

Interpréteur de commandes – shell

Philippe Quéinnec

14 mai 2019

Plan

- 1 Interpréteur de commandes
 - read-eval(-print)
 - Interprétation
- 2 Commandes simples
 - Métacaractères
 - Redirections
 - Filtres / Couplage par tube
 - Parcours récursif
- 3 Langage de script
 - Variables
 - Quotes
 - Composition de commandes
 - Structures de contrôle

Rôle d'un interpréteur

Shell = programme d'interface interactive entre le système d'exploitation et un utilisateur.

```
Tant que vivant faire
  lire une commande
  interpréter la commande
  exécuter la commande
fintq
```

Les commandes sont :

- quelques commandes internes ;
- n'importe quel programme exécutable.

Programmation à gros grain (*in the large*)

Interpréteur interactif et de script

Aide à l'interaction

- métacaractères (`rm *.c`)
- historique des commandes passées
- complétion
- alias (raccourci)
- personnalisation de l'environnement de travail

Script

Fichier contenant un enchaînement de commandes

Interprétation d'une commande

Exécution de `ls -l *.c >/tmp/toto`

- 1 Découpage en mots (séparés par des espaces)
- 2 Décomposition en commandes élémentaires / composées
- 3 Expansions diverses (variables, sous-commandes, métacaractères)
`*.c` remplacé par `foo.c bar.c`
- 4 Re-découpage en mots → commande (1^{er} mot) et arguments
`ls • -l • foo.c • bar.c`
- 5 Création d'un processus fils
`if (fork()==0) { /*fils*/ } else { wait(...); }`
- 6 Gestion des redirections dans le fils
`fd = open("/tmp/toto"); dup2(fd,1);`
- 7 Exécution de la commande avec ses arguments (dans le fils)
`exec("ls",...);`



Plan

- 1 Interpréteur de commandes
 - read-eval(-print)
 - Interprétation
- 2 **Commandes simples**
 - Métacaractères
 - Redirections
 - Filtres / Couplage par tube
 - Parcours récursif
- 3 Langage de script
 - Variables
 - Quotes
 - Composition de commandes
 - Structures de contrôle

LA commande essentielle : man

`man sujet` ou `man -a sujet` (-a = all) ou `man -k motclef`

NOM

`ls` - Afficher le contenu de répertoires

SYNOPSIS

`ls [OPTION] ... [FICHIER] ...`

DESCRIPTION

Afficher les informations des FICHIERS (du répertoire courant par défaut). Les entrées sont triées alphabétiquement si aucune des options `-cftuvSUX` ou `--sort` n'est indiquée.

Les paramètres obligatoires pour les options de forme longue le sont aussi pour les options de forme courte.

`-a, --all`

inclure les entrées débutant par `<< . >>`



Métacaractères du shell

Expressions régulières évaluées sur l'arborescence des fichiers.

- * une suite quelconque de caractères
- ? un caractère quelconque
- [abc] un caractère parmi l'ensemble
- [0-9] un caractère dans l'intervalle
- [^a-f] un caractère dans le complément

Exemple : `rm *.old/*.c`

Redirections

Un processus créé par le shell dispose de 3 flots prédéfinis : entrée standard (0), sortie standard (1), sortie d'erreur standard (2).

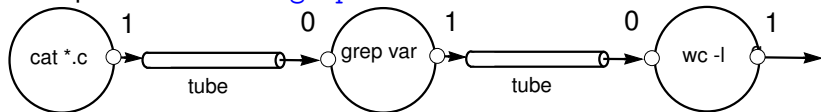
- `< fichier` redirige l'entrée standard depuis le fichier
- `> fichier` redirige la sortie standard vers le fichier en écrasement
- `>> fichier` redirige la sortie standard vers le fichier en ajout
- `n>&m` redirige le descripteur `n` vers `m`
- `2>&1` redirige la sortie d'erreur vers la sortie standard

Exemple : `cc toto.c >/tmp/err 2>&1`

Couplage par tube

C1 | C2 exécute en parallèle C1 et C2 après avoir créé un tube (pipe), reliant la sortie standard de C1 à l'entrée standard de C2.

