SYSTÈME D'EXPLOITATION UNIX PROGRAMMATION SCRIPTS

Dr. Fatimazahra EL BIACH
Dr. Siham LAMZABI
Dr. Mohammed BELATAR

2024

Plan du semestre

Semaine 1 : Programmation par scripts

Semaine 2 : Programmation par scripts (suite)

Semaine 3 : Mécanismes de base

Semaine 4 : Fonctionnement en interactif

Semaine 5 : Shell-scripts portables

Semaine 6 : Mécanismes complémentaires

Semaine 7 : Mécanismes complémentaires (suite)

Semaine 8 : Evaluation

Semaine 9 : Automatisation des tâches

Semaine 10 : Automatisation des tâches (suite)

Semaine 11 : Automatisation des tâches (suite)

Semaine 12 : Manipulation des fichiers et les données

Semaine 13 : Manipulation des fichiers et les données (suite)

Semaine 14 : Langage AWK

Semaine 15 : Langage AWK (suite)

Semaine 16 : Evaluation

Introduction et Rappels

semaine 1

Shell, c’est quoi exactement?

- Le Shell : Il s’agit d’une interface texte entre l’utilisateur et le système informatique
  - Tout se fait au clavier
  - Pas de clic de souris
- L’utilisateur tape des commandes qui sont exécutées par le système
  - Le shell est donc un « interpréteur de commandes »
  - Chaque commande a une syntaxe particulière
  - Il existe des milliers de commandes différentes
- Les commandes peuvent aussi provenir d’un fichier
  - Le fichier contient les commandes à exécuter
  - L’utilisateur appel le fichier plutôt que de taper toutes les commandes
    - Utile pour les tâches répétitives

Pourquoi le shell?

- Le shell reste le moyen le plus efficace pour contrôler le système. C’est aussi le plus utilisé sous Linux/Unix.
- Le shell est un véritable environnement de programmation
  - Variables, boucles, structures de contrôle « if »
  - Programmes
- Les programmes écrits pour le shell sont interprétés au moment de l’exécution
  - Aucune compilation préalable n’est nécessaire
  - On peut profiter de différents outils développés sous différents langages
  - Les performances n’égalent pas un programme en C
- Contrôler une machine sans y avoir un accès physique (Serveur)

Rappel : langages Compilés vs Interprétés

Leurs principales différences sont les suivantes :

- Après l'écriture d'un fichier script, il est possible de le soumettre directement à l'interpréteur de commandes, tandis qu'un code source écrit en langage compilé doit être traduit en instructions de code machine compréhensibles pour le processeur.
- Le code compilé étant directement compris par le processeur du système, son exécution est très rapide, alors qu'un script doit être interprété dynamiquement, ce qui ralentit sensiblement l'exécution.
- Le fichier exécutable issu d'une compilation est souvent volumineux et n'est utilisable que sur un seul type de processeur et un seul système d'exploitation. À l'inverse, un fichier script est généralement assez réduit et directement portable sur d'autres processeurs ou d'autres systèmes d'exploitation du moment que l'interpréteur de commandes correspondant soit disponible.
- Un fichier compilé est incompréhensible par un lecteur humain. Il n'est pas possible d'en retrouver le code source. Cela peut garantir le secret commercial d'un logiciel. Inversement, un fichier script est directement lisible et modifiable, et peut contenir sa propre documentation sous forme de commentaires.

Les scripts shell

Script: Un script est un fichier contenant une série d'ordres que l'on va soumettre à un programme externe pour qu'il les exécute. Ce programme est appelé interpréteur de commandes. Il existe de nombreux interpréteurs de commandes. Naturellement, le shell en fait partie, tout comme certains outils tels que Sed et Awk que nous verrons ultérieurement, ainsi que d'autres langages tels que Perl, Python, Tcl/Tk, Ruby, etc. Ces langages sont dits interprétés, par opposition aux langages compilés (comme C, C++, Fortran, Ada, etc.)

Rappel des principes de base

- Super utilisateur : root
  - s’occupe de l’administration du système UNIX (installation des logiciels, création des profils utilisateurs, sauvegarde et restauration des données etc...)
- Hôte (serveur) :
  - système centralisé sur lequel peuvent se connecter les utilisateurs
- Console (client) :
  - un écran, le plus souvent noir, destiné à recevoir des commandes Shell via un clavier, branchée directement à la machine Hôte
- Terminal :
  - Environnement d’entrée/sortie de texte permettant l’émulation d’une console

