Table des matières

1 Introduction

Les objectifs de la formation

Connaître les caractéristiques des principaux outils de scripting Unix/Linux
Savoir lire des scripts Unix/Linux écrits en Shell, Perl, Python Ruby ou AWK
Être capable d'écrire des scripts simples d'exploitation Unix/Linux
Comprendre comment choisir l'outil le plus adapté pour résoudre un problème particulier

Prez perso et demander le niveau des gens

2 Le Shell - les bases

2.1 Qu'est ce qu'un shell

Un shell Unix est une interface homme machine (IHM) en ligne de commande (CLI). Il fournit à la fois un langage de commandes interactives et un langage de scripting. Le shell traite des commandes ou scripts.
Il ne faut pas confondre un shell avec un terminal. Un terminal était initialement physiquement un écran et un clavier. Aujourd'hui lorsque l'on parle de terminal on parle d'émulateur de terminal, c'est une catégorie de logiciels permettant de fournir un GUI pour lancer *des* shells (bash, python, zsh, fish, powershell, ruby ...).
Émulateur de terminaux connus : alacritty, Windows Terminal, urxvt, GNOME Terminal, PuTTY.

La confusion est courante car sur Windows historiquement le nom du shell et de l'émulateur de terminal étaient les mêmes (cmd, powershell...), ce n'est plus le cas avec Windows 11 et le Windows Terminal.

2.2 Différents shells Unix

"sh" (shell command langage) est un spécification de langage défini par POSIX mais n'est pas une implémentation en lui même. Il y a diverses implémentations, la plus connue étannt Bash. Le fichier /bin/sh est en réalité un lien symbolique vers une implémentation sur la plupart des systèmes Linux, souvent bash. ls -l /bin/sh
Bash est l'implémentation la plus connue et utilisée, nous utiliserons donc Bash au cours de cours.
Quelques autres implémentations connues sont Ksh (Korn Shell), qui est une implémentation plus ancienne que Bash et surtout présente sur des systèmes moins récents. Zsh est l'implémentation par défaut sur MacOS et offre aussi des fonctionnalités pratiques en mode intéractif (complétion tab avec un menu naviguable ou encore une forte customisabilité par exemple).

Quasiment tout les shells Unix suivent a minima ce qui est décrit par POSIX, la plupart rajoutent ensuite diverses fonctionnalités. Lorsque l'on fait un script qui serait amené à être utilisé sur divers systèmes qui n'auraient pas forcément le même shell il peut être judicieux de se contenter d'utiliser ce que POSIX décrit. Exemple de fonctionnalité disponible sur Bash et qui n'est pas "POSIX compliant" : test et [] sont POSIX compliant mais [[]] ne l'est pas. Les deux premiers sont strictements pareils, [étant un alias de test, le dernier permet notamment d'utiliser des "Wildcards Patterns" comme *.

Pour information lancer un script via /bin/sh avec /bin/sh étant un symlink vers bash va lancer bash en mode posix ce qui rendra bash le plus POSIX compliant possible. Posix mode : https ://www.gnu.org/software/bash/manual/html_node/Bash-POSIX-Mode.html

En pratique il est rare d'utiliser des fonctionnalités non disponibles sur d'autres shell tout comme il est au final rare d'utiliser autre chose que Bash, néanmoins il peut être utile de garder ceci dans un coin de la tête.

2.3 Rappels Linux essentiels pour le scripting

Il y a plusieurs aspects de Linux qui sont importants à comprendre pour mieux comprendre le scripting sous Unix.

stdin / stdout / stderr recup stdout et in des process dans /proc/pid/fd/1 et 2

return code, droits fichiers, +x, shebang

2.4 Intérets des Shells Unix

Le shell malgré sa syntaxe archaique et ses fonctionnalités limitées vis à vis de langages interprétés à usage général a tout de même encore des avantages.

2.4.1 Actions uniques

Utile lorsque l'on veut faire une tâche relativement simple que quelques fois. Maitriser le shell permet d'être beaucoup plus rapide dans un environnement Unix Exemple : extraire des données textes et les process sommairement cat / sed / cut / tr ...

2.4.2 Manipulation de binaires

L'une des force de Bash par rapport à d'autres langages et sa facilité à manipuler directement des binaires. En python, par exemple, on peut aussi manipuler des binaires, mais on sent clairement que cela est moins pensé pour. Exemple : Utiliser un exécutable propriétaire (import baroc), utiliser l'output d'un script python et l'injecter ailleurs etc

2.4.3 Manipulation de texte

Tout est texte Exemple : lire des logs

2.4.4 Manipulation OS linux

les deux derniers = Manipulation linux pur car linux déjà tout fichiers et texte + binaires unix

Philo kiss, pleins de petits binaires linux avec une seule tache, trop complexe go python

2.5 Détails syntaxiques

2.5.1 Redirections et charactères spéciaux

Les flux standards sur Linux sont le standard out (stdout), standard error (stderr) et standard input (stdin). Normalment chaque process Linux possède un flux de chaque par défaut.

Ces flux servent à transmettre des données entre une source et une sortie. Les données sur linux sont toujours du texte et la commande lancée représente une des extrémités de ces flux.

