Mémo Linux

v 4.2

Table des matières

1– Commandes Linux....2
2– Droits....4
3– Gestion des utilisateurs....6
4– Scripts Bash....8
5– Expressions régulières (ou expressions rationnelles)....12
6– grep....14
7– find....15
8– sed....16
9– awk....17
10– Divers....18

1– Commandes Linux

Commandes de bases

`mkdir toto`	Créer le répertoire `toto`
`cd toto`	Entre dans le répertoire `toto`
`cd ..`	Entre dans le répertoire parent
`cd ~`	Entre dans le répertoire d'accueil
`touch a b c`	Créer les fichiers `a`, `b` et `c`
`echo "blabla" > toto`	Créer le fichier `toto` (s'il n'existe pas) contenant le texte « blabla »
`ls`	Affiche le contenu du répertoire courant
`ls *.py`	Idem mais n'affiche que les fichiers d'extension « .py »
`ls -a`	Idem que `ls` + affiche les éléments caches
`ls -l`	Idem que `ls` + affiche les droits : propriétaire + groupe + autre
`ls toto/`	Affiche le contenu de `toto/`
`rmdir toto/`	Efface `toto/` s'il est vide
`rm -r toto/`	Efface le répertoire `toto/`
`rm -r *`	Efface le répertoire courant
`rm a.txt`	Efface le fichier `a.txt`
`cp a.txt b.txt`	Copie `a.txt` sous le nom `b.txt`
`cp -r a/ b/`	Copie `a/` sous le nