Scripts SHELL 1

Matière de Max Mignotte

Modifications: Alena Tsikhanovich

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

Shell

- Responsabilités
  - Fournir
    - Une interface de ligne de commande (interpréteur)
    - Un langage de programmation qui permet d'exécuter des Scripts (programmes shell)
      - Faire une redirection des I/O
      - Faire une substitution de paramètres, variables, …
- Exemple
  - **`sort -n phonelist > phonelist.sorted`**
    - Trier (**`sort`**) dans l'ordre numérique (**`-n`**) et rediriger (**`>`**) le résultat de l'exécution de la commande dans le fichier **`phonelist.sorted`**

Scripts

- Ensemble de commandes système mises ensemble dans un fichier texte et combiner entre eux par un langage de programmation destiné à être interprété (Shell)

Shell et Administrateur Système

- Utilise le shell pour communiquer avec le noyau du système
  - Configurer le système, les services du réseau, variables d'environnement etc.
  - La plupart de l'administration d'un système est contrôlée par des scripts shell

Historique

- L'indépendance du Shell de l'OS a conduit au développement de douzaine de shells différents (très similaires mais avec leurs propres sémantiques et syntaxes)
  - Bourne Shell (Steven Bourne) – sh
    - Inclus dans la première version de UNIX en 1979
  - C Shell - csh, 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

Syntaxe d'une commande Unix ou de bash

[chemin/]nom_cmd [option ...] [argument ...]

- Une *option* est généralement introduite par le caractère **tiret** (ex : **-a**) ou dans la syntaxe GNU par deux caractères **tirets** consécutifs (ex : **--version**). Elle précise un fonctionnement particulier de la commande
- Les *arguments* désignent les objets sur lesquels doit s'exécuter la commande.
- Lorsque l'on souhaite connaître la syntaxe ou les fonctionnalités d'une commande *cmd* il suffit d'exécuter la commande **man** *cmd* (ex : **man echo**). L'aide en ligne de la commande *cmd* devient alors disponible

Historique

- Bash Shell & TsShell – **bash**, **tcsh**
  - Similaires au Korne Shell, mais gratuits, bash shell est un shell standard pou LINUX
- Pour savoir quel shell vous utilisez par défaut

```
echo $SHELL
> /bin/tcsh
```

- Pour savoir quels sont les shells valides sur votre machine

```
more /etc/shells
```

Configuration

- Pour changer le shell UNIX, modifiez le contenu du fichier `.pshrc` afin qu'il contienne **UNE** ligne de la forme
  - `SHELL=bash`

ou

  - `SHELL=tcsh`

Accès à Linux en ayant un autre SE

- Il y a plusieurs options:

1) utilisez le serveur de l'université de Montréal à distance:
https://acces-web.sens.umontreal.ca/nxwebplayer

2)Windows Subsystem for Linux (Windows 10 and Windows Server 2019):
https://docs.microsoft.com/en-us/windows/wsl/install-win10

3) "linux in the browser" :
https://bellard.org/jslinux/vm.html?url=alpine-x86-xwin.cfg&mem=256&graphic=1

4) utilisez un "linux container" : https://linuxcontainers.org/lxd/try-it/

5) installez linux dans VirtualBox : https://www.virtualbox.org/wiki/Guest_Oses

6) installez linux en "dual boot" :
https://www.instructables.com/How-to-Dual-boot-Linux-and-Windows-on-a-PC-with-W/

Éditer les scripts à distance avec emacs

- **`emacs -nw script.sh`**
- modifier le fichier **`script.sh`**
- sauver avec **`C-x C-s`**
- quitter avec **`C-x C-c`**

Scripts SHELL, base

- Tous les fichiers script exécutés par le shell ont l'extension .sh
  - **`myscript.sh`**
- Doivent commencer par la ligne (précise au shell courant quel interpréteur doit être utilisé)
  - **`#! /bin/sh`**
- Un fichier doit être rendu exécutable
  - **`chmod +x myscript.sh`**
    - **`chmod`** – commande "**`change mode`** "
    - **`+x`** – ajouter la permission en exécution
    - Argument 2 de la commande **`chmod`** nom de fichier(répertoire)
- Lancez l'exécution: **`./myscript.sh`**