Le stdin est le flux d'entrée dans la commande lancée, le stdout la sortie normale de la comande et le stderr est le canal pour les erreurs.
Par défaut le stdout et le stderr sont envoyés dans le terminal en cours. On peut rediriger ces sorties vers d'autres scripts, commandes ou fichiers.

Sur Linux tout est un fichier, même ces trois flux, ils sont dénommés 0 pour le stdin, 1 pour le stdout et 2 pour le stdout. Ces fichiers sont situés dans /proc/PID/fd.

Redirection vers un fichier :

On peut utiliser > et » pour rediriger le stdout vers un fichier, les doubles chevrons permettent d'ajouter à la fin d'un fichier déjà existant tandis que le simple chevron écrase le fichier. Dans les deux cas si le fichier n'existait pas il sera créé.

Cela ne redirige pas le stderr par défaut. Script d'exemple : # !/bin/sh echo hello world cat nexistepas

chmod +x test.sh ./test.sh > out

On peut expliciter ce qui est redirigé en précisant le numéro du fichier correspondant.
./test.sh 2> errors
./test.sh 1> out
./test.sh 1> out 2> errors
Si on veut rediriger à la fois le stdout et le stderr vers le même fichier avec une syntaxe plus courte que d'écrire deux fois le fichier on peut faire :
./test.sh > out 2>&1
Cela dit au shell de rediriger 2 (stderr) vers la même sortie que 1 (stdout).

< > » et sterr redirection

pipes et flow de texte Les caractères spéciaux (jockers, échappements, redirection)

2.5.2 Structures de contrôle

if then else fi while do done until do done for do done case esac
test et [et [[

2.5.3 Variables

2.5.4 Fonctions

2.5.5 Arguments parsing

- vs –

2.5.6 Binaires utiles

Force et efficience des binaires linux cut, cat, echo, grep, tr, sed, xargs, tail, df, ls diff
man ! RTFM

2.6 Exercices

Récupérer la liste des pourcentages de remplissages des filesystems

2.6.1 Script

2.6.2 Sans scripts

df | tail -n +2 | tr -s " " | cut -d " " -f 5
Puissance de juste one commande + pipe + redirections
Le Shell POSIX/ISO
L'écriture de script Shell
Activation des commandes POSIX/ISO
Les caractères spéciaux (jockers, échappements, redirection)
Les variables
Les structures de contrôle

3 Le langage Perl - les bases

Prez, utilité de nos jours, spécifité
Présentation de Perl
Les variables scalaires, les tableaux, les opérateurs
Les instructions de contrôle
Les tableaux associatifs (hash)

4 Le langage Ruby - les bases

Présentation de Ruby
Les variables
Les chaînes de caractères
Les structures de contrôle
Les tableaux, les itérateurs - Les hash

5 Le langage Python - les bases

Python est un langage de programmation interprété à usage extrêmement populaire de nos jours. Sa facilité facilité d'apprentissage et de lecture est ce qui a fait sa popularité initiale, aujourd'hui c'est la communauté qui en fait sa force avec les milliers de modules et programmes Python disponibles.

Pour un administrateur système et/ou un devops Python est un langage très attirant, son principal défaut, la potentielle lenteur au runtime, n'est pas un problème dans nos cas d'usage. Python est aussi ce qui est derrière Ansible et permet la création de modules Ansible customs, le plus important outil d'infrastructure as code (IAC) du moment. Il est aussi par défaut installé sur la très grande majorité des distributions Linux.

5.1 Syntaxe

Variables et expressions
Les tableaux, les chaînes de caractères
Les instructions de contrôle
Les dictionnaires (hash)

5.2 Exercices Python

6 Les expressions régulières (RegExp)

regex car tout est texte Importance de grep et sed RegExp en Shell (via grep et sed)
RegExp en Perl (normes)
RegExp en Python (module re mais osef)

7 La modularité en Shell, Perl, Python et Ruby

parler de direnv, de requirements.txt, de venv python, de classes python (osef un peu) y a des trucs pour shell (bpkg) mais pas le but, illogique, shell = spécifique task

Les fonctions => dans les parties syntaxes
Les paquetages=> oui
L'approche objet => syntaxe
Utilisation de bibliothèques externes=> oui

8 La programmation parallèle en Shell, Perl, Python et Ruby

différentes concurrency xargs pipes et parallel pour shell multithreading et async pour python

9 Résoudre des problèmes avec le Shell, Perl, Python et Ruby

Ecrire des scripts d'exploitation (activer une application, les signaux, ...)
Manipuler des fichiers
Faire des calculs
Ecrire des CGI Web
Accéder à des bases de données
Manipuler des fichiers XML (parsing, validation, création)
Créer des applications réseaux TCP/IP

Ptit serveur web Génération de fichier dans un dossier service shell filewatcher qui trigger un truc xargs et des pipes pour multiprocessing perl ou grep pour regex awk pour un truc
9 - AWK : un sous-ensemble POSIX/ISO du langage Perl
10 - Conclusion
Quel outil pour quoi faire ?

Subtitle