

DevOps : fondamentaux

Un peu de Linux, de git, de bash et de Python

Gilles Pietri

23 octobre 2017

- 1 Administration Linux
- 2 Versioning et git
- 3 Développement avec Bash
- 4 Python pour l'administrateur



Introduction

Cette formation a pour objectif de garantir les connaissances de base pour pouvoir aborder sereinement les outils « DevOps » de mise en conformité comme Ansible, Puppet, Saltstack...

Elle ne s'adresse pas à des grands débutants, mais à des administrateurs déjà opérationnels sur Linux, ayant également une base d'algorithmique établie.



Administration Linux

- 1 Administration Linux
 - Gestion des paquets et logiciels
 - Gestion des utilisateurs
 - Répertoires et permissions
 - Configuration SSH
 - Éditeurs de texte



Gestionnaire de paquets

On distingue 2 familles principales selon les distributions :

- RedHat based : format RPM, outils rpm, et yum
- Debian based : format deb, outils dpkg, et apt-get ou apt



Paquets façon RedHat

Utilisation

- Recherche textuelle : `yum search mot-clé` Recherche pour un fichier donné : `yum whatprovides */tmux`
- Installation : `yum install paquet`
- Suppression : `yum remove paquet`

Configuration des dépôts distants `/etc/yum.repos.d/*`

Note pour RedHat : les dépôts EPEL augmentent les logiciels disponibles sur RHEL / CentOS.

Paquet `epel-release` sur CentOS.



Paquets façon Debian

Utilisation

- Recherche textuelle : `apt-cache search mot-clé`
Recherche pour un fichier donné : `apt-file search tmutex`
- Installation : `apt install paquet`
- Suppression : `apt remove paquet`

Configuration des dépôts distants `/etc/apt/sources.list` et `sources.list.d/*`

Pour faciliter les recherches et la comparaison des versions,

`http://packages.debian.org` pour Debian, `http://packages.ubuntu.com` pour Ubuntu



Utilisateurs

Héritage UNIX, PAM

Définition d'un utilisateur

- login, uid
- groupes (/etc/group), groupe primaire, groupes secondaires
- mot de passe, fichier shadow
- répertoire \$HOME
- shell par défaut

Les incontournables pour s'en sortir : whoami, id, getent



Permissions

- Rappel des permissions de base
- setuid, setgid sur les fichiers, sur les répertoires
- Suppression restreinte au propriétaire d'un fichier
- ACL étendues

Commandes essentielles : ls, chmod, chown, [gs]etfac



SSH

- Présentation / Rappels pour OpenSSH
- `/etc/ssh/sshd_config` (et `ssh_config`)
- Clés publiques, clés privées, serveur, client
- Notion d'agent de clé

Les incontournables : `ssh`, `ssh-keygen`, `ssh-copy-id`, `ssh-agent`

Sur Windows : Putty, Puttygen, Pageant, plink



vim/emacs/nano

Bases de vi : POSIX. Présent sur tous les UNIX, et intégré dans toutes les distributions.
nano : plus facile, mais pas toujours disponible !

Guide de survie :

- ECHAP : sortir du mode en cours
- i : mode insertion
- :wq write and quit, sauvegarder et quitter
- :q! quit now : quitter sans sauvegarder

Nous utiliserons vim prioritairement pour la formation.



Versioning et git

- 2 Versioning et git
 - Intérêt du suivi de version et des SCM
 - Présentation de git
 - Mise en œuvre locale
 - Dépôts distants



Historique

Importance de pouvoir revenir en arrière, comparer, et suivre l'historique d'un fichier.

cpold

La tentation est grande de recourir à une stratégie de nommage (communément appelée cpold) :

cp fichier fichier.old

cp fichier fichier.old2

cp fichier fichier.old.20170720

cp fichier fichier.old.gilles.20170720

...

Attention aussi à la variante déguisée, qui consiste à garder l'historique sous forme de commentaires au sein des fichiers.



