SCRIPTS AVEC BASH

Guillaume Chanel

Remerciements à Jean-Luc Falcone

Septembre 2019

SCRIPTS

PROBLÈME RÉEL

- Aggréger des listes d'emails
- Supprimer les doublons
- Les adresses peuvent être écrites avec des casses différentes

SOLUTION POSSIBLE

A l'aide des commandes shell suivante:

cat concatène plusieurs fichiers

tr transforme les caractères

sort trie les lignes

uniq rend unique les lignes répétées

```
$ cat mail1.txt mail2.txt | tr '[A-Z]' '[a-z]' | sort | uniq > result.txt
```

LES SCRIPTS

- Un script est un fichier texte contenant une séquence de commandes shell
- Le script peut être exécuté comme une commande

PREMIER SCRIPT

backup1.sh

```
#Efface les anciens backups
rm -f ~/data/backup/*

#Backup les fichiers .txt et .dat
cp ~/data/*.txt ~/data/*.dat ~/data/backup/
chmod 440 ~/data/backup/*
```

EXÉCUTER UN SCRIPT (`source`)

La commande `source` permet d'éxécuter un script **dans le shell actuel**

```
$ source backup.sh
```

EXÉCUTER UN SCRIPT (#!)

On peut préciser l'interpréteur par défaut du script avec les caractères #! au début du script (prononcer *hash-bang*, *she-bang* ou *sha-bang*):

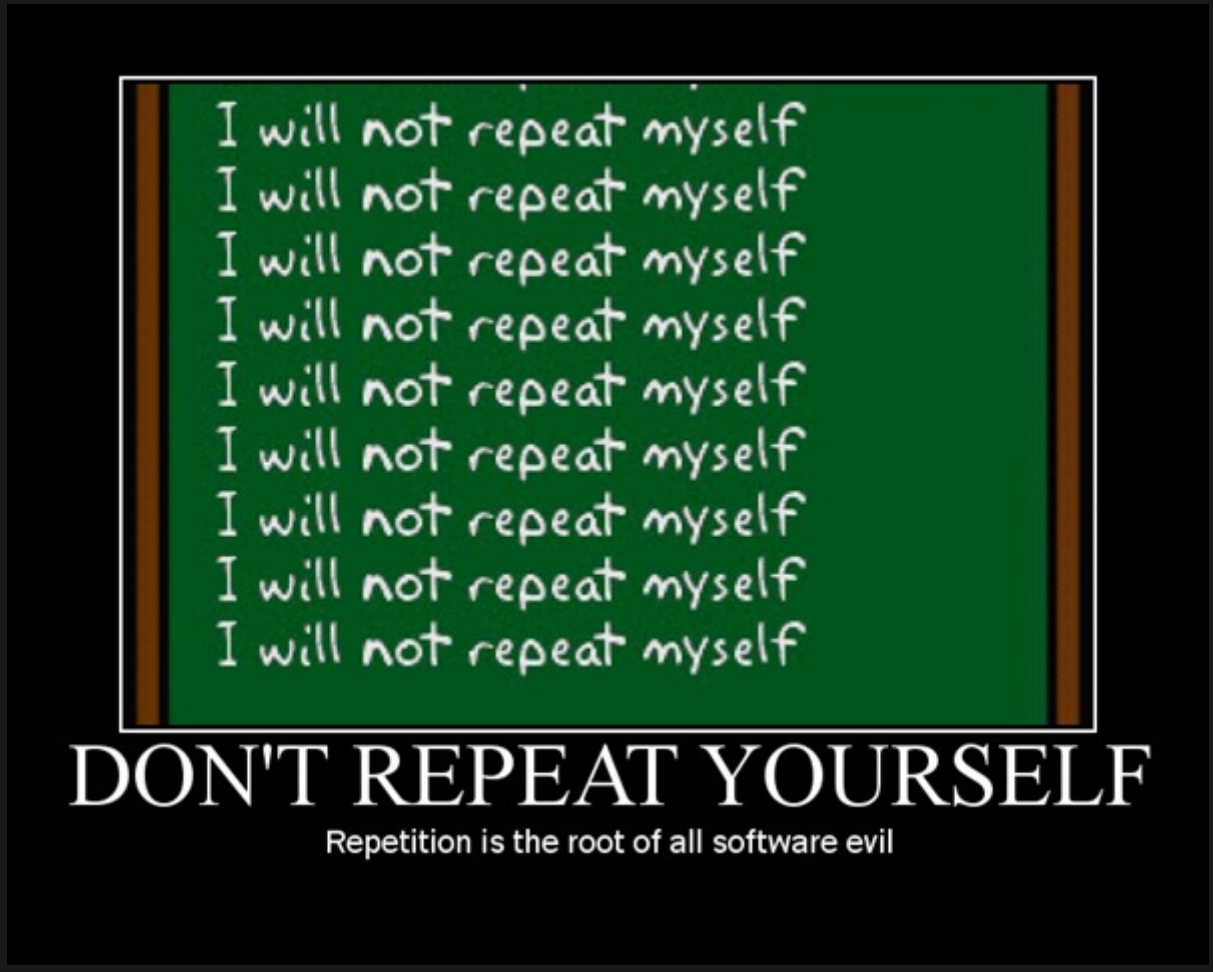
backup1.sh

```
#!/bin/bash

#Efface les anciens backups
rm -f ~/data/backup/*

#Backup les fichiers .txt et .dat
cp ~/data/*.txt ~/data/*.dat ~/data/backup/
chmod 440 ~/data/backup/*
```

