Shell

TD1 - Introduction

A. Krähenbühl

Telecom Nancy, Université de Lorraine

14 Avril 2014

Plan

Plan

Qu'est-ce que c'est ?

Qu'est-ce que c'est ?

Ensemble de programmes qui permettent :

- d'attendre des commandes, de les analyser et d'y répondre
- d'éditer des fichier
- de gérer les ressources : disques, processeur, périphériques, ...
- etc.

Exemples : linux, windows, Mac, etc.

Linux

Linux

- Système d'exploitation écrit en C
- Version libre de Unix conçue par Linus Torvalds

Pourquoi Linux et pas Windows ?

Pourquoi Linux et pas Windows ?

- Connaitre Linux c'est un peu les connaitre tous
- Code source libre
- Windows vu dans d'autres matières

Qu'est-ce qu'un interpréteur de commandes ?

Rôle principal d'un OS

Qu'est-ce qu'un interpréteur de commandes ?

Rôle principal d'un OS

Attendre et exécuter les commandes de l'utilisateur.

Qu'est-ce qu'un interpréteur de commandes ?

Rôle principal d'un OS

Attendre et exécuter les commandes de l'utilisateur.

Quel programme remplit ce rôle ?

Qu'est-ce qu'un interpréteur de commandes ?

Rôle principal d'un OS

Attendre et exécuter les commandes de l'utilisateur.

Quel programme remplit ce rôle ?

L'interpréteur de commande !

Qu'est-ce qu'un interpréteur de commandes ?

Rôle principal d'un OS

Attendre et exécuter les commandes de l'utilisateur.

Quel programme remplit ce rôle ?

L'interpréteur de commande !

Les commandes peuvent être données graphiquement ou en mode texte.

Le shell

Qu'est-ce qu'un **shell** ?

Le shell

Qu'est-ce qu'un **shell** ?

Un interpréteur de commandes textuel d'Unix

Le shell

Qu'est-ce qu'un **shell** ?

Un interpréteur de commandes textuel d'Unix

Pourquoi le nom de **shell** ?

Le shell

Qu'est-ce qu'un **shell** ?

Un interpréteur de commandes textuel d'Unix

Pourquoi le nom de **shell** ?

C'est une **coquille** autour du noyau de l'OS.

Le shell

Qu'est-ce qu'un **shell** ?

Un interpréteur de commandes textuel d'Unix

Pourquoi le nom de **shell** ?

C'est une **coquille** autour du noyau de l'OS.

Il existe plusieurs shells : Bash, dash, csh, ksh, zsh, ...

Analyse de l'exécution d'une commande

1. Attente d'une séquence de caractères terminée par Entrée
2. Analyse de la séquence : est-elle correcte ?
3. - Si oui, exécution de la commande et attente de la fin de la commande
   - Si non, affichage d'un message d'erreur
4. Retour à l'étape 1

Gestion des commandes

Gestion des commandes

- commande & : Lancer une commande en arrière-plan
- CTRL + Z : Suspendre une commande
- bg : Relencer la tâche suspendue en tache de fond
- fg : Relancer la tâche suspendue au premier plan
- CTRL+C : Tuer la tâche de fond

Différence programme/processus

Différence programme/processus

Un programme truc passif sur le disque dur

Un processus instance de programme en cours d'exécution

Syntaxe

```
<nom_commande> [<arguments>]
```

Syntaxe

```
<nom_commande> [<arguments>]
```

Exemples :

```
cp tp1.c tp2.c
mkdir TP
emacs un\_fichier
gcc tp1.c
ls
```

Syntaxe

```
<nom_commande> [<arguments>]
```

Exemples :

```
cp tp1.c tp2.c
mkdir TP
emacs un\_fichier
gcc tp1.c
ls
```

Exo : Que font ces commandes ?

Syntaxe

```
<nom_commande> [<arguments>]
```

Exemples :

```
cp tp1.c tp2.c
mkdir TP
emacs un\_fichier
gcc tp1.c
ls
```

Exo : Que font ces commandes ?

→ **RTFM** !

Types de commande

- Celles qui modifient qqch (*mkdir*) vs. celles qui ne modifient rien (*ls*)
- Celles qui retournent un résultat (*ls*) vs celles qui ne retournent rien (*rmdir*)

Plan

Qu'est-ce que c'est ?

Qu'est-ce que c'est ?

Deux types de fichiers exécutables :

Qu'est-ce que c'est ?

