Administration GNU/Linux
TP1 : Programmation shell

Nathaël Pajani
&
David Odin

2016

Objectifs :
— Se familiariser avec un système GNU/Linux.
— Explorer l'arborescence du système et découvrir son fonctionnement.
— Utiliser les commandes simples les plus courantes.
— Initiation aux scripts *shell*.
— Initiation à `awk`

1 Exploration et notions d'administration.

Note préliminaire : certaines commandes présentées ici nécessitent les droits d'administration (root) et ne pourront donc pas être exécutées pendant ce TP. Il s'agit d'une introduction au TP numéro 3.

Exercice 1 : `ls`, `grep` et `wc`
— Listez tous les fichiers et les répertoires (qui sont des fichiers !) du répertoire courant.
— Combien contient il de fichiers, tous types confondus ?
— Combien contient il de sous répertoires ?
— Combien contient il de fichiers ordinaires ?
— Combien contient il de liens symboliques ?

Exercice 2 : shell et configuration
— Quel est votre *shell* par défaut ?
— Quel est le nom (et l'emplacement) du fichier de ressources propre à chaque utilisateur pour ce *shell* ?
— Dans quel langage est écrit ce fichier ?
— Créez le fichier de ressources pour votre *shell* si ce n'est pas déjà fait, et ajoutez dedans les éléments suivants :
  — Modification de la variable `$PS1`
  — Ajout du support des couleurs : `eval $(dircolors -b)` et `alias ls='ls --color=auto'`
  — Ajout des alias pour `ls -l` (*ll*) et `ls -a` (*la*)
  — Chargement du fichier */etc/bash_completion* (`source /etc/bash_completion`)
— Comment faudrait-il faire pour qu'un fichier `.bashrc` soit créé pour chaque nouvel utilisateur créé (dans son répertoire personnel) ? (recherchez `"skel bashrc"` avec votre moteur de recherche préféré)

Exercice 3 : infos système
— Quels dossiers contiennent des informations sur l'état actuel du système ?
— Combien de processus sont actuellement en cours d'exécution ?
— Quel est le processeur de la machine que vous utilisez, et combien de cœurs a-t-il ?
— De combien de mémoire vive dispose la machine que vous utilisez ?
— Quels sont tous les systèmes de fichiers actuellement montés sur votre poste ?
— Quels sont les types de systèmes de fichiers actuellement supportés par le noyau en cours d'exécution sur votre machine ?

Exercice 4 : redirections

— Comment logguer les messages d'erreur (et seulement eux) dans un fichier lors de l'exécution d'une commande ?
— Comment ignorer ces messages d'erreur ?
— Comment utiliser le contenu d'un fichier comme entrée d'une commande ? (deux solutions)

2 Scripts.

Exercice 5 : Premiers pas

Un script *shell* est un fichier texte qui peut être interprété par un *shell*. Vous pouvez éditer un script en utilisant un éditeur de texte comme `vim`, `kate` ou `gedit`.

— Éditer le fichier `mon_script` avec le code suivant :

```
1 #!/bin/bash
2
3 echo
4 echo "Mon premier script"
5 echo "Nous sommes le $(date)"
6 num=$(ls . | wc -l)
7 name=$(basename $(pwd))
8 echo "Le dossier $name contient $num element"
9 echo
```

— Que signifie la première ligne de ce script ?
— Comment exécuter ce script ?

Exercice 6 : Premier Script

Si vous ne l'avez pas fait, il est temps de créer votre premier script.

Pour cela, dans un éditeur de texte quelconque, tapez les lignes suivantes que vous sauverez dans le fichier `premier.sh`

```
1 #!/bin/bash
2 # script écrit par David Odin et Nathaël Pajani
3 # Les lignes commençant par un # sont des commentaires
4
5 echo Mon premier script
```

Rendez ensuite ce script exécutable et exécutez le à l'aide des commandes suivantes :

```
1 chmod u+x premier.sh
2 ./premier.sh
```

Exercice 7 : Paramètres

Si l'on exécute un script de cette manière :

```
./second.sh plop "bonjour à tous" bla
```

... on dit alors que que le script `second.sh` a trois paramètres ;
- `plop`
- `bonjour à tous`
- `bla`

Les paramètres d'un script *shell* se retrouvent dans les variables `$1`, `$2`, `$3`, ... `$9`

D'autres variables particulières concernent aussi les paramètres du script :
- `$#` contient le nombre de paramètres du script
- `$*` contient tous les paramètres dans une seule variable
- `$@` contient tous les paramètres dans une seule variable de manière légèrement différente (voir "Paramètres spéciaux" ("Special Parameters") de la section "PARAMETERS" de la page du manuel de bash).

