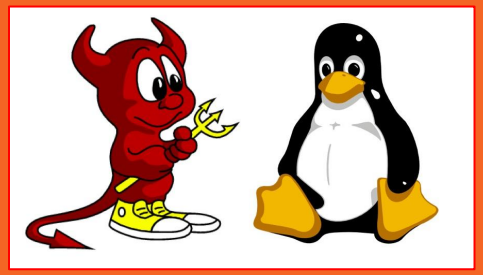

Unix/Linux
ISEN - AP4

4 - Bash script basics

2019 - J. Hochart

Infos

Référence

Supports disponibles en ligne après le cours (j+1):

www.hochart.fr/LIN

Contact

Pour toute question sur le cours ou les exos:

jul@hochart.fr

Licence

Références (4 - Shell script basics)

Livres

- Linux in à nutshell - Edition 6
  - Auteurs: Ellen Siever, Stephen Figgins

Shell

Programme qui agit en **intermédiaire entre l'utilisateur et l'OS**. Le shell est un **interpréteur** de **commandes**.

Variétés

- **Original Bourne Shell - /bin/sh** (de nos jours c'est un symlink vers /bin/bash): standard pour le scripting
- **C Shell** de Berkeley **(CSH et TCSH)**: a apporté des features: historique des commandes et job control: standard pour l'interactif
- **Korn shell (KSH):** Bourne Shell intégrant des fonctions de CSH
- **Bash**: Bourne Again Shell: Réécriture GNU + Posix
- **Zsh:** Evolution de Ksh avec des features add.
- Ici, on parle de Bash (standard)

Invoquer bash

- bash [options] [arguments]
  - -c cmd : commande à exécuter
  - -s: lire les commandes sur STDIN

  (les plus utiles)

Fichiers de conf

- **/etc/profile:** exécuté à chaque login
- Le 1er fichier trouvé parmi: **~/.bash_profile, ~/.bash_login, ou ~/.profile:** exécuté à chaque login
- **~/.bashrc:** exécuté à chaque fois que bash est lancé *(sauf si il est invoqué par /bin/sh, alors $ENV est utilisé)*

Globbing - base

Characters	Meaning
*	Match any string of zero or more characters.
?	Match any single character.
[*abc...*]	Match any one of the enclosed characters; a hyphen can specify a range (e.g., **a-z**, **A-Z**, **0-9**).
[**!***abc...*], [^*abc...*]	Match any character *not* enclosed as above.
~	Home directory of the current user.
~*name*	Home directory of user *name*.
~+	Current working directory ($PWD).
~-	Previous working directory ($OLDPWD).

Linux in à nutshell - 6th Ed - O'Reilly - Filename metacharacters

Classes POSIX

Class	Characters matched	Class	Characters matched
alnum	Alphanumeric characters	**graph**	Nonspace characters
alpha	Alphabetic characters	**print**	Printable characters
blank	Space or tab	**punct**	Punctuation characters
cntrl	Control characters	**space**	Whitespace characters
digit	Decimal digits	**upper**	Uppercase characters
lower	Lowercase characters	**xdigit**	Hexadecimal digits

Linux in à nutshell - 6th Ed - O'Reilly - POSIX classes

```
printf 'Please enter a character: '
IFS= read -r c
case $c in
  ([[:lower:]]) echo lowercase letter;;
  ([[:upper:]]) echo uppercase letter;;
  ([[:alpha:]]) echo neither lower nor uppercase letter;;
  ([[:digit:]]) echo decimal digit;;
  (?) echo any other single character;;
  ("") echo nothing;;
  (*) echo anything else;;
esac
```

Examples - Globbing

$ ls new* #new, new.1

$ cat ch? #ch9 mais pas ch10

$ vi [A-R]* #Fichiers qui commencent par A..B..R

$ pr !(*.o|core) | lp # Afficher les fichiers qui ne sont ni des objets ni des coredumps

Caractères spéciaux

Character	Meaning
;	Command separator
&	Background execution
( )	Command grouping
|	Pipe
< > &	Redirection symbols
* ? [ ] ~ + - @ !	Filename metacharacters
" ' \	Used in quoting other characters
`	Command substitution
$	Variable substitution (or command or arithmetic substitution)
space tab newline	Word separators

