Bases de l'Architecture et des Systèmes (BAS)
Partie Systèmes

Eléments de shell de Bourne

Expressions régulières Scripts

Variables et expressions

Structures de contrôle

Équipe pédagogique Système, Univ. Toulouse

Références :

- Cours de Avi Silberschatz, Peter Baer Galvin et Greg Gagne, site `https://codex.cs.yale.edu/avi/os-book/OS10/slide-dir/index.html`
- Livre : « Operating System Concepts Tenth Edition » Avi Silberschatz, Peter Baer Galvin and Greg Gagne, John Wiley & Sons, Inc. ISBN 978-1-118-06333-0

Sommaire

Expressions régulières

Rôle et définition

- Moyen de définir des modèles (pattern) de lignes de texte transmises à des commandes
- Expression régulière
  - Chaîne de caractères à laquelle une ligne de texte peut correspondre ou non
  - Composée de
    - caractères normaux
    - caractères spéciaux = « métacaractères des expressions régulières » ($\neq$ métacaractères du shell)
- Non spécifiques au shell de Bourne
  - Très utilisées pour l'écriture de scripts shell
  - Souvent au travers de commandes comme `expr` et surtout `grep`

Principaux métacaractères des expressions régulières I

- `.` : caractère quelconque
  - Exemple : `..` $\longrightarrow$ modèle d'une ligne formée de 2 caractères quelconques (comme `aa`, `bc`, `45`, `6t`, etc.)
- `*` : répétition, du caractère qui précède, de longueur quelconque éventuellement vide
  - Exemple : `a*` $\longrightarrow$ modèle d'une ligne formée d'un nombre quelconque de `a` (y compris 0)
- `^` : début d'une ligne
  - Exemple : `^a.*` $\longrightarrow$ modèle d'une ligne commençant par le caractère `a`
- `$` : fin d'une ligne
  - Exemple : `.*a$` $\longrightarrow$ modèle d'une ligne se terminant par le caractère `a`
- `\` : protège le caractère suivant, qu'il soit caractère normal ou métacaractère des expressions régulières
  - Exemple : `a\$b` $\longrightarrow$ modèle d'une ligne formée des 3 caractères `a$b`

Principaux métacaractères des expressions régulières II

- `\{n\}` : répétition de `n` fois le caractère précédent, qu'il soit caractère normal ou métacaractère des expressions régulières
  - Exemple : `a\{3\}` $\longrightarrow$ modèle d'une ligne formée de 3 caractères `a`
- `[liste_caractères]` : caractères de la liste, définis par énumération ou par intervalle
  - Exemple : `^[a1b2]` $\longrightarrow$ modèle d'une ligne contenant un caractère `a` ou `b` ou `1` ou `2`

Expressions régulières

Écriture de la chaîne

- Entre apostrophes (recommandé)
  - Les caractères `$`, `*`, `\`, etc. ne seront pas interprétés comme des métacaractères du shell
  - Exemple : `'*a ?'`
- Seule ou entre guillemets
  - Certains métacaractères du shell seront interprétés
  - Exemple : `$var` ou `"$var"`

Script

Définition & Exécution

- Ensemble de commandes Unix rassemblées dans un fichier
  - Commentaires : ce qui suit `#`
- Fichier (le plus souvent) d'extension `.sh`
  - Exemple : `./nom_script.sh`
- Fichier Lancement d'un script
  - À l'aide de la commande `sh` (voir sa description)
    - `$ sh nom_script.sh [paramètres_du_script . . . ]`
  - En rendant le script exécutable puis en l'exécutant comme une commande
    - `$ chmod u+x nom_script.sh`
    - `$ ./nom_script.sh [paramètres_du_script . . . ]`

Script

Shells père / fils

- Script = commande
  - Exécutée dans le shell « courant » = « Shell père »
- Lancer l'exécution d'un script $\longrightarrow$ lancer un nouveau shell
  - Dit « shell fils »
  - De même nature que le « shell père » (`csh`, `ksh`, `sh`...)
    - Possibilité de préciser le shell invoqué au début du script
    - Exemple : `#! /bin/sh`
  - Porte le nom du script lancé
  - Les commandes du script y seront interprétées