La commande **shift** permet de décaler les paramètres : le deuxième devient le premier, le troisième devient le deuxième, etc.

Pour cet exercice, vous devrez réaliser un script nommé `parametres.sh` qui affichera le nombre de ses paramètres, puis chacun des paramètres de cinq façons différentes :
- En parcourant la variable $*,
- En parcourant la variable $@,
- En parcourant "$*",
- En parcourant "$@",
- En utilisant la commande **shift**

Testez en utilisant les paramètres plop, "bonjour à tous", et bla.

Exercice 8 : Les conditionnelles dans les scripts
- Syntaxe :

```
if [  condition  ]
then
    action1
else
    action2
fi
# Attention: les espaces autour des crochets sont obligatoires
```

Le *shell* (la commande [en réalité) offre plusieurs opérateurs permettant de vérifier diverses conditions sur les fichiers :

- `-e nom-fichier` vrai si le fichier existe
- `-d nom-fichier` vrai si le fichier est un répertoire
- `-f nom-fichier` vrai si le fichier est un fichier ordinaire
- `-r nom-fichier` vrai si le fichier est accessible en lecture

Consultez la page man de [pour plus d'informations.
- Modifiez le script ci-dessus pour afficher une aide si aucun paramètre n'est passé, ou afficher un message d'erreur si le fichier n'existe pas ou n'est pas un *fichier ordinaire*.
- Ajoutez un appel à l'utilitaire *file* lorsqu'il s'agit d'un *fichier ordinaire*.

3 AWK

AWK est probablement l'une des commandes les plus puissantes disponibles dans le *shell*.

Exercice 9 : `grep` et plus

Il est assez facile d'utiliser `awk` pour remplacer `grep`. Indiquez comment avoir le comportement de

```
1 grep chaine < fichier
```

en n'utilisant que `awk`.

L'utilitaire `wc` compte le nombre de lignes, de mots et de caractères d'un fichier (de son entrée standard).

Sachant que :
— la variable `NR` contient le numéro de ligne en cours de traitement,
— `NF` contient le nombre de colonne de la ligne courante,
— la fonction `length()` renvoie la taille d'une chaîne de caractères,
proposez un programme `awk` implémentant les fonctionnalités de `wc`.

Exercice 10 : Numérotation

La variable `NR` permet aussi de numéroter les lignes d'un fichier. Comment ?

Proposez un programme `awk` qui numérote uniquement la première ligne et les lignes multiples de 5.

Exercice 11 : Lignes dupliquées

La ligne courante est accessible via la variable `$0`Cette variable peut évidemment être copiée.

En utilisant cela, proposez un programme `awk` supprimant les lignes dupliquées consécutives d'un fichier.

Le programme `awk` suivant supprime les lignes dupliquées (même non consécutives) d'un fichier. Comment fonctionne-t-il ? `awk '!a[$0]++'`

Le programme `tac` affiche les lignes d'un fichier dans l'ordre inverse. Proposez un programme `awk` qui implémente la même fonctionnalité.

Exercice 12 : Analyse de logs

Pour cet exercice, il sera probablement plus simple d'écrire un *script* `awk`. Cela se fait de la même manière que pour un script `bash`, mais la première ligne devra être

```
1 #!/usr/bin/awk -f
```

Nous voulons analyser le fichier `log.txt` présent sur le site du module afin d'avoir un résumé ressemblant à cela :

```
1 Voiture1 : 28 km
2 Camion8 : 52 km
3 Voiture2 : 3 km
4 voiture1 : 10 km
```

Vous devrez pour cela détecter les lignes du genre : `START Voiture1` et `FINISH Voiture1` et sommer les kilomètres entre ces lignes.