Deux types de fichiers exécutables :

- Les fichiers binaires
- Les fichiers de scripts

Qu'est-ce que c'est ?

Deux types de fichiers exécutables :

- Les fichiers binaires
- Les fichiers de scripts

On va s'intéresser aux **scripts shell**.

Exemple de script

```
#! /bin/bash

cd $HOME/Images
mkdir TP1
cd TP1
cp /home/adrien/client.conf .
echo le repertoire courant est
pwd
echo il contient
ls -l
```

Les arguments

Les arguments sont désignés à partir de leur position sur la ligne de commande :

- $0 : le nom du script
- $1 : le premier argument
- $2 : le deuxième argument
- ...
- $# : le nombre d'arguments

Exemple avec le script *affiche_args*.

Condition *if*

```
#! /bin/bash

if [ $1 -gt $2 ]
# attention aux espaces apres '[' et avant ']'
then
    echo $1 est plus grand que $2
else
    echo $1 n'est pas plus grand que $2
fi
```

Comparaisons d'entiers

Pour comparer deux entier :

- -gt (greater than, $>$)
- -ge (greater or equal, $>=$)
- -lt (less than, $<$)
- -le (less or equal, $<=$)
- -eq (equal, $=$)
- -ne (not equal, $=$)

Exemple :

```
#! /bin/bash

if [ $1 -gt $2 ]
then
    echo $1 est plus grand que $2
fi
```

Comparaisons de chaines

Pour comparer deux chaines de caractères :

- ==
- !=

Exemple :

```
#! /bin/bash

if [ "$1" == "$2" ]
then
    echo $1 est le meme mot que $2
fi
```

Comparaisons booléennes

Pour composer deux expressions booléennes :

- ! (not)
- -o (or)
- -a (and)

Exemple :

```
#! /bin/bash

if [ "$1" == "$2" -a "$2" == "$3" ]
then
    echo $1 est le meme mot que $3 : transitivite !
fi
```

Tests sur les fichiers

- -f : teste si un fichier est présent ou non (dans le répertoire courant)
- -d : teste si un repertoire est present ou non
- -x : teste si un fichier (du repertoire courant) est exécutbale
- -r : teste si un fichier (du repertoire courant) est acccessible en lecture
- -w : teste si un fichier (du repertoire courant) est acccessible en ecriture

Exemple :

```
#! /bin/bash

if [ -r "$1" ]
then
    echo Vous avez le doit de lire le fichier $1
fi
```

La boucle for

```
#! /bin/bash

for i in 1 2 3 4 5
do
    echo $i
done
# ou for (( i=1 ; i<=5 ; i++ ))
# ou for i in {0..5}
# ou for i in {0..5..1}

# Pour les fichiers :
for file in *.tex
do
    echo $file
done
# ou for file in `ls *.tex`
```

La boucle while

```
#! /bin/bash

i=0
while [ $i -lt 5 ]
do
     echo $i
     i=$(( i + 1 ))
     # ou i=`expr $i + 1`
     # ou (( i++ ))
done
```

Le switch

```
#! /bin/bash

case "$1" in
    "vendredi") echo "C'est presque bon" ;;
    "samedi") echo "Enfin le week-end !" ;;
    "dimanche") echo "Deja la fin :(" ;;
    *) echo "Vivement le week-end...";;
esac
```

Rediriger les flux

La redirection de flux consiste à diriger la **sortie standard** (l'écran) et/ou l'**entrée standard** (le clavier) vers un **fichier** :

- *cmd* > *fich* : redirige la sortie de la commande vers un fichier qui est écrasé
- *cmd* >> *fich* : redirige la sortie de la commande vers un fichier par concaténation
- *cmd* < *fich* : redirige l'entrée standard en argument de ma commande

```
#! /bin/bash

echo "Je suis la premiere ligne" > fich.txt
echo "Je suis la deuxieme ligne" >> fich.txt
echo "J'ecrase les deux autres lignes" > fich.txt
```

Enchainer les commandes

On peut se servir de **la sortie** d'une commande **comme entrée de la suivante** et former ainsi un **tube** de commandes :

- *cmd*1|*cmd*2 : la sortie de *cmd*1 sert d'entrée à *cmd*2

C'est équivalent à faire *cmd*1 $>$ *fich*; *cmd*2 $<$ *fich*.

```
#! /bin/bash

if [ ! -d "$1" ]
then
    nbFichiers=‘ls $1 | wc -l‘
    echo il y a $nbFichiers fichiers dans le repertoire $1
fi
```

Tableau blanc non interactif