Première NSI - Système d'exploitation Linux

Cours

qkzk

Linux et les systèmes libres

Le BASH

1. Le terminal

Quelle que soit la déclinaison de Linux, on trouve une application "terminal" qu'on peut lancer et l'interpréteur de commande avec lequel on intéragit est par défaut Bash. (*Bourne Again Shell*) qui est l'interpréteur de commandes le plus courant sous Linux et aussi sous OSX. Il est également possible de l'activer sous windows 10.

Le terminal fonctionne avec le principe REPL : Read Eval Print Loop.

1. Read. L'utilisateur tape une commande qui est lue par l'interpréteur,
2. Eval. Cette commande est exécutée et retourne une chaîne de caractères,
3. Print. La chaîne de caractères est affichée à l'écran,
4. Loop. On recommence.

2. Les commandes de base

Toutes ces commandes acceptent de nombreuses options dont on peut consulter la documentation en tapant `man ls` par exemple pour la commande `ls`. Celle-ci comporte des options pour affiche les fichiers cachés `ls -a` ou encore pour afficher les détails et permissions d'un fichier `ls -l`

Commande	Description
`ls`	Lister le contenu du répertoire courant
`cp`	Copier des fichiers ou des répertoires
`mv`	Déplacer ou renommer des fichiers ou des répertoires
`rm`	Effacer des fichiers ou des répertoires
`cd`	Se déplacer dans l'arborescence
`cat`	Visualiser le contenu d'un fichier
`echo`	Afficher un message ou le contenu d'une variable
`touch`	Créer un fichier vide ou réinitialiser le *timestamp* d'un fichier

3. Les répertoires fondamentaux

Dans un système UNIX, on dispose d'une **arborescence de fichiers** ancrée sur `/`, la "racine" (*root*) du système de fichiers. Voici quelques points d'entrée de cette arborescence :

```
/
  bin     ← Commandes de base du système
  dev     ← Fichiers représentant les dispositifs matériels (devices) du système
  etc     ← Fichiers de configuration du système
  home    ← Répertoire d'accueil (HOME) des utilisateurs
  lib     ← Librairies
  mnt     ← Points de montage (clés usb etc.)
  proc    ← État du système et de ses processus
  root    ← Répertoire de l'administrateur système
  run     ← Variables d'état du système depuis le boot
  sys     ← Informations sur le noyau et les périphériques
```

```
usr     ← Logiciels installés avec le système, base de données etc.
var     ← Données fréquemment utilisées et modifiées
```

4. Quelques mots sur BASH

Bash (acronyme de Bourne-Again shell) est un interpréteur en ligne de commande de type script. C'est le shell Unix du projet GNU.

Usage

Comme tous les interpréteurs en ligne de commande de type script, Bash exécute quatre opérations fondamentales :

- Il fournit une liste de commandes permettant d'opérer sur l'ordinateur (lancement de programmes, copie de fichiers, etc.) ;
- Il permet de regrouper ces commandes dans un fichier unique appelé script ;
- Il vérifie la ligne de commande lors de son exécution ou lors d'une éventuelle procédure de vérification et renvoie un message d'erreur en cas d'erreur de syntaxe ;
- En cas de validation, chaque ligne de commande est interprétée, c'est-à-dire traduite dans un langage compréhensible par le système d'exploitation, qui l'exécute alors.

Les scripts sont de courts programmes généralement faciles à construire. Bash offre un service de gestion de flux, c'est-à-dire qu'il permet que le résultat d'un script (la sortie) soit transmis à un autre script (l'entrée). De cette façon, les scripts peuvent être « chaînés », chacun effectuant une seule tâche bien délimitée.

Les scripts peuvent être exécutés manuellement par l'utilisateur ou automatiquement par le système. Par exemple, dans la distribution GNU/Linux Ubuntu, le répertoire `resume.d` contient un certain nombre de scripts qui s'exécutent automatiquement lors du redémarrage du système, c'est-à-dire après la fin de la mise en veille de celui-ci. Ces scripts servent à relancer les différents programmes interrompus par la mise en veille.