Travail collaboratif

Lorsque l'on travaille à plusieurs sur les mêmes fichiers, il y a 2 aspects importants à prendre en considération :

- L'accès simultané aux fichiers (verrouillage)
- La gestion des conflits le cas échéant

Le premier est en général assez simple à mettre en œuvre, tandis que le second va demander un peu plus d'attention



Git

Git est l'outil utilisé et codé par les développeurs de Linux originalement

- Première version en 2005
- Apporte toutes les fonctionnalités attendues
- Omni-présent, et dispo via github.com



Git

Antisèche rapide

- Création d'un dépôt dans un répertoire donné : `git init repertoire`
- Ajout des fichiers à l'index de git : `git add fichier`
- Création d'une révision : `git commit -m "Commentaire utile"`
- Historique : `git log`

Configuration locale

L'historique présentant les auteurs des révisions, il peut être utile de configurer le nom et l'email associé aux commits locaux via :

```
git config --global user.name "Gilles Pietri"
```

```
git config --global user.email "gpietri@dawan.fr"
```



Retour en arrière avec git

Afin d'éviter le trop classique `cpold`, il faut être assuré de savoir revenir en arrière avec son outil de SCM.

Voici la façon la plus simple de ramener un fichier tel qu'il est sur le dépôt ou à une version antérieure :

`git checkout XXX -- fichier` : restaure le fichier à la version XXX (HEAD par défaut) dans la copie de travail



Git

Aspect serveur

Un « serveur » git est très simple, et se construit le plus facilement avec SSH :

Sur le serveur : `git init --bare mondepot.git`

Sur le client : `git clone login@serveur:mondepot.git`

Gestion des branches distantes : `git remote -v`

Interfaçage et forges logicielles

Même si l'utilisation en mode serveur est très simple, en général les développeurs ont besoin d'associer des outils de gestion et de suivi de projet autour des SCM. On se tournera alors vers des plate-formes comme gitlab, jira, bitbucket, ...



Développement avec Bash

- 3 Développement avec Bash
 - Mise en œuvre de Bash
 - Syntaxe essentielle
 - Entrées/Sorties, Paramètres
 - Manipulations de texte



Bash

- Hashbang / Shebang : `#! /bin/bash`
- Permissions en exécution : `chmod +x monscript`
- Exécution : `./monscript`



Variables

- `variable=valeur`
- `echo $variable`
- `source` et `.`
- `export` / `declare`



Conditions

- tests : test / expr / if
- opérateurs : -f, -d, -z, =, -eq ...
- bloc conditionnel :

```
if [ -f fichier.txt ]; then echo "fichier existe" ; fi
```



Boucles

- boucle for :

```
for fichier in *.txt ;  
do  
    echo "Fichier : $fichier";  
done
```

- boucle while

- instruction read

```
while read ligne;  
do  
    echo "Ligne : $ligne"  
done; < fichier.txt
```



Entrées/Sorties

- Entrée / Sortie standard
- Sortie d'erreur
- Redirections et tubes



Paramètres

- \$0 \$1 ... \$9
- \$@ \$# et shift



Bash "défensif"

Lors de la rédaction de scripts, il peut être prudent de demander à bash d'être plus protecteur. Une bonne manière de se prémunir de comportement bizarre est de s'habituer à quelques options :

- Refuser l'utilisation de variables non définies (-u)
- Quitter le programme en cas d'erreur sur une ligne (-e)
- Renvoyer une erreur si la moindre commande d'une chaîne (pipe) est en erreur :
pipefail

On peut débiter les scripts ainsi protégés comme ceci :

```
set -o nounset -o pipefail -o errexit
```



Mentalité UNIX

La mentalité UNIX dicte entre autres qu'il faut disposer d'un programme pour faire une chose, et la faire bien. C'est pour cela que l'on retrouve une myriade d'outils pour manipuler les fichiers texte directement.

Pour bien appréhender cela, il faut avoir bien compris la notion de redirections, d'entrée et sortie standard.



cut

cut permet de récupérer les colonnes d'un fichier en utilisant un délimiteur particulier.

- -d délimiteur
- -f champs

Utilisation de cut

```
cut -f1 -d: /etc/passwd
```

```
cut -f3,5 -d\; fichier.csv
```



grep

grep permet de chercher une chaîne dans un fichier en utilisant des expressions régulières.