Exemple : `cat *.c | grep var | wc -l`



Concatène les fichiers .c du répertoire courant (commande cat), sélectionne uniquement les lignes contenant la chaîne var (commande grep), et affiche le nombre de telles lignes (commande wc).

Commandes utiles pour le filtrage

La majorité des commandes unix lisent depuis l'entrée standard et écrivent sur la sortie standard si aucun fichier n'est spécifié.

<code>cat f1 f2...</code>	concatène les fichiers sur la sortie standard
<code>grep motif</code>	filtre les lignes contenant le motif
<code>cut</code>	extraction de colonnes : <code>cut -d, -f3-7</code>
<code>head -20</code>	afficher les 20 premières lignes
<code>tail -30</code>	afficher les 30 dernières lignes
<code>wc</code>	compter les caractères / mots / lignes
<code>tr</code>	convertir / éliminer des caractères : <code>tr '[0-9]' ' '</code>
<code>sort</code>	trier les lignes d'un fichier
<code>sed</code>	éditeur de flot

Exemple

Trouver le numéro de la ligne contenant le plus de a :

```
tr -d -c 'a\n' <fic | cat -n | sort -k2 \  
| tail -1 | cut -f1
```

- 1 tr élimine (-d) tous les caractères qui ne sont pas (-c) un 'a' ou une fin de ligne
- 2 cat -n numérote les lignes : 1 aaa / 2 aa / 3 aaaa
- 3 sort trie alphabétiquement sur le deuxième champ (-k2)
- 4 tail -1 garde la dernière ligne (donc la plus longue)
- 5 cut garde uniquement le premier champ (le numéro de la ligne)

Parcours récursif : find

`find répertoire expression1 expression2...`

- Parcourt récursivement le répertoire fourni en paramètre, et évalue les expressions sur chacun des fichiers rencontrés.
- Arrête l'évaluation pour le fichier courant dès qu'une expression est fausse.
- L'évaluation de l'expression peut comporter l'exécution de commandes portant sur le fichier courant.

```
find . -name '*.c' -size -200c -ctime +10 \  
    -exec rm -f '{}' ';' -print
```

- Recherche les fichiers à partir du répertoire courant (.)
- dont le nom correspond à *.c,
- de taille inférieure à 200 caractères,
- changé il y a plus de 10 jours,
- pour lesquels l'exécution de la commande rm réussit,
- et affiche leur nom.

find : expressions

- `-name motif` : nom de base (sans les répertoires) correspond au motif
- `-size [+|-]n[ckMG]` : taille supérieure/inférieure à n en octets/ko/Mo/Go
- `-type [dfl]` : type répertoire / fichier régulier / lien symbolique / ldots
- `-ctime [+|-]n` : changé depuis plus/moins de n jours
- `-newer fichier` : plus récent que fichier
- `-print` : affiche le nom du fichier
- `-ls` : affichage long des informations du fichier
- `-exec cmde ';' :` exécute la commande (en substituant '{}' par le nom du fichier)
- `-ok cmde \;` : idem en demandant confirmation

Plan

- 1 Interpréteur de commandes
 - read-eval(-print)
 - Interprétation
- 2 Commandes simples
 - Métacaractères
 - Redirections
 - Filtres / Couplage par tube
 - Parcours récursif
- 3 Langage de script
 - Variables
 - Quotes
 - Composition de commandes
 - Structures de contrôle

Script

```
#!/bin/sh -e
# La ligne précédente spécifie l'interpréteur de commande
# utilisé pour interpréter ce script.
# On peut spécifier des options (-e = terminaison dès
# qu'une commande renvoie un code non nul)

# Range le premier argument du script dans la variable x
x=$1
# Change le répertoire courant
cd $x
# Affiche le nombre de caractères du nouveau
# répertoire courant
pwd | wc -c
```



Variables

- Affectation :
`var=valeur`
(pas d'espace autour du =)
- Obtention de la valeur :
`$var`
- Arguments du script :
`$1, $2, ..., $*` (tous)
- Variables prédéfinies :
`$HOME, $PATH ...`
- Par défaut, variables non héritées par les processus créés par le script. Mais :
`export var`
déclare une *variable d'environnement*, héritée par les processus fils.



