LISV Laboratoire d'ingénierie des systèmes de Versailles
UVSQ université PARIS-SACLAY

Linux : TP 4

Halim Djerroud

révision 1.0

Exercice 1 : Exploration approfondie des processus

1. Listez les processus de votre session avec `ps`. Comparez les sorties de `ps -ef`, `ps aux` et `ps -ejH`. Quelles différences remarquez-vous ?
2. Identifiez au moins deux processus différents :
   — un processus système (démarré automatiquement par Linux),
   — un processus utilisateur (lancé par vous).
   Relevez leurs PID, PPID et UID.
3. Surveillez les processus actifs avec `top`. Observez l'évolution du pourcentage CPU/mémoire d'un processus au cours du temps. Faites de même avec `htop` et `atop` si disponible (sinon ignorez).
4. Lancez un programme qui consomme des ressources (par ex. une boucle infinie en bash ou un calcul volontairement long).

   Exemple de scripte gourmand :

```bash
#!/bin/bash
# boucle_infinie.sh : script volontairement consommateur de CPU

echo "Ce script va lancer une boucle infinie..."
echo "Appuyez sur CTRL+C pour l'arrêter (ou utilisez kill)."

# Boucle qui calcule sans fin
while true
do
    echo $((42**42)) > /dev/null
done
```

   Repérez-le avec `top` et arrêtez-le avec `kill`. Vérifiez qu'il n'apparaît plus.
5. Question de réflexion :
   — Pourquoi est-il risqué de tuer un processus système « critique » ?
   — Donnez un exemple de processus que vous ne devriez jamais terminer.

Exercice 2 : Jobs, signaux et exploration de /proc

1. Lancez une commande longue (par ex. `sleep 500`) au premier plan. Interrompez-la avec `Ctrl+Z`. Listez vos jobs et reprenez-la en arrière-plan. Ramenez-la ensuite au premier plan.
2. Trouvez son PID avec `ps` ou `pgrep`. Explorez son répertoire `/proc/<PID>` :
   — Quel est le contenu de `cmdline` ?
   — Que dit le fichier `status` sur l'état du processus ?
   — Quels fichiers apparaissent dans le répertoire `fd/` ?
3. Envoyez-lui un signal `SIGTERM`. Vérifiez son effet. Relancez-le puis envoyez-lui un signal `SIGKILL`. Quelle différence constatez-vous ?
4. Observez la liste des signaux disponibles avec `kill -l`. Recherchez au moins deux autres signaux utiles et décrivez leur usage.
5. Question de réflexion :
   — Pourquoi faut-il privilégier `SIGTERM` avant `SIGKILL` ?
   — Que devient un processus zombie et comment peut-il disparaître ?

Exercice 3 : Manipulation des variables d'environnement

1. Créez une variable `COURS` avec la valeur `Linux` et affichez-la. Lancez ensuite un nouveau shell (`bash`) et vérifiez si la variable existe encore.
2. Reprenez la même variable mais exportez-la avant de relancer un nouveau shell. Est-elle disponible cette fois-ci ? Expliquez la différence.
3. Affichez les variables d'environnement liées à votre session (`PATH`, `HOME`, `USER`, `SHELL`). Notez leur rôle.
4. Modifiez temporairement `PATH` pour y inclure le répertoire courant (`.`). Créez un petit script `hello.sh` qui affiche « Hello ». Vérifiez qu'il s'exécute sans préciser `./`.
5. Question de réflexion : pourquoi est-il dangereux de mettre `.` au début de `PATH` ?

Exercice 4 : umask et persistance des variables

1. Affichez votre valeur actuelle de `umask`. Créez un fichier et un répertoire, puis notez leurs permissions par défaut.
2. Modifiez temporairement `umask` à `002` et recommencez l'opération. Comparez les différences.
3. Revenez à votre valeur initiale de `umask`.
4. Rendez permanente une variable personnalisée (par exemple `TP=Linux3`) en l'ajoutant à votre fichier `~/.bashrc`. Rechargez ce fichier sans fermer votre session. Vérifiez que la variable est bien disponible.
5. Question de réflexion : dans un projet de groupe, quelle différence cela ferait-il d'utiliser `umask 022` ou `umask 002` ?

