Linux – LTE 2 - ISSBA

Année universitaire 2007-2008

Linux
Shell / Traitement des données

Jean-Michel RICHER
Faculté des Sciences, H206

Plan

- Commandes de traitement des fichiers textes
  - cut, grep, tr, uniq, sort, sed

- Le Shell
  - Les variables
  - Les scripts

    If, for, while

Les commandes des fichiers textes

- Sous Unix/Linux il peut être intéressant de gérer des fichiers de données au format texte
  - C'est le cas des fichiers de configuration ainsi que de certains fichiers :

    Utilisateur /etc/passwd

- Les commandes qui permettent la gestion de fichiers sont les suivantes :
  - grep, cut, sort, uniq, tr, sed : traitement ligne à ligne
  - elles sont utilisées de concert avec le symbole | (pipe) qui permet de rediriger la sortie d'une commande vers l'entrée d'une autre commande

Commande grep

- grep permet de sélectionner les lignes d'un fichier texte qui contiennent un motif
  - **`grep [options] motif [fichier]`**
  - Options
    - -n affiche le numéro de ligne
    - -c compter le nombre de lignes
    - -v ne pas sélectionner les lignes qui correspondent au motif
    - -R considérer les sous-répertoires

Le motif grep

- Il s'agit d'une expression régulière
  - ^ début de ligne
  - $ fin de ligne
  - [0-9] tout chiffre
  - [^0-9] tout sauf un chiffre
  - ? l'élément précédent est facultatif
  - * l'élément précédent apparaît 0 ou n fois
  - + l'élément précédent apparaît 1 ou n fois
  - {n} l'élément précédent apparaît n fois
  - {n,m} l'élément précédent apparaît entre n et m fois

Commande grep

- Exemple :

  apparaître
  aperitif
  apprendre
  apostropher

```
grep  "ap\{1,2\}e\?r"
```

Commande cut

- Elle permet de découper une ligne de texte en champs

```
cut [options] fichier
```

  - Options
    - -d permet de spécifier le séparateur de champs
    - -f permet de spécifier les champs à récupérer

  - Exemple

```
cut –d ';' –f1,3
```

Commande cut

- Exemple

toto;maths;10
lagaffe;francais;6
rambo;maths;3

```
cut -d ';' -f1,3
```

toto;10
lagaffe;6
rambo;3

Commande sort

- Elle permet de trier les lignes d'un fichier

  sort [options] [fichier]

  - Options

    -r dans l'ordre inverse

    -g tri de valeur numériques

    Par défaut le tri est réalisé sur des valeurs alphanumériques

Commande uniq

- Elle permet de supprimer les lignes consécutives identiques
  - Exemple pour connaître le nombre d'utilisateurs différents connectés :

```
who | cut -d ' ' -f1 | sort | uniq | wc -l
```

Commande tr

- Elle permet de supprimer ou remplacer des caractères
  - Supprimer les retours chariot de windows

```
tr -d '\015' <file1.txt  >file2.txt
```

  - Modification, mettre en majuscules

```
cat notes.txt | tr 'a-z' 'A-Z'
```

Commande sed

- Elle permet de modifier des parties d'un fichier

```
cat notes.txt | sed -e "s/;/:/g"

cat notes.txt | sed -e "s/\([a-z]*\);\([a-
  z]*\);\([0-9]*\)/\2 \1 \3/g"
```

Le Shell

- il s'agit de l'interpréteur de commande qui dispose d'un langage de programmation (~basic)
  - bash (Bourne again Shell)
  - csh
  - ksh
  - sh (par défaut, écrit par S.R. Bourne)
  - tcsh (amélioration de csh)
  - zsh (le plus complet)

Les variables du Shell

- Le Shell possède un certain nombre de variables prédéfinies
  - HOME répertoire racine de l'utilisateur
  - USER nom de l'utilisateur
  - HOSTNAME nom de la machine
  - DISPLAY nom de l'écran graphique
  - PATH liste des répertoires des exécutables

- Pour les visualiser, taper :

```
setenv
```

- Pour définir une variable globale pour le Shell :

```
export DISPLAY=g205-1:0.0
```

Utilisation des variables

- Les variables sont non typées
- La déclaration est simple

```
v=1
v=“toto“
```

- Pour utiliser une variable (obtenir sa valeur), il faut la faire précéder du symbole $

```
echo $v
v=`expr $v + 1`
```