Script

Paramètres positionnels

- Dix paramètres positionnels : `$0`, `$1`, . . ., `$8` et `$9`
- `$0` : nom du shell
- `$1`, . . ., `$9` : pas de valeur a priori (chaîne vide)
- Attribuer des valeurs à ces paramètres
  - Utiliser la commande set
    - Exemple : `set alpha beta` $\longrightarrow$ `$1` vaudra « `alpha` » et `$2` vaudra « `beta` »
  - Lancer le script avec des paramètres : affectation selon l'ordre
    - Exemple : `./nom_script.sh 1 2 3` $\longrightarrow$ `$0` vaudra « `./nom_script.sh` », `$1` « `1` », `$2` « `2` » et `$3` « `3` »

Paramètres spéciaux

- Leur nom commence par le métacaractère $
- Leur valeur dépend exclusivement du contexte du shell
- $# : nombre de paramètres
- $* et $@ : chaîne de caractères composée par la liste des paramètres, séparés par des espaces
  - "$*" est remplacé par un seul mot
  - "$@" est remplacé par la liste des paramètres dans laquelle chaque paramètre est un mot différent
- $? : code de retour de la dernière commande exécutée dans le shell
  - Après l'exécution d'un script, c'est le code de retour de la dernière commande exécutée par ce script
- $$: numéro du processus du shell

Paramètres spéciaux

Exemple

- Soit le script `exemple.sh` de contenu

```
#! /bin/sh
echo "\$0=$0, \$1=$1, \$2=$2, \$3=$3, \$4=$4, \
\$5=$5, \$6=$6, \$7=$7, \$8=$8, \$9=$9"
echo "\$#=$#, \$*=$*, \$?=$?, \$\$=$$"
```

- L'exécution de la commande : `./exemple.sh toto titi tata` produit

```
$0=./exemple.sh, $1=toto, $2=titi, $3=tata, $4=,
$5=, $6=, $7=, $8=, $9=
$#=3, $*=toto titi tata, $?=0, $$=19681
```

Variables et expressions

- Tout shell permet de gérer des variables
- Nom d'une variable composé de lettres, chiffres et du caractère `_`
  - Ne peut commencer par un chiffre
  - Exemple : `var`, `ma_variable_13`, `temp31`, etc.
- Valeur obtenue en faisant précéder son nom du caractère `$`
  - Exemple : `$var` est la valeur de la variable `var`
- La valeur d'une variable est une chaîne de caractères
  - Exemple : `123` qui est la chaîne « `123` » ou `abc` qui est la chaîne « `abc` »

Variables

Affectation

- À l'aide de la commande `read`
- Exemple :

```
$ read var
hello
$ echo $var
hello
```

- En utilisant la syntaxe : variable=expression
  - Où expression est une expression du shell ayant comme valeur une chaîne de caractères
  - Exemple : `var=hello`
- Attention
  - Il n'y a **pas** d'espaces autour de =
  - La syntaxe dépend du shell (syntaxe différente en Cshell, par exemple)

Concaténation d'expressions

- S'écrit en juxtaposant des expressions qui peuvent être
  - une chaîne de caractères (exemple : `toto`)
  - la valeur d'un paramètre positionnel (exemple : `$2`)
  - la valeur d'une variable (exemple : `$var`)
  - la capture de la sortie standard d'une commande (exemple : `` `pwd` ``)
- En utilisant – si besoin est – les délimiteurs `"` et `'` $\longrightarrow$ tenir compte de leur influence sur l'interprétation des métacaractères du shell
- Exemple :
  `var=hello`
  `var="$var world !"` $\longrightarrow$ `var` vaudra « `hello world !` »

Variables prédéfinies du shell

- `SHELL` : nom du shell courant
- `PS1` : configuration de l'invite de commande (prompt)
- `HOSTNAME` : nom de la machine
- `USER` : nom de l'utilisateur connecté (équivalent à `LOGNAME`)
- `HOME` : répertoire d'accueil
- `PWD` : répertoire de travail courant
- `PATH` : liste de répertoires où rechercher les commandes
- ...
- La commande `env` permet d'afficher leur valeur

Variables vs Paramètres positionnels

- Deux concepts différents
- Pour modifier la valeur d'un paramètre
  - Appeler le script avec des paramètres
  - Commande `set`
- Pour modifier la valeur d'une variable
  - Affecter une valeur à une variable
  - Commande `read`

Variables

Accès à la valeur

- Faire précéder son nom de $
- Utiliser des accolades pour lever des ambiguïtés

Exemple 1

```
$ vari=impecc
$ variable=parfait
$ echo $variable
```

Variables

Accès à la valeur

- Faire précéder son nom de $
- Utiliser des accolades pour lever des ambiguïtés

Exemple 1

```
$ vari=impecc
$ variable=parfait
$ echo $variable
parfait
```

Variables

Accès à la valeur

- Faire précéder son nom de $
- Utiliser des accolades pour lever des ambiguïtés

Exemple 1

```
$ vari=impecc
$ variable=parfait
$ echo $variable
parfait
$ echo ${vari}able
```