Exécuter backup1.sh **dans un nouveau processus**:

```
$ chmod u+x backup1.sh
$ ./backup1.sh
```

DRY


I will not repeat myself
I will not repeat myself
I will not repeat myself
I will not repeat myself
I will not repeat myself
I will not repeat myself
I will not repeat myself
I will not repeat myself
I will not repeat myself
DON'T REPEAT YOURSELF
Repetition is the root of all software evil

VARIABLES

On assigne des variables avec le symbole =.

On accède à la valeur de la variable avec le symbole $:

```
$ message="Je suis ... "
$ echo message
$ echo $message
$ echo ${message}
```

VARIABLES

On assigne des variables avec le symbole =.

On accède à la valeur de la variable avec le symbole $:

```
$ message="Je suis ... "
$ echo message
$ echo $message
$ echo ${message}
$ echo "${message}Guillaume"
```

EXEMPLE DE SCRIPT

backup2.sh

```
#!/bin/bash

DATA=~/data
BACKUP=$DATA/backup

#Efface les anciens backups
rm -f $BACKUP/*

#Backup les fichiers .txt et .dat
cp $DATA/*.txt $DATA/*.dat $BACKUP
chmod 440 $BACKUP/*
```

CONDITIONS (`if then`)

Il existe des structures de contrôle `if` comme dans la plupart des languages:

```
if [ EXPRESSION ]; then
    cmd1
    cmd2
    ...
fi
```

CONDITIONS(if then else)

```
if [ EXPRESSION ]; then
    cmd1
    cmd2
    ...
else
    cmd10
    cmd11
    ...
fi
```

LES TESTS

On peut tester une expression avec la commande `test` généralement utilisée avec des crochets:

```
test EXPRESSION
[ EXPRESSION ]
```

TESTS SUR LES FICHIERS

-e FILE	Le fichier `FILE` existe
-f FILE	Le fichier `FILE` existe et est un "vrai" fichier
-d FILE	Le fichier `FILE` existe et est un répertoire
-x FILE	Le fichier `FILE` existe et est éxécutable
-w FILE	Le fichier `FILE` existe et on peut écrire
FILE1 -nt FILE2	Le fichier `FILE1` est plus récent que le fichier `FILE2`
FILE1 -ot FILE2	Le fichier `FILE1` est plus ancien que le fichier `FILE2`

EXEMPLE DE SCRIPT

backup3.sh

```
...
DATA=~/data
BACKUP=$DATA/backup

#Si le repertoire existe
if [ -d $BACKUP ]; then
    #Efface les anciens backups
    rm -f $BACKUP/*
else
    #Crée le repetoire
    mkdir -p $BACKUP
fi
...
```

AUTRE EXPRESSIONS DE TEST

! EXPRESSION	Retourne vrai si `EXPRESSION` est fausse
EXPR1 -a EXPR2	Retourne vrai si `EXPR1` **et** `EXPR2` sont vraies
EXPR1 -o EXPR2	Retourne vrai si `EXPR1` **ou** `EXPR2` sont vraies
STRING1 = STRING2	Retourne vrai si `STRING1` égale à `STRING2`
STRING1 != STRING2	Retourne vrai si `STRING1` n'est pas égale à `STRING2`

BOUCLES (`for`)

- Itère sur chaque éléments d'une liste.
- Les éléments sont séparés par des espaces ou des retours à la ligne
- Assigne une variable utilisable à l'intérieur de la boucle

```
for VAR in LIST; do
    CMD1 $VAR
    CMD2 $VAR
    ...
done
```

Une liste peut être:

- des éléments séparés par des espaces: `for ANIMAL in chat chien oiseau`
- un range: `for I in {1..9}`
- une expression représentant un ensemble de fichiers: `for FILE in *.txt`

EXEMPLE DE SCRIPT

backup4.sh

```
...
#Backup les fichiers .txt et .dat
for f in $DATA/*.txt $DATA/*.dat; do
    echo "Copie de $f"
    cp $f $BACKUP
    chmod 440 $f
done
...
```

SUBSTITUTION DE COMMANDE

- Il est souvent intéressant de capturer le résultat d'une commande.
- Il existe deux formes de syntaxes:
  - `` `CMD OPTIONS ARGUMENTS` ``
  - `$( CMD OPTIONS ARGUMENTS )`

```
$ echo "Mon nom d'utilisateur est: `whoami`"
$ echo "Mon nom d'utilisateur est: $(whoami)"
$ a=`ls *.sh`
$ a=$(ls *.sh)
```

ARGUMENTS

Il est possible de récupérer les arguments passés à un script en utilisant, les variables $1, $2, $3, etc.

backup5.sh

```
#!/bin/bash
DATA=$1
BACKUP=$2
...
```

Usage:

```
$ ./backup5.sh ~/data ~/data/backup
```

FONCTIONS

Les fonctions se définissent de la manière suivante:

```
function show {
echo "$1 ------ $2"  # $1 et $2 sont les 1er et 2ème paramètres passés à la fonction
}
show foo bar  # appel à la fonction
```

Remarques

On peut accéder aux arguments des fonctions en utilisant: $1, $2, etc. Les arguments du script sont cachés.