Les scripts Shell

- Les scripts sont des programmes écrits en Shell et que l'on peut exécuter depuis l'interpréteur de commande
- Ils doivent commencer par la ligne suivante qui indique qu'il s'agit d'un script et quel Shell il faut utiliser

`#!/bin/sh` ou **`#!/bin/bash`** …

Redirection des entrées/sorties

- les caractères suivants ont une signification particulière
  - > redirection de la sortie standard

```
ls –l > fichiers.txt
```

  - >> idem mais ajoute en fin de fichier
  - 2> redirection de la sortie des erreurs

```
commande.sh 2>/dev/null
```

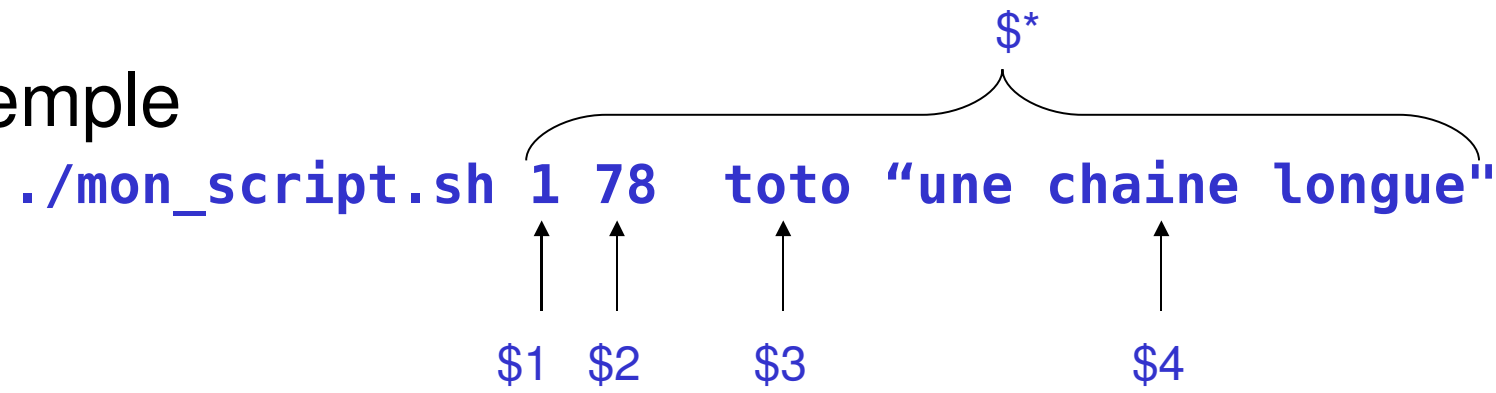
  - < redirection de l'entrée standard

Les scripts Shell

- Lors de l'appel (exécution) d'un script, on peut passer des arguments, les variables suivantes permettent de manipuler ces arguments

  - $# nombre d'arguments
  - $* l'ensemble des arguments
  - $1 $2 … $9 chaque argument pris séparément

- Exemple



La commande echo

- Elle permet l'affichage d'information

```
echo "bonjour "

echo "bonjour $nom “

echo $nom
```

La commande exit

- Elle permet d’arrêter le programme

```
exit 1
```

La conditionnelle if then else

- If-then-else

```
if condition ; then
  instructions
else
  instructions
fi
```

- Exemple : on sort du programme s'il n'y a pas d'arguments

```
if test $# –eq 0 ; then    (ou if [ $# -eq 0 ] then)
  echo “il manque des arguments"
fi
```

Les comparaisons avec if

- on peut comparer des chaines de caractères
  - chaine vide ? **`-z CHAINE`**
  - égalité **`CHAINE1==CHAINE2`**
  - différence **`CHAINE!= CHAINE2`**
- ou nes nombres
  - égalité **`-eq`**
  - différence **`-ne`**
  - inférieur, inférieur ou égal **`-lt`** **`-le`**
  - supérieur, supérieur ou égal **`-gt`** **`-ge`**
- **opérateurs booléens**
  - **et logique** **`-a`**
  - **ou logique** **`-o`**

La boucle for

- Permet d'énumérer des valeurs

```
for variable in  liste-de-valeurs ; do
  instructions
done
```

- Exemple : afficher les valeurs 1 à 5

```
for i in 1 2 3 4 5 ; do
  echo $i
done
```

La commande while

- permet d'exécuter des instructions tant qu'une condition est vérifiée

```
i=1
while [ $i –lt 10 ] ; do
  echo $i
  i=`expr $i + 1`
done
```