Exercice 5 : Scripts Bash

1. Créez un fichier `bonjour.sh` contenant :

```
#!/bin/bash
echo "Bonjour $USER, nous sommes le $(date)"
```

2. Rendez le script exécutable et exécutez-le de deux manières différentes :
   — en appelant directement le fichier,
   — en passant par l'interpréteur `bash`.
3. Observez les différences si vous exécutez avec `source bonjour.sh` au lieu de `./bonjour.sh`.
4. Question de réflexion : quelle différence entre `bash script.sh` et `source script.sh` ?

Exercice 6 : Arguments et interaction

1. Créez un script `infos.sh` qui affiche :
   — le nom du script (`$0`),
   — le premier argument (`$1`),
   — le nombre total d'arguments (`#`).
2. Testez votre script avec différentes commandes, par exemple :

```
./infos.sh Linux est génial
```

3. Modifiez le script pour qu'il demande à l'utilisateur de saisir son prénom avec la commande `read`, puis affiche un message de bienvenue personnalisé.
4. Question de réflexion : dans quel cas est-il préférable d'utiliser des arguments plutôt qu'une saisie interactive ?

Exercice 7 : Conditions avec `if`

1. Écrivez un script `pair.sh` qui prend un nombre en argument et affiche s'il est pair ou impair. (Indice : utilisez l'opérateur arithmétique `%`).
2. Modifiez le script pour qu'il affiche aussi un message d'erreur si aucun argument n'est donné.
3. Question de réflexion : comment vérifier qu'un argument est bien un nombre avant de l'utiliser ?

Exercice 8 : La boucle `for`

1. Écrivez un script `liste.sh` qui affiche tous les fichiers `.txt` du répertoire courant avec leur taille (commande `ls -lh`).
2. Modifiez le script pour qu'il affiche aussi le nombre total de fichiers `.txt`.
3. Question de réflexion : que se passe-t-il si aucun fichier `.txt` n'existe ?

Exercice 9 : La boucle `for` et `seq`

1. Écrivez un script `nombres.sh` qui affiche les nombres de 1 à 10 en utilisant une boucle `for` et la commande `seq`.
2. Modifiez le script pour qu'il affiche uniquement les nombres pairs entre 1 et 20.
3. Question de réflexion : comment pourriez-vous modifier le script pour que les bornes (début et fin de la séquence) soient passées en arguments de la ligne de commande ?

Exercice 10 : La boucle `while`

1. Écrivez un script `compteur.sh` qui demande un nombre à l'utilisateur puis compte de 1 jusqu'à ce nombre en utilisant une boucle `while`.
2. Ajoutez une pause d'une seconde entre chaque affichage (`sleep 1`).
3. Question de réflexion : quelle différence entre `while` et `until` ?

Exercice 11 : La structure `case`

1. Écrivez un script `menu.sh` qui affiche un petit menu :
   — `1)` Afficher la date,
   — `2)` Afficher l'utilisateur courant,
   — `3)` Quitter.
2. Utilisez `read` pour demander le choix à l'utilisateur, et `case` pour exécuter la commande correspondante.
3. Ajoutez un cas par défaut qui affiche « Choix invalide ».
4. Question de réflexion : dans quel cas pratique un tel script pourrait-il être utile ?

Mini-projet : Analyse d'un répertoire avec un script Bash

Objectif : écrire un script Bash capable d'analyser un répertoire en fonction de paramètres passés en ligne de commande.

Travail demandé : Écrivez un script `analyse_dir.sh` qui prend deux arguments :
— le chemin d'un répertoire,
— une option d'analyse (parmi `taille`, `nbfichiers`, `extensions`).
Le script doit :

1. Vérifier que l'utilisateur a bien fourni deux arguments, sinon afficher un message d'usage : `Usage : ./analyse_dir.sh <répertoire> <option>`.
2. Vérifier que le premier argument est bien un répertoire valide, sinon afficher une erreur.
3. Selon la valeur du deuxième argument (`case`) :
   — **taille** : afficher la taille totale du répertoire (commande `du -sh`).
   — **nbfichiers** : afficher le nombre de fichiers réguliers contenus (boucle + test).
   — **extensions** : lister les extensions des fichiers présents et leur nombre d'occurrences (triées par fréquence).
4. Ajouter un cas par défaut si l'option est invalide.
5. Bonus : si aucun argument n'est donné, demander interactivement le répertoire et l'option avec `read`.

Exemples d'utilisation :

```
./analyse_dir.sh ~/Documents taille
./analyse_dir.sh ~/projet nbfichiers
./analyse_dir.sh ~/projet extensions
```