

Système d'exploitation

Florent Devin



Ecole Internationale des Sciences du Traitement de l'Information

Scripts

Présentation

- Automatise un processus manuel
- Enchaînement de plusieurs commandes Unix
- Peu d'interaction avec l'utilisateur
- Exécution via un interpréteur de commandes (shell)
- Différents types de shell \Leftrightarrow différents langages de script

Langages de script

- Plusieurs types de langages
- Écriture rapide
- Pas de compilation
- Commandes centrées sur les objets existants (commandes unix)
- Informations majoritairement textuelles ⇒ **typage faible**

Famille des scripts

- **sh** : SHell d'origine (le plus simple)
- **csh** : C-SHell proche de la syntaxe de C (ancien)
- **bsh** : Bourne SHell, propriété de Bell (mécanisme de tubes)

Famille des scripts

- **tcsh** : Tenex C SHell, version étendue du **csh** (complétion automatique, ed, manipulation des historiques)
- **ksh** : Korn SHell, fusion de **csh**, et de **bsh** (historique des commandes)
- **bash** : Bourne Again SHell, open source, reprend l'ensemble des avancées
- **zsh** : Zero SHell, fusion des shells précédent, très puissant
- d'autres existent +/- exotiques +/- utilisés (**ash**, **dash**, **esh**, **scsh**, ...)

Variables du shell

- Shell : environnement \Leftrightarrow utilisation de mémoire
- Possibilité de définir des variables
- Possibilité d'interagir avec les variables, et l'utilisateur
- Utilisation de variables d'environnement
 - Permet de spécifier un comportement par défaut

Résumé

- Shell un exécutable écrit en C qui interprète le Shell Script
- Shell Script : un langage donc une syntaxe de script
- Script : une forme de programme qui exécute directement son source
- Script Shell : un script (un fichier source) écrit en langage de script

Avantages / inconvénients

- Avantages

- Syntaxe hypercompacte très rapide à écrire
- Accès à toutes les commandes accessibles du système

- Défauts

- Règles syntaxiques très locales
- Variabilité des écritures suivant le shell utilisé
- Règles implicites dans la tokenisation qui exigent une rigueur absolue
- Écrire compact, c'est aussi rendre la lecture difficile

Variables

Déclaration

- Définition d'une variable :

```
maVariable="une_variable "  
maVariable2=54
```

- un nom, = et une valeur textuelle ou littérale
- Le nom doit être un nom de variable légal (règles du C)
- **PAS D'ESPACE AUTOUR DU EGAL**

Utilisation

```
echo $maVariable
```

une variable

```
echo $maVariable2
```

54

```
echo maVariable
```

maVariable

- Besoin de précéder le nom par \$ pour avoir la valeur
- echo : permet d'afficher ce qui suit

Substitution

- Substitution dans les chaînes de caractères

```
>$ maVariable="variable"
```

```
>$ echo "Voici une $maVariable"
```

Voici une variable

- Non substitution dans les suites de caractères

```
>$ echo 'Voici une $maVariable'
```

Voici une \$maVariable

- Stockage du résultat d'une commande dans une variable

```
maVariable='ls -l | grep -e /^d/'
```

Résumé

Trois types de chaînes

- Chaînes littérales : `'`
 - Chaînes substituées : `"`
 - Chaînes exécutées : ```
-
- `${toto}` : permet de borner le nom de la variable
 - Lève l'ambiguïté lors de l'interprétation

Manipulation de chaînes de caractères

Introduction

- Deux familles : concaténation, extraction
- Concaténation : ajout de chaînes
- Extraction : découpage et récupération de morceaux

Troncatures

- Deux commandes : `basename`, `dirname`
- `basename` : enlève le chemin ou l'extension d'un fichier
- `dirname` : garde le chemin

```
basename /etc/apache2/http.conf
```

```
http.conf
```

```
basename /etc/apache2/http.conf .conf
```

```
http
```

```
dirname /etc/apache2/http.conf
```

```
/etc/apache2/
```

Troncatures avancées

- $\${var\#\text{mod}}$: suppression de la plus courte sous chaîne à gauche
- $\${var\#\#\text{mod}}$: suppression de la plus longue sous chaîne à gauche
- $\${var\%\text{mod}}$: suppression de la plus courte sous chaîne à droite
- $\${var\%\%\text{mod}}$: suppression de la plus longue sous chaîne à droite
- $\${var:\text{ind}}$: supprime les caractères de 0 à ind d'une chaîne
- $\${var:\text{ind}:\text{nb}}$: extrait nb caractères à partir de ind

Troncatures avancées

- $\${\#var}$: donne la longueur de la chaîne
- $\${var/mod/}$: supprime la première occurrence de mod dans var
- $\${var//mod/}$: supprime toutes les occurrences de mod dans var
- $\${var/mod1/mod2}$: remplace la première occurrence de mod1 par mod2
- $\${var//mod1/mod2}$: remplace toutes les occurrences de mod1 par mod2

Attention

Syntaxes non combinables !!

Tests

Notation

- Deux notations :
 - `test expr`
 - `[expr]`
- Négation : `! expr`
- Conjonction : `expr1 -a expr2`
- Disjonction : `expr1 -o expr2`

Test sur les chaînes

- `s1 = s2` : test d'égalité
- `s1 != s2` : test de différence
- `-n s1` : chaîne non vide
- `-z s1` : chaîne vide

Test sur les nombres

- `n1 -eq n2` : =
- `n1 -ne n2` : ≠
- `n1 -lt n2` : <
- `n1 -le n2` : ≤
- `n1 -ge n2` : ≥
- `n1 -gt n2` : >

Test sur les fichiers

- `-e fic` : existence du fichier
- `-s fic` : existence d'un fichier non vide
- `-f fic` : existence du fichier ordinaire
- `-d fic` : existence du répertoire
- `-r fic` : existence du droit de lecture
- `-w fic` : existence du droit d'écriture
- `-x fic` : existence du droit d'exécution
- ...

Structures de contrôle

Conditionnelles

```
if cond
then
    ...
    ...
fi
```

Alternatives simples

```
if cond
then
    ...
    ...
else
    ...
    ...
fi
```

Alternatives multiples

```
if cond
then
    ...
    ...
elif cond
then
    ...
    ...
else
    ...
    ...
fi
```

Aiguillages multiples

```
case var in  
  cas1) ...  
        ...;;  
  cas2) ...  
        ...;;  
  *) ...  
    ...;;  
esac
```

boucle

```
for var in liste_valeur
do
    ...
    ...
done
```

Test en tête

```
while cond
do
    ...
    ...
done
```

Lecture d'un fichier ligne par ligne

```
while read var
do
    ...
done < fic
```

Divers

Divers

- `mktemp` : crée un fichier temporaire dans `/tmp` et renvoie le nom du fichier
- Expression arithmétique : `$((expr))`
- Récupération des arguments : `$0 $1, ...`
- Nombre de paramètres du script : `$#`
- Décalage des paramètres : `shift`
- Définition des variables : globales

Création de fonction

```
toto () {  
    ...  
    ...  
}
```

```
toto $a $b
```

Valeur de la fonction : dernière action effectuée

Enchaînement de commandes

Valeur de retour

- Programme : valeur de retour
- Possibilité de chaîner des commandes avec `&&` et `| |`