Caractères Spécieux

Caractère	Sens
~	Répertoire maison
#	Commentaire
$	Expression d'une variable
&	Exécution en arrière plan
*	N'importe quelle chaîne de caractères
|	« Pipe », L'opérateur | , appelé *tube*, est un opérateur caractéristique des shells et connecte la sortie d'une commande à l'entrée de la commande suivante.
[...-...]	Commence/termine un ensemble de caractères
{ ... }	Commence/termine un bloc de commandes
;	Sépare deux instructions
‘	Guillemet fort
“	Guillemet faible

Caractères Spéciales

Caractère	Sens
<	Redirection de l'entrée
>	Redirection de la sortie
/	Séparateur de répertoires de chemin de nom
?	N'importe quel unique caractère
!	Pipeline avec Non logique

Exemples

- Supposons qu'on a les fichiers **`bob, darien, dave, ed, frank, fred`** dans le répertoire courant

Expression	Désignation
`fr*`	`frank, fred`
`*ed`	`ed, fred`
`b*`	`bob`
`*e*`	`dariene, dave, ed, fred`
`*r*`	`dariene, frank, fred`
`*`	`bob, darien, dave, ed, frank, fred`
`d*e`	`dariene, dave`

Exemples

- Les crochets permettent de spécifier un ensemble de caractères explicite ou intervalle de caractères séparés par un tiré

Expression	Désignation
[abc]	a, b, c
[-_]	-, _
[a-c]	a, b, c
[a-z]	Toutes les lettres minuscules
[!0-9]	Tout ce qui n'est pas chiffre
[0-9!]	Tous les chiffres et symbole !
[a-zA-Z]	Toutes les lettres majuscules/minuscules

Scripts SHELL, base

- Pour lister tous les fichiers .c, .h et .o
  - `ls *.[cho]`
  - `ls *.{c,h,o}`
  - `*` – une suite quelconque (éventuellement vide) de caractères
- Pour lister tous les fichiers commençants par « b » et « e » de tous les utilisateurs (usr1, usr2, etc.)
  - `ls /usr*/[be]*`
- Pour afficher tous les noms de fichier commençants par code. et finissants par un caractère au plus
  - `ls code.?`

Scripts SHELL, base

- 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}`**
- Les caractères spécieux doivent être protégés si on désire les afficher, les rechercher, etc.
  - **`echo "2*3 > 5 est vrai"`**
  - **`echo 2\*3\> est vrai`**
- Exemple de commandes qui utilisent souvent des caractères spécieux
  - **`echo`**
  - **`find: find –name \*.jpg`**
  - **`grep: grep image *.tex`**

Scripts SHELL, base

- Une des forces du Bash est de pouvoir contrôler précisément d' où viennent et où vont les entrées et sorties d'un programme
- Flux d'entrés/sorties – 3 fichiers standards associés à un programme
  - Entrée standard (stdin): normalement le clavier
  - Sortie standard (stdout): normalement l'écran
  - Sortie Erreur (stderr): normalement l'écran aussi
- Un programme prend son entrée au clavier et écrit ses résultats à l'écran
- SHELL permet de changer ce comportement

Scripts SHELL, base

- Redirection d'entrées/sorties
- SHELL permet de changer ce comportement

Exemple	Description
cmd1 | cmd2	stdout de cmd1 est stdin de cmd2
cmd1 < file	Lire le stdin à partir d'un fichier
cmd1 >> file	Écrire stdout à la fin d'un fichier
cmd1 < file1 > file2	Stdout de l'exécution de cmd1 avec file1 comme entrée écrire dans le file2

Scripts SHELL, base

- Commande copier
  - **`cp file1 file2`**
- L'action équivalente avec les redirections i/o
  - **`cat < file1 > file2`**
- Affichage du listing d'un répertoire page/page
  - **`ls -l | more`**
- Affichage avec tri alphabétique du 1er champ de `myfile`
  - **`cut -d : -f1 myfile | sort`**
    - **`cut`** – découpage
    - Option **`-d`** – délimiteur
    - **`f1`** – "field 1"

