tous les arguments
@	Tous les arguments
\$	ID process

```
#!/bin/bash
```

```
# argv.sh script
```

```
echo Il y a $# arguments
```

```
echo Le nom du script est $0
```

```
echo $1 est le premier argument
```

```
echo $2 est le second argument
```

```
echo $3 est le troisième argument
```

```
echo Tous les arguments : $@
```

```
echo Numéro du process : $$
```

```
argv.sh arg1 arg2 arg3
```

```
> Il y a 3 arguments
```

```
> Le nom du script est argv.sh
```

```
> arg1 est le premier argument
```

```
> arg2 est le second argument
```

```
> arg3 est le troisième argument
```

```
> Tous les arguments : arg1 arg2 arg3
```

```
> Numéro du process : 30116
```

## SCRIPTS SHELL VARIABLES ARGUMENTS

*Commande shift*

```
#!/bin/bash
# shiftargv.sh script

echo Avant le shift
echo $1 est le premier argument
echo $2 est le second argument
echo $3 est le troisième argument
echo Tous les arguments : $@
shift
echo Après le Shift
echo $1 est le premier argument
echo $2 est le second argument
echo $3 est le troisième argument
echo Tous les arguments : $@
```

### **shiftargv.sh arg1 arg2 arg3**

- > Avant le shift
- > Le nom du script est argv.sh
- > arg1 est le premier argument
- > arg2 est le second argument
- > arg3 est le troisième argument
- > Tous les argument : arg1 arg2 arg3
- > Après le shift
- > arg2 est le premier argument
- > arg3 est le second argument
- > est le troisième argument
- > Tous les argument : arg2 arg3

*Commande shift n*  
*Décalage de n arguments*



## SCRIPTS SHELL EXERCICE

### Exercice

*Usage :*  
*findfile.sh string dirname*

```
#!/bin/bash  
# findfile.sh script  
find $2 -type f -exec grep $1 \{\} \;
```

#### **find *arg***

Va chercher tous les fichiers dans le répertoire spécifié en arg de type fichier (-type f)

-exec : envoie le nom du fichier en cours de traitement comme argument à la commande qui suit (\{\})

#### **grep *arg1 arg2***

Recherche la présence du mot donné en arg1 dans le fichier donné en arg2

## SCRIPTS SHELL

### CONDITIONS - TESTS

#### Condition - Test

```
if [ expression ]  
then  
  command1  
  command2  
fi
```

```
if [ expression ]; then  
  command1  
  command2  
fi
```

```
if [ $day == "Monday" ]  
then  
  echo First day of the week  
fi
```

#### *Conditions sur les fichiers/directories*

Syntaxe	Correspondance
-d name	name existe et c'est un répertoire
-f name	name existe et c'est un fichier
-r name	name existe et il est lisible (fichier ou rep.)
-w name	name existe et on peut écrire dessus (fichier ou rep.)
-x name	name existe et il est exécutable (fichier ou rep.)

## SCRIPTS SHELL

### CONDITIONS - TESTS

#### *Conditions sur les entiers*

Syntaxe	Correspondance
$n1 -eq\ n2$	$n1 = n2 ?$
$n1 -ne\ n2$	$n1 \neq n2 ?$
$n1 -gt\ n2$	$n1 > n2 ?$
$n1 -ge\ n2$	$n1 \geq n2 ?$
$n1 -lt\ n2$	$n1 < n2 ?$
$n1 -le\ n2$	$n1 \leq n2 ?$

#### *Conditions sur les chaînes de caractères*

Syntaxe	Correspondance
<i>string</i>	<i>string n'est pas nul</i>
<i>-n string</i>	<i>string n'est pas de longueur nulle</i>
<i>-z string</i>	<i>string est de longueur nulle</i>
<i>s1 == s2</i>	<i>string s1 identique de s2</i>
<i>s1 != s2</i>	<i>string s1 différent de s2</i>

#### *Conditions composées*

Syntaxe	Correspondance
<i>( expr )</i>	<i>vraie si l'expression entre parenthèse est vraie</i>
<i>! expr</i>	<i>vraie si l'expression est fausse</i>
<i>expr1 -a expr2</i>	<i>vraie si les 2 expressions sont vraies</i>
<i>expr1 -o expr2</i>	<i>vraie si l'une des 2 expressions est vraie</i>