Quotes

- `\caractère` : supprime la nature spéciale du caractère
(*? \$ > ; _ () ...)
- `"chaîne $v *.c"` : avec interprétation des \$ et des '...',
sans remplacement des autres caractères spéciaux
- `'chaîne $v *.c'` : aucune interprétation des caractères
spéciaux
- `'cmde arguments'` : exécute la commande et remplace la
chaîne par la sortie de la commande

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

```
# Récupère dans x le début du répertoire courant:
```

```
# /home/queinnec/Cours/SC/slides => /home/queinnec/Cours
```

```
x='pwd | cut -d/ -f1-4'
```

```
# et cherche tous les fichiers .c à partir de là
```

```
find $x -name '*.c' -print
```



Composition de commandes

Code de sortie

Tout programme unix termine avec un code de succès/erreur dans 0..255 : la valeur de `exit(n)`.

Convention : 0 = ok / non-0 = erreur

- séquence : `C1; C2; C3`
- alternative : `C1 || C2 || C3` : exécute en séquence C1, C2, C3 jusqu'à ce qu'une commande ait un retour normal
- séquence et : `C1 && C2 && C3` : exécute en séquence C1, C2, C3 jusqu'à ce qu'une commande ait un retour anormal
- sous-processus : `(cmde)` : exécute cmde dans un sous-processus

Commande `test` ou `[`

`test expression` ou `[expression]`

- `-f fichier` : fichier existe
- `-x fichier` : fichier existe et est exécutable
- `-z chaîne` : la chaîne est de longueur nulle
- `-n chaîne` : la chaîne est de longueur non-nulle
- `ch1 = ch2` : les chaînes sont égales
- `ch1 != ch2` : les chaînes sont différentes
- `ch1 -eq ch2` : les nombres sont égaux
- `ch1 -ne ch2` : les nombres sont différents
- `ch1 -gt ch2` : le premier nombre est plus grand (greater than) que le second

```
[ $v = "toto" -o \( $v -gt 20 -a $v -lt 30 \) ]
```



Répétition for

```
#!/bin/sh
# $* : les arguments du script
for fic in $*; do
    echo $fic
    rm $fic
done
```

```
for i in 1 2 3 4; do
    echo bonjour $i
done
```

Répétition while

```
#!/bin/sh
# Tant que le premier argument est non vide
# et correspond à un fichier existant
while [ -n "$1" -a -f "$1" ]; do
    # l'affiche
    echo $1
    # et décale les arguments à gauche
    shift
done
```

Choix if

```
#!/bin/sh
if [ -f "$1" ]; then
    if rm "$1"; then
        echo "Le fichier $1 a été détruit"
    else
        echo "La destruction de $1 a échoué"
    fi
elif [ -d "$1" ]; then
    echo "$1 est un répertoire"
fi
```

Évaluation d'expression expr

expr **affiche** (sur sa sortie standard) le résultat de l'évaluation de l'expression fournie.

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

```
n=0
```

```
while [ $n -lt 10 ]; do
```

```
    echo $n; n='expr $n + 1'
```

```
done
```

```
# Regular expression matching
```

```
# Renvoie le dernier composant du répertoire courant:
```

```
# Expression régulière :
```

```
# . = un caractère quelconque, * = répétition
```

```
# [^/] = caractère quelconque sauf /
```

```
# \(...\) = capture affichée par expr
```

```
dir='pwd'
```

```
lastcomp='expr "$dir" : '.*\/\([^\/]*\)'
```