Scripts SHELL, base

- Pour exécuter une commande qui ne requiert pas d'entrée de l'utilisateur et durant ce temps en parallèle de faire une autre chose, utilisez un symbole & après la commande pour l'exécuter en tâche de fond
- Exemple: décompresser un gros fichier `gcc.tar` en tâche de fond
  - **`uncompress gcc.tar &`**
- Différence de deux fichiers et redirection du résultat dans **`filedif.txt`**
  - **`diff file1.txt file1.txt.old > filedif &`**
- Pour avoir de l'aide/info sur une commande
  - **`man diff`**

Scripts SHELL, base

- Groupements de commandes

Exemple	Description
cmd1; cmd2	Exécute cmd1 et ensuite cmd2
cmd1 && cmd2	Exécute la seconde commande ssi cmd1 est réussie
cmd1 || cmd2	La seconde commande n'est exécutée que si la première est fausse

- Exemples:
  - **`date; pwd; ls`**
    - **`date`** – afficher la date courante
    - **`pwd`** – afficher le répertoire courant
    - **`ls`** – afficher le contenu du répertoire courant
  - **`date; pwd > outfile`**

Scripts SHELL, base

- Groupements de commandes
- Chercher la chaîne de caractères « error » dans le fichier **`file.txt`** et afficher « error » si ce mot a été trouvé

```
grep ‘error‘ file.txt && echo “error“
```

- Chercher la chaîne de caractères « error » dans le fichier **`file.txt`** et afficher le message **“`pas d’erreurs`“** si cette chîne n’a pas été trouvée
  - **`grep ‘error‘ file.txt || echo “pas d’erreurs“`**

Scripts SHELL, Variables

- Définition
  - **var=valeur**
- Il faut mettre les guillemets si la valeur contient des espaces ou des caractères spéciaux
  - **hi="Hello World"**
- Substitution de variables
- Pour utiliser la valeur d'une variable
  - **echo $hi**

    > Hello World
  - **echo ${hi}s**

    > Hello Worlds

Scripts SHELL, Variables

- Substitution de variables

```
cmd=echo
$cmd ${hi}s
> Hello Worlds
```

- Exemple

```
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
```

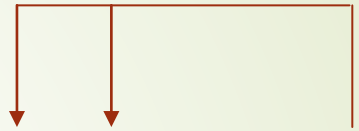
Exécution

```
> Mon nom est max
> Mon nom est $moi
> Mon nom est max
> Mon nom est $moi
> Mon nom est
```

Scripts SHELL, Variables

- Pour effacer une définition de variable

```
unset moi
echo Mon nom est $moi
> Mon nom est
```



Erreur: `i = 10` (espaces!)

```
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 =>` Exécution: `> 10234`

Scripts SHELL, Variables

- 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 seconds depuis que le shell est invoqué

```
echo $USER $BASH $BASH_VERSION $BASH_VERSINFO
> tsikhana /bin/psh 4.4.12(1)-release 4
```

Scripts SHELL, Variables Arithmétiques, assignations

```
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
```

Scripts SHELL, Variables Arithmétiques, assignations

```
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
```

Scripts SHELL, Variables

- Variables arguments – variables dédiées aux arguments du programme script

Variables	Sens - Valeur
#	Nombre d'arguments
0	Nom du script
n	(nb 1-9) Nième argument
*	Tous les arguments
@	Tous les arguments
$	ID process

```
#!/bin/sh
#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

La commande interne **shift** décale la numérotation des paramètres de position
Syntaxe : **shift [n]**
(n+1) => *1* , *(n+2)* => 2 , etc.
Une fois le décalage effectué, le paramètre spécial **#** est mis à jour

```
#!/bin/sh
#shiftarv.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: $@
```

Scripts SHELL, Variables Arguments Commande shift

```
shiftarv.sh arg1 arg2 arg3
> Avant le shift
> arg1 est le premier argument
> arg2 le second argument
> arg3 est le troisième argument
> Tous les arguments: arg1 arg2 arg3
> Après le shift
> arg2 est le premier argument
> arg3 est le second argument
> est le troisième argument
> Tous les arguments: arg2 arg3
```

```
#!/bin/sh
#shiftarv.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: $@
```