## SCRIPTS SHELL

### CONDITIONS -TESTS

#### Exemples

```
if [ $# -lt 1 ]
```

Si le nb d'argument est < 1

```
if [ -f $1 ]
```

Si l'argument 1 est le nom d'un fichier

```
if [ -d /tmp ]
```

Si le répertoire /tmp existe

```
if [ $x ]
```

Si la valeur de la variable x est non nulle/défini

```
if [ $x != error ]
```

Si la valeur de la variable x n'est pas *error*

```
if [ $height -gt 64 -a $weight -ge 120 ]
```

Si ((height>64)&&(weight≥120))

```
if [ $day == Sat -o $day == Sun ]
```

Si la valeur de *day* est *Sat* ou *Sun*

```
if [ ! \(-f $1 -a -r $1\) ]
```

Si le premier argument n'est pas (un fichier lisible)

```
if [ : ]
```

Toujours vraie

```
if [ file1 -nt file2 ]
```

Si file1 est plus nouveau que file2

## SCRIPTS SHELL

### CONDITIONS -TESTS

```
if [ expression1 ]; then
  cmd1
elif [ expression2 ]; then
  cmd2
else
  cmd3
fi
```

```
#!/bin/bash
# mylist.sh script

if [ $# -ne 1 ]; then
  echo Mauvais nombre d \ ' argument
  exit
fi

if [ -d $1 ]; then
  cat $1
else
  echo $1 n \ ' est pas un répertoire de fichier
fi
```

## SCRIPTS SHELL

### CONDITIONS - TESTS

```
case value in
  pattern1)
    cmd1;;
  pattern2)
    cmd2;;
  .
  .
esac
```

```
#!/bin/bash
# octaldisplay.sh script

if [ $# -ne 1 ]; then
  echo Mauvais nombre d' argument
  exit
fi

case $1 in
  0) echo Pas de permission;;
  1) echo Permission exécuté;;
  2) echo Permission écriture;;
  3) echo Permission exécuté et écriture;;
  4) echo Permission lecture;;
  5) echo Permission lecture et exécuté;;
  6) echo Permission lecture et écriture;;
  7) echo Permission lecture, écriture et exécuté;;
  *) echo Ce n'est pas un nombre octal d' 1 digit;;
esac
```

# SCRIPTS SHELL

## BOUCLES

### Boucles

```
for var in list
do
  cmds
  ...
done
```

```
for name in Harry Susan Bob Jane
do
  echo Hello $name
done
```

```
> Hello Harry
> Hello Susan
> Hello Bob
> Hello Jane
```

### Exemples

```
#!/bin/bash
# listc.sh script

for name in *.c
do
  echo $name
done
```

```
#!/bin/bash
# pargc.sh script

for arg in $*
do
  echo $arg
done
```

## SCRIPTS SHELL

### BOUCLES

#### Boucles

```
while [ expression ]
do
  cmds
done
```

```
#!/bin/bash
# countdown.sh script
a=100
while [ $a -gt 0 ]; do
  echo -n "$a "
  a=$((a - 1))
done
echo -e "\ Booum !"
```

```
#!/bin/bash
# countdownarg.sh script
while [ $# -gt 0 ]; do
  echo $1
  shift
done
```

```
#!/bin/bash
# count.sh script
i=1
while [ $i -ge 0 ]; do
  echo $i
  let i = $i + 1
done
```



## SCRIPTS SHELL BOUCLES

### Boucles

```
until [ expression ]  
do  
  cmds  
done
```

```
for (( init; cond. d'arrêt; incrémente ))  
do  
  cmds  
done
```

```
#!/bin/bash  
# count.sh script  
for (( i=0; i<100; i++ ))  
do  
  echo $i  
done
```

### La commande Break

```
#!/bin/bash  
# option.sh script  
YES=1   Ot=0   Oa=0   OI=0  
for arg in $*  
do  
  case $arg in  
    -t) Ot=$YES;;  
    -a) Oa=$YES;;  
    -l) OI=$YES;;  
    *) echo argument invalide  
       break  
  esac  
done  
echo option -t -a -l : $Ot $Oa $OI
```