Linux in à nutshell - 6th Ed - O'Reilly -Special characters

Quoting

“ “: Tout ce qui est entre est pris littéralement, excepté

- **$VAR:** Sera substitué par le contenu de VAR
- `: Sera substitué (commande)
- “: Marque la fin

‘ ‘: Tout ce qui est entre est pris littéralement, sauf ‘

\: Le caractère d’après est pris littéralement

- Entre “ ”, utilisé pour escaper “ **$ ou** ‘
- Peut escaper lui-meme (\\), un espace ou un newline

Caractères spéciaux

Sequence	Value	Sequence	Value
\a	Alert	\t	Tab
\b	Backspace	\v	Vertical tab
\c*X*	Control character X	*nnn*	Octal value *nnn*
\e	Escape	\x*nn*	Hexadecimal value *nn*
\E	Escape	\'	Single quote
\f	Form feed	\"	Double quote
\n	Newline	\\	Backslash
\r	Carriage return		

Linux in à nutshell - 6th Ed - O'Reilly -Special characters

Exemples

```
$ echo 'toto "tutu" titi'

toto "titi" tutu

$ echo "plop 'blah \"woot\""

plop 'blah "woot"

$ echo "nb de fichiers: `ls | wc -l`"

nb de fichiers:  18

$ echo "La variable \$x contient $x"

La variable $x contient 443
```

Syntaxe des commandes

Syntax	Effect
cmd &	Execute *cmd* in background.
cmd1 ; *cmd2*	Command sequence; execute multiple *cmd*s on the same line.
{ *cmd1* ; *cmd2* ; }	Execute commands as a group in the current shell.
(*cmd1* ; *cmd2*)	Execute commands as a group in a subshell.
cmd1 | *cmd2*	Pipe; use output from *cmd1* as input to *cmd2*.
cmd1 `cmd2`	Command substitution; use *cmd2* output as arguments to *cmd1*.
cmd1 $(*cmd2*)	POSIX shell command substitution; nesting is allowed.
cmd $((*expression*))	POSIX shell arithmetic substitution. Use the result of *expression* as argument to *cmd*.
cmd1 && *cmd2*	AND; execute *cmd1* and then (if *cmd1* succeeds) *cmd2*. This is a "short-circuit" operation; *cmd2* is never executed if *cmd1* fails.
cmd1 || *cmd2*	OR; execute either *cmd1* or (if *cmd1* fails) *cmd2*. This is a "short-circuit" operation; *cmd2* is never executed if *cmd1* succeeds.
! *cmd*	NOT; execute *cmd*, and produce a zero exit status if *cmd* exits with a nonzero status. Otherwise, produce a nonzero status when *cmd* exits with a zero status.

Linux in à nutshell - 6th Ed - O'Reilly -Command forms

Exemples

```
$ nroff file > file.txt &  # Format in the background

$ cd; ls # Execute sequentially

$ (date; who; pwd) > logfile # All output is redirected

$ sort file | pr -3 | lp # Sort file, page output, then print

$ vi `grep -l ifdef *.c` # Edit files found by grep

$ egrep '(yes|no)' `cat list` # Specify a list of files to search

$ egrep '(yes|no)' $(cat list) # POSIX version of previous

$ egrep '(yes|no)' $(< list) # Faster, not in POSIX

$ grep XX file && lp file # Print file if it contains the pattern;

$ grep XX file || echo "XX not found" # Otherwise, echo an error message
```

Redirections

- **0 - STDIN - Entrée standard**
- **1 - STDOUT - Sortie standard**
- **2 - STDERR - Sortie d'erreur**

cmd > file : Envoyer la sortie de cmd vers file (**écrasement**)

cmd >> file: Envoyer la sortie de cmd vers file (**append**)

cmd < file : **Input** de cmd = file

Redirection multiple

Syntax	Effect
cmd **2>***file*	Send standard error to *file*; standard output remains the same (e.g., the screen).
cmd **>** *file* **2>&1**	Send both standard error and standard output to *file*.
cmd **&>>***file*	Append both standard error and standard output to *file*.
cmd **&>** *file*	Same. Preferred form.
cmd **>&** *file*	Same.
cmd **>** *f1* **2>***f2*	Send standard output to file *f1*, standard error to file *f2*.
cmd **| tee** *files*	Send output of *cmd* to standard output (usually the terminal) and to *files*. (See the example in Chapter 3 under **tee**.)
cmd **2>&1 | tee** *files*	Send standard output and error output of *cmd* to standard output (usually the terminal) and to *files*.
cmd **|&**	Same as *cmd* **2>&1 |** to send standard error through a pipe.