Fonctionnement

Bash est un shell qui peut être utilisé soit en mode interactif, soit en mode batch :

- mode interactif : Bash attend les commandes saisies par un utilisateur puis renvoie le résultat de ces commandes et se place à nouveau en situation d'attente.
- mode batch : Bash interprète un fichier texte contenant les commandes à exécuter.

Arborescence et flux

Observons quelques moyens de naviguer dans l'arborescence d'un système UNIX, puis voyons comment manipuler les entrées/sorties et les redirections

1. Navigations et entrées / sorties en shell BASH

1. Naviguer dans une arborescence, les chemins

- Pour aller dans son HOME en utilisant un "chemin absolu", c'est-à-dire un chemin depuis la racine, l'utilisateur `bob` peut faire : `cd /home/bob/`
- Il peut aussi utiliser le raccourci `~`, et faire `cd ~` ou plus simplement `cd`
- Pour remonter dans le répertoire parent, on utilise `cd ..` et on désigne le répertoire courant avec un point "`.`"
- Ainsi, si on veut copier le fichier `toto` qui se trouve dans le répertoire parent vers le répertoire courant, on tape : `cp ../toto .` Nous utilisons cette fois un chemin "relatif" qui part de la position actuelle.

2. Entrées et sorties

- **Entrées**
  La commance `read` permet d'effectuer une saisie utilisateur.

  ```
  read var # Saisie de var (stdin)
  echo $var # Réaffichage, remarquez le $ qui précède le nom d'une variable
  ```

- **Sorties**
  Une commande affiche normalement son résultat sur la sortie standard *stdout*.
  On peut aussi envoyer ce résultat de commande vers un fichier.

Exemple : `echo "Bonjour" > salut.txt` envoie le mot “Bonjour” dans le fichier `salut.txt` qui est crée (ou réinitialisé s'il existait préalablement), puis, si on tape `echo "Tout le monde" >> salut.txt`, on ajoute une nouvelle ligne au fichier existant qui contient maintenant 2 lignes.

- **Sortie d'erreur**
  Les sorties d'erreur ne s'affichent pas sur la sortie standard, il existe un troisième flux, le flux de sortie d'erreur dénommé *stderr*

  Par exemple, s'il n'y a pas de fichier `toto` dans le répertoire courant, la commande :

  ```
  cat toto
  ```

  déclenche une erreur indiquant que le fichier `toto` n'existe pas. Si on ajoute :

  ```
  cat toto 2>/dev/null
  ```

  on effectue la même commande, mais on redirige vers le périphérique “null” les éventuelles sorties d'erreur pour qu'elles ne s'affichent pas.

3. Les filtres, tubes et redirections

- Unix permet aussi l'utilisation de **filtres** comme
  - `grep` pour afficher les lignes qui contiennent un mot clé ;
  - `wc` pour compter des lignes ou des caractères ;
  - `sort` pour trier ;
  - `cut` pour extraire des colonnes dans un fichier ;
  - `tr` pour transposer ou supprimer des caractères.
- Ces filtres s'utilisent généralement avec des tubes (*pipes*) notés “`|`”.

  Exemple :

  ```
  cat monfic | wc -l
  ```

  compte le nombre de lignes dans le fichier `monfic` et :

  ```
  cat monfic | sort > fictrie
  ```

  trie les lignes du fichier `monfic` selon l'ordre lexicographique puis place le résultat correspondant dans le fichier `fictrie`

2. Scripts BASH

- Une suite de commandes permet à l'administrateur d'automatiser certaines tâches, on parle alors de “scripts”, stocké dans un fichier d'extension “`.sh`”. Les arguments du script peuvent être invoqués avec `$1, $2` etc., leur nombre avec `$#` et leur liste avec `$*`.
- Voici, par exemple un script `efface.sh` qui, au lieu d'effacer le fichier qu'on passe en argument, le déplace dans un dossier “poubelle” à la racine du HOME et l'utilisateur (la première ligne indique l'interpréteur utilisé).

```
#! /bin/bash
# Ce script a un argument : un nom de fichier
#  - crée si besoin un répertoire poubelle dans le répertoire HOME
# de l'utilisateur
#  - déplace le fichier donné en argument dans ce répertoire
# poubelle
# vérifie d'abord si ce dossier existe dans ~ :
if [ -d ~/poubelle ]
then
    echo "Le répertoire poubelle existe déjà dans votre Home."
else # sinon, on le crée
    mkdir ~/poubelle
    echo "Repertoire poubelle inexistant. Il est crée."
fi
# On déplace dans poubelle le fichier donné en argument
mv $1 ~/poubelle
```

Droits et permissions sous UNIX

Le système de droits et de permissions sous UNIX est aspects fondamentaux de la gestion de la sécurité du système.