## SCRIPTS SHELL

### BOUCLES

```
#!/bin/bash
# loop.sh script
i=1
while [ : ]
do
    echo $i
    let i=$i+1
    if [ $i -gt 10 ]; then
        break
    fi
done
```

#### Exemples

```
#!/bin/bash
# convert.sh script
for filename in *.jpg *.gif *.tif
do
    ppmfile=${filename%.*}.ppm
    echo conversion de $filename en $ppmfile
    convert $filename $ppmfile
done
```

- `${param%pattern}`  
-Efface le pattern de fin à la string param
- `${param##pattern}`  
-Efface le pattern de début à la string param

## SCRIPTS SHELL VARIABLES AVANCÉS

### Variables Avancés

*Syntaxe de la substitution de variables avancées*

Syntaxe	Correspondance
<code>\${var:-val}</code>	<i>Si var existe on l'utilise sinon utilise val</i>
<code>\${var:=val}</code>	<i>Si var existe on l'utilise sinon var=val</i>
<code>\${var:?message}</code>	<i>Si var existe on l'utilise sinon affiche le message et exit</i>

```
echo je suis ${moi:-personne}
```

```
> Je suis personne
```

```
moi=max
```

```
echo je suis ${moi:-personne}
```

```
> Je suis max
```

```
echo $today does not exist
```

```
> does not exist
```

```
echo ${today:-'date'}
```

```
> Wed Dec 17 16:52:10 EST 2003
```

*Usage :*

*findfile.sh string dirname*

```
#!/bin/bash
```

```
# findfile.sh script
```

```
find ${2:-.} -type f -exec grep $1 \{\} \;
```

## SCRIPTS SHELL

### EXEMPLES DE SCRIPTS

#### Exemples

---

```
#!/bin/bash
# devine.sh script
sec='date +%S'
let num=$sec%10

if [ $num -eq 0 ]; then
    num=10
fi

while [ : ]
do
    echo Devine un nombre entre 1 et 10
    read x
    if [ $x -eq $num ]; then
        echo Gagné
        break
    else
        echo Désolé essaie encore
    fi
done
```

---

```
#!/bin/bash
# nbfiles.sh script
nbfiles='ls * | wc -l'
echo $nbfiles
```

---

## SCRIPTS SHELL

### EXEMPLES DE SCRIPTS

---

*Usage :*

*edg.sh string1 string2 file1*  
*substitue toute string1 dans file1 par string2*

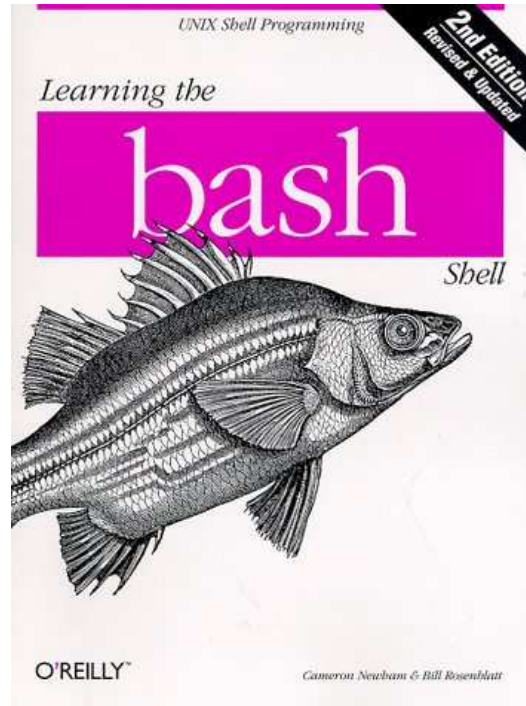
```
#!/bin/bash
# edg.sh script
ed $3 <%
g/$1/s//$2/g
w
%
```

---

```
#!/bin/bash
# rmexec.sh script
for x in *
do
[ -x $x -a -f $x.c ] && echo $x; done | xargs rm -f
```

**SCRIPTS SHELL**  
**LIVRE DE RÉFÉRENCE**

*Pour en savoir plus*



**Learning the Bash Shell**  
*O'Reilly, Cameron Newham & Bill Rosenblatt*  
ISBN 1-565992-347-2