- Rappel des bases d'expression régulières
- grep -c, grep -v, grep -i

Utilisation de grep

```
grep gilles /etc/passwd
```

```
grep -i error /var/log/syslog
```

```
grep -v DEBUG error.log
```



sed

sed s'avère utile pour modifier efficacement un ou plusieurs fichiers en s'appuyant notamment sur les expressions régulières

- Mode substitution
- Mode suppression
- Mode en place

Utilisation de sed

```
sed 's/^\\s *error;/error' smb.conf
```

```
sed '/^$/d' fichier
```

```
sed -i.bak 's/enable/disable/' *.conf
```



Python pour l'administrateur

- 4 Python pour l'administrateur
 - Présentation de Python
 - Joies de Python 2 vs Python 3, Packaging
 - Syntaxe essentielle
 - Entrées/Sorties
 - Modules essentiels, outils pour l'administrateur



Python

Python est un langage très versatile, disponible sur l'ensemble des plateformes courantes (Linux, Windows, MacOS, autres UNIX...). Il peut servir à scripter (ce qui nous intéresse ici), à écrire des applications Web, ou même des applications « lourdes ».



Python

- Python 2.7 : présent partout, maintenu
- Python 3.x : présent partout, maintenu, mais...
- On y arrive.



Scripts Python

- validation version 2 ou 3
- hashbang : `#!/usr/bin/env python`
- `chmod +x` sur Linux et/ou association des `.py` sur Windows



Variables et structures de base

```
booléen = True # ou False  
entier = 10 # Opérateurs classiques disponibles  
reel = 10.8 # 10.0 est donc un réel  
chaine = 'toto' # Ou indifféremment avec des " "  
liste = [1, 2, 'trois', 4.0]  
dico = {'clé1': 25, 'clé2': "toto"}
```



Blocs Python

Python a la particularité d'utiliser l'indentation pour identifier les blocs d'instructions. Les blocs s'ouvrent avec " : " et se terminent par un retour à l'indentation de base.

```
def bloc:  
    # instruction bloc  
    # instruction bloc  
# fin du bloc
```



Conditions

- Opérateurs classiques booléens (==, != ...) et bien sûr **True/False**
- Opérateur d'appartenance **in** (chaîne, liste, clés)
- Bloc déclaré par **if** condition:

Exemple

```
if "--help" in sys.argv:
    print('Documentation : ...')
else:
    print('Programme !')
# Fin du bloc
```



Boucles

- Parcourir « liste » via element : `for element in liste:`
- fonction `range()`
- itérations classiques (listes, dictionnaires, fichiers, ...)

Exemple

```
for fichier in glob.glob('/tmp/*.csv'):  
    print('Fichier : ', fichier)
```



Modules

Utilisation des modules en Python

Python fournit de base énormément de modules pour gérer différents aspects de la programmation.

Utilisation du module sys :

```
import sys
```

Utilisation d'une partie ou une fonction d'un module :

```
from datetime import date
```

Importation de toutes les fonctions / sous-éléments d'un module

```
from monmodule import *
```



Entrées/Sorties

- Paramètres de la ligne de commande : `sys.argv`
- Interaction console : `reponse = input('oui ou non ?')`
- Manipulation de fichiers `with open('fichier.txt', 'r') as f:`
- Lecture entrée standard : `sys.stdin`



Modules de base pour l'administrateur

- sys (argv, std*, path, exit, ...)
- os (chdir, chmod, chown, listdir, popen, ...)
- shutil et glob : manipulation de fichiers en général
- json et yaml : configurations plus lisibles



Présentation de PyPi

PyPi est le gestionnaire de librairies externes pour Python, incluant une gestion des dépendances. Il peut être utilisé pour faire des installations utilisateurs de modules plutôt que sur le système en root.

```
pip search module
```

```
pip install module
```



Modules de PyPi

- Manipulation de paramètres : argparse, docopt, ...
- Templating (configurations, pages Web...) : jinja2
- Accès web / API : requests
- Informations système : psutil
- Pilotage de protocoles niveau 7 : smtplib, twistedmatrix
- Applications web : Flask ou Django