Rappel des principes de base

- Voir Annexe : liste des commandes de base
- Caractères de contrôle :
  - <cr> fin de ligne (retour chariot)
  - <tab> tabulation
  - <bs> ^H, backspace, effacement du caractère précédent
  - <del> ^?, souvent identique à <bs>
  - <^C> interruption d'un processus attaché au terminal (SIGINT/2)
  - <^\> arrêt / quit (SIGQUIT/3)
  - <^Z> suspension d'un processus en premier plan (SIGSTOP/19)
  - <^U> effacement de la ligne complète
  - <^W> effacement du mot qui précède
  - <^D> fin de fichier => si le shell lit, fin d'un shell
  - <^S> suspension de l'affichage écran (Xoff)
  - <^Q> reprise de l'affichage écran (Xon)

```
$ stty -a
```

Rappel des principes de base

- le “prompt” (invite de commandes): variable PS1
  - $ en mode utilisateur
  - # en mode super-utilisateur
- Prompt secondaire : variable PS2
  - > en mode interactif
- Chemin de recherche des noms de commandes : variable PATH
- Structure d’une commande
  - une commande simple est une séquence de mots séparés par un séparateur blanc et de redirections. Le premier mot désigne le nom de la commande à exécuter, les mots suivants sont passés en arguments à la commande.
  - la valeur retournée d'une commande est celle de son exit

Rappel des principes de base

- Les commandes :
  - syntaxe : <commande> <liste d'arguments>
  - sensibilité à la casse (majuscules/minuscules)
  - importance du séparateur “espace”
- le manuel (help) :
  - man <commande Unix>
- Historique :
  - commande “fc”
  - commande “history”
- Nettoyage de de l'écran
  - commande “clear”

Rappel des principes de base

- Enchaînement de commandes:
  - Commande simple en arrière plan
    - $ cmd [arg ...] &
  - Pipeline (tube de communication entre commandes)
    - $ cmd1 [arg ...] | cmd2 [arg ...] | …
  - Exécuter la commande suivante en fonction de l'exit de la première
    - $ cmd1 [arg ...] || cmd2 [arg ...]
    - $ cmd1 [arg ...] && cmd2 [arg ...] ...
  - Séquencement de commandes successives en premier plan
    - $ cmd1 [arg ...]; cmd2 [arg ...]; cmd3 …
  - Groupement des commandes : exécution en sous-shell
    - $ (cmd1 [arg ...]; cmd2 [arg ...]; cmd3 …)
  - Groupement des commandes : exécution dans le shell courant
    - $ { cmd1 [arg ...]; cmd2 [arg ...]; cmd3 …;}

Les scripts shell : exemple

Un ensemble de commandes, d'outils et de programmes rassemblés dans un seul fichier dans une structure particulière permettant d'accomplir une ou plusieurs tâches

- Exemple :

```
# Test de l'existence du fichier
if [ -f $logfile ]
then
  rm $logfile
  echo effacement de ipscan.log effectue
fi

echo scan des ip...
```

Choix du shell

- Plusieurs versions du shell:
  - sh (Bourne Shell) : créé par Stephen R. Bourne en 1977
  - csh (C Shell) : créé par Bill Joy en 1979
  - tcsh (C Shell amélioré) : créé par Ken Greer en 1979
  - ksh (Korn Shell) : créé par David Korn en 1982
  - bash (Bourne again shell) : créé par Brian Fox en 1987
  - zsh (Z Shell) : créé par Paul Falstad en 1990
- Possibilité de choisir
  - 1 exemplaire particulier du Shell par utilisateur
  - Il peut lancer plusieurs Shells sur sa session
- Sous Linux, on peut choisir son shell (commande chsh)
  - Le shell bash domine le marché depuis longtemps

Pourquoi écrire un script shell ?

- Exemples de domaines d'utilisation :
  - lancer quelques commandes disposant d'options complexes, et qu'on désire employer régulièrement sans avoir à se reporter sans cesse à leur documentation.
  - scripts d'initialisation (de boot) du système, permettent d'énumérer les périphériques disponibles, et de les initialiser, de préparer les systèmes de fichiers, les partitions, et de configurer les communications et les services réseau.
  - préparation ou installation de certaines applications spécifiques (logiciels) nécessitent un paramétrage complexe.
  - tâches automatisées pour vérifier l'état du système (mémoire, espace disque, antivirus...) et maintenance périodique.
  - exploiter la sortie de certaines commandes et programmes, et les traces des scripts Shell enregistrées dans des fichiers de journalisation

TD 1

1. Affichez les shells disponibles sur votre système
2. Affichez votre shell actuel
3. Affichez votre shell par défaut
4. Basculez d’un shell à un autre puis fermer chaque shell ouvert
5. Changez votre shell par défaut
6. Vérifiez la disponibilité des éditeurs de texte et familiarisez-vous avec l’un d’eux

TP 1

1. En utilisant les commandes tail et head, écrire un script qui permet de réordonner les lignes d'un fichier "losange" contenant un losange dont les lignes sont inversées.
2. Re-écrivez votre script en une seule ligne.