Droits et groupes

1. Le monde selon UNIX

Unix sépare le monde en trois catégories du point de vue des droits :

- L'utilisateur (user) ;
- Le groupe (group) ;
- Le reste du monde (others).

2. Exemple de lecture de droits

- En utilisant la commande `ls -l monfic.sh` par exemple, on obtient :

```
-rwxr--r-- 1 roza   staff   0   6 mai 11:56 monfic.sh
```

- La partie `-rwxr--r--`, indiquant les droits du fichier, se lit en omettant le tiret du début, puis en décomposant en trois parties :
  - `rwx` (utilsateur) ;
  - `r--` (groupe) ;
  - `r--` (autres).
- Chaque partie est elle-même composée de trois lettres :
  - droit de lecture `r` ;
  - droit d'écriteure `w` ;
  - droit d'exécution `x` : on peut exécuter le fichier en l'invoquant par son nom, dans cet exemple `./monfic.sh`.
- On sait donc que `monfic.sh` est accessible en lecture au groupe "Staff" et aux autres.
- On sait en outre que le fichier appartient à l'utilisateur "`roza`".

3. Les droits d'un répertoire

- Créeons un répertoire `www` dans notre HOME et lisons les droits correspondants avec la commance `ls -l www`, on obtient par exemple :

  ```
  drwxr-xr-x  2 roza  staff   64 6 mai 14:17 www
  ```

- Le `d` initial signifie qu'il s'agit d'un répertoire.

Remarque : le droit `x` pour un répertoire est le droit de traverser ce répertoire.

2. Changer des droits

1. La commande `chmod`

Seul le propriétaire d'un fichier (ou l'utilisateur "root") peut changer ses permissions d'accès. Il le fait avec la commande `chmod` dont voici quelques exemples d'utilisation.

Droits	Syntaxe
Donner les droits de lecteure au groupe `g`	`chmod g+r monfic.sh`
Donner les droits d'écriture au propriétaire `u`	`chmod u+w monfic.sh`
Retirer les droits d'exécution aux autres (*others*) `o`	`chmod o-x monfich.sh`
Donner les droits d'exécution à tous	`chmod ugo+x monfic.sh`

Remarque : On peut aussi utiliser `a` (all) à la place de `ugo`.

2. Droits symboliques et numériques

Il est également possible d'utiliser un codage octal (base 8) pour les droits.

- L'écriture symbolique : `rwc r-x r-w`
- Correspond à l'écriture binaire `111 101 101` soit `755` en octal

Par exemple `chmod 755 monfic.sh` donne les droits `-rwxr-xr-x` au fichier `monfic.sh`.

3. Lancement d'un script et fichiers exécutables

- On peut lancer un script (ou autre) en l'invoquant par son nom s'il est exécutable. On précise parfois que le script est dans le répertoire courant en le précédant d'un `./` devant son nom :

  ```
  ./monfic.sh
  ```

- Si le script n'est pas exécutable on peut toujours le lancer en tapant :

  ```
  source monfic.sh
  ```

- Ces deux méthodes ne sont pas équivalentes : dans le premier cas, un nouveau `shell` est crée tandis que dans le second, les commandes du script s'exécutent dans le `shell` courant.

Remarque : Tous les fichiers exécutables posent des problèmes potentiels de sécurité, tout fichier exécutable pouvant se transformer en éventuel "cheval de Troie" (logiciel malveillant). Dans un site web par exemple, les fichiers HTML, CSS, images, JavaScript ou PHP n'ont pas à être exécutables, le droit de lecture suffit.