Linux in à nutshell - 6th Ed - O'Reilly -Multiple redirection

Fonctions

```
# fatal --- print an error message and die:
fatal () { # defining function fatal
    echo "$0: fatal error:" "$@" >&2 # messages to standard
    error
    exit 1
}
...
if [ $# = 0 ] # not enough arguments
then
    fatal "not enough arguments" # call function with message
fi
```

Variables

```
VAR=1234                    # définition
VAR=ABC$VAR                 # concaténation
echo $VAR                   # ABC1234
```

- Utiliser des braces si la variable n’est pas séparée du texte
  - echo ${u}root

Variables builtin de Bash

- $# Nb d'args en ligne de commande
- $? Code re retour de la dernière commande
- $$ PID du process en cours
- $0 Argument 0: le script
- $n Argument N: 1 = token 1, ...
- $*, $@ Tous les arguments de la CLI ($1 $2 ...).
- "$*" Tous les arguments en une seule string
- PWD Le répertoire en cours

Variables builtin de Bash

- EDITOR: Editeur par défaut
- ENV: Script exécuté au démarrage
- HOME: Le homedir
- IFS: Le séparateur par défaut
- PATH: Le search path pour les commandes (A:A:B)
- PS1: Le prompt primaire ($ par défaut)
- TMOUT: Après N secondes sans interaction, le shell exit

Expressions arithmétiques

```
$ expr 1 + 1

2

$ myvar=$(expr 1 + 1)

echo $myvar

2

$ expr $myvar \* 3

6
```

Expressions arithmétiques

```
$ echo $myvar

3

$ echo $((myvar+2))

5
```

Job control

- Peu utile
- À creuser si vous voulez

Commandes builtin

- ! Négation d'un pipeline (if ! who |grep jho > /dev/null)
- # Commentaire
- : La commande nulle (dans un if qui à besoin d'une commande par exemple)
- #! "Shebang": Indique au système que ce n'est pas un binaire et donne l'interpréteur à utiliser
- . "Source": Lit et exécute le fichier (pas forcément un exécutable)
- [[ ]] "Test": La même chose que test

Commandes builtin

- **History** Agir sur l'historique (afficher, clear, …)
- **Kill** Ex: kill -9 -1 : Envoyer le signal 9 à PID 1
- **Pwd** Print working dir
- **Test** Tester une condition
- **Time** Time l'exécution d'une commande

Commandes builtin

- **Alias:** ex: alias ll=”ls -al”
- **Break:** Break une boucle
- **Cd** Changedir
- **Continue** Continue une boucle
- **Declare** Déclaration de variables complexes
- **Do/done** Début / fin de boucle
- **Echo** Ecrit sur stdout
- **True** Retourne 0
- **False** Retourne non 0

Case

```
case value in
 pattern1) cmds1;;
 pattern2) cmds2;;
 .
 .
 .
esac
```

```
while :         # Null command; always true
do
   printf "Type . to finish ==> "
   read line
   case "$line" in
      .) echo "Message done"
         break ;;
      *) echo "$line" >> $message ;;
   esac
done
```

For / while

```
for x [in list]
do
 commands
done
```

```
while condition
do
 commands
done
```

```
for item in `cat program_list`
do
     echo "Checking chapters for"
     echo "references to program $item..."
     grep -c "$item.[co]" chap*
done
```

If

```
if condition1
then commands1
[ elif condition2
then commands2 ]
 .
 .
 .
[ else commands3 ]
fi
```

```
if [ $counter -lt 10 ]
then number=0$counter
else number=$counter
fi
```

Test

while test $# -gt 0 Tant qu'il y à des args...

while [-n "$1"] Tant que l'argument n'est pas vide

if [$count -lt 10] Si cout < 10

if [-d REP] Si le répertoire REP existe

if ["$answer" != "YES"] Si la réponse n'est pas YES

if [! -r "$1" -o ! -f "$1"] Si le 1er argument n'est pas un fichier lisible ni un fichier régulier

Lecture ligne à ligne

```
while read line; do

    # $line contient la string de la ligne

done < filename
```

Code de retour

```
Exit 0   # All good

Exit 1   # Problem

...

Exit 255 # Problem
```