FONCTION ET `case`

```
function extract() {
    if [ -f $1 ] ; then
        case $1 in
            *.tar.gz) tar xvzf $1  ;;
            *.bz2)    bunzip2 $1   ;;
            *.rar)    unrar x $1   ;;
            *.gz)     gunzip $1    ;;
            *.tgz)    tar xvzf $1  ;;
            *.zip)    unzip $1     ;;
            *)        echo "'$1' cannot be extracted" ;;
        esac
    else
        echo "'$1' is not a valid file"
    fi
}
```

AUTRES SUJETS

- Opérations arithmétiques
- Tableaux
- Variables définies par défaut
- Règles pour les guillemets (*quotation*)

Bash beginners guide

Ryan's bash scripting tutorial

CONFIGURER BASH

FICHIERS DE CONFIGURATIONS

Bash utilise plusieurs fichiers cachés à la racine du home pour obtenir une configuration:

`.bashrc`	exécuté à chaque nouveau shell (non-login)
`.bash_profile`	shell de login uniquement.
`.bash_logout`	exécuté lorsqu'on quitte un shell de login

SHELL DE LOGIN (`.bash_profile`)

- Les shells de logins sont ceux pour lequels on doit s'authentifier:
  - Connexion à distance
  - Console virtuelle (`tty`)
- Un terminal lancé depuis l'interface graphique n'est pas un shell de login
- La pratique actuelle est d'appeler `.bashrc` depuis `.bash_profile` (automatique sous Mac).

.bash_profile

```
...
if [ -f ~/.bashrc]; then
    source ~/.bashrc
fi
...
```

CONFIGURATION DU SHELL (`.bashrc`)

Habituellement, on utilise `.bashrc` pour définir les éléments suivants:

- Variables d'environement
- Alias
- Permisissions par défaut
- Comportement de Bash

VARIABLES D'ENVIRONEMENT (`export`)

- Variables accessibles aux processus
- Tous les langages de programmation (sérieux) permettent d'y accéder.
- Les processus enfants héritent des variables d'environement du parent.
- On accède à leur valeur avec un $ (comme une variable normale).
- On les définit avec `export`:

```
$ export VAR1="hello world"
$ echo $VAR1
```

VARIABLES D'ENVIRONEMENT STANDARD

$PATH	chemin d'accès
$HOME	répertoire Home
$LANG	Language par défaut
$DISPLAY	Serveur graphique
$PS1	Prompt
$HOSTNAME	Nom de la machine

LE CHEMIN PAR DÉFAUT

- Le chemin d'accès indique où chercher les exécutables.
- La variable d'environement $PATH définit le chemin par défaut.
- Si l'éxécutable n'est pas dans le chemin par défaut, il faut préciser le chemin complet.

Exemple de chemin par défaut

```
/home/falcone/bin:/usr/lib/lightdm/lightdm:
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin:/usr/games
```

Dans quel fichier ? .bashrc ou .bash_profile ?

```
export PATH=~/bin:$PATH
```

LES ALIAS

- On peut définir des *alias* pour raccourcir l'invocation de commandes et de leurs arguments
- On peut accéder à la commande originale en utilisant un backslash.
- La commande `alias` seule permet d'afficher tous les alias.

Dans .bashrc

```
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
alias ls='ls --color=auto'
alias ll='ls -lh'
alias la='ll -a'
```

PERMISSIONS PAR DÉFAUT (`umask`)

- Il est possible de spécifier les permissions par défaut grâce à la commande `umask`.
- Syntaxe: `umask [permissions]`
  - Les permissions sont les permissions **à enlever** en octal.
- Utilisée seule, `umask` donne sa valeur actuelle.

Exemple

Par défaut on veut:

- retirer au groupe le droit d'écrire
- retirer aux autres tous les droits

```
umask 027
```

COMMANDES UTILES

awk '{print $n}'	selectioner la *n* ème colonne.
basename <FILE>	récupère le nom du fichier (sans le chemin)
dirname <FILE>	récupère le chemin d'un fichier
sed s/JPEG/jpg/	remplace 'JPEG' par 'jpeg'
uname -n	retourne le nom de la machine
cut -c m-n	extrait les caractères de *m* à *n* de chaque ligne
sed '1d'	ommet la première ligne

RESSOURCES

1. Livres
    - Unix Power Tools, 2002, O'Reilly.
2. Liens
    - http://linuxconfig.org/Bash_scripting_Tutorial
    - http://tldp.org/LDP/abs/html/abs-guide.html
    - http://www.ibm.com/developerworks/library/l-bash-test/index.html
    - https://ryanstutorials.net/bash-scripting-tutorial/