Variables

Accès à la valeur

- Faire précéder son nom de $
- Utiliser des accolades pour lever des ambiguïtés

Exemple 1

```
$ vari=impecc
$ variable=parfait
$ echo $variable
parfait
$ echo ${vari}able
impeccable
$
```

Variables

Accès à la valeur

- Faire précéder son nom de $
- Utiliser des accolades pour lever des ambiguïtés

Exemple 1

```
$ vari=impecc
$ variable=parfait
$ echo $variable
parfait
$ echo ${vari}able
impeccable
$
```

Exemple 2

```
$ var=p
$ $varpwd
```

Variables

Accès à la valeur

- Faire précéder son nom de $
- Utiliser des accolades pour lever des ambiguïtés

Exemple 1

```
$ vari=impecc
$ variable=parfait
$ echo $variable
parfait
$ echo ${vari}able
impeccable
$
```

Exemple 2

```
$ var=p
$ $varpwd
```

La variable n'existe pas, sa valeur est donc la chaîne vide comme si l'utilisateur avait appuyé sur « entrée »

Variables

Accès à la valeur

- Faire précéder son nom de $
- Utiliser des accolades pour lever des ambiguïtés

Exemple 1

```
$ vari=impecc
$ variable=parfait
$ echo $variable
parfait
$ echo ${vari}able
impeccable
$
```

Exemple 2

```
$ var=p
$ $varpwd
$ ${var}wd
```

La variable n'existe pas, sa valeur est donc la chaîne vide comme si l'utilisateur avait appuyé sur « entrée »

Variables

Accès à la valeur

- Faire précéder son nom de $
- Utiliser des accolades pour lever des ambiguïtés

Exemple 1

```
$ vari=impecc
$ variable=parfait
$ echo $variable
parfait
$ echo ${vari}able
impeccable
$
```

Exemple 2

```
$ var=p
$ $varpwd
$ ${var}wd
/users/linfg/l1inf201
$
```

La variable n'existe pas, sa valeur est donc la chaîne vide comme si l'utilisateur avait appuyé sur « entrée »

Structures de contrôle

- Contrôlent l'ordre dans lequel les différentes commandes d'un script sont exécutées
- Sont de 3 types
  - Séquence : par défaut, les commandes s'exécutent l'une après l'autre dans l'ordre de leur écriture
  - Choix / Sélection / Alternative . . . : permet de choisir d'exécuter un bloc de commandes plutôt qu'un autre
  - Boucle / Répétition : permet de répéter l'exécution d'un bloc de commandes

Structures de contrôle

Conditions

- Un choix ou une répétition se base sur un test = une condition
- Une condition est réalisée par l'appel d'une commande retournant un code
  - Valant 0 $\longrightarrow$ la condition est « vraie »
  - Différent de 0 $\longrightarrow$ la condition est « fausse »
- La commande peut-être quelconque
- C'est souvent la commande `test` qui est utilisée

Structures de contrôle I

Choix

```
if condition1
then
   séquence_commandes1
elif condition2
then
   séquence_commandes2
else
   séquence_commandes3
fi
```

```
case expression in
cas1)
   séquence_commandes1
;;
cas2|cas3)
   séquence_commandes2
;;
*)
   séquence_commandes3
esac
```

- `cas1`, `cas2`, `cas3`, etc. sont des chaînes de caractères
  - Elles peuvent utiliser éventuellement les métacaractères du shell

Structures de contrôle II

Choix

- Le caractère | signifie «ou»
- le caractère * (métacaractère du shell) signifie «n'importe quelle chaîne de caractères ne commençant pas par .»
- Le double point virgule (;;) équivaut à la commande `break`

Structures de contrôle I

Répétitions

```
while condition
do
   séquence_commandes
done
```

```
until condition
do
   séquence_commandes
done
```

```
for variable in liste_de_cas
do
   séquence_commandes
done
```

Remarques :

- `liste_cas` est une liste de chaînes de caractères
  - Qui peut utiliser éventuellement les métacaractères du shell
- La commande `break` peut être utilisée

Structures de contrôle II

Répétitions

- La syntaxe

```
for i
do
...
```

équivaut à :

```
for i in "$@"
do
...
```

- $\longrightarrow$ la variable i prend successivement la valeur de chacun des paramètres positionnels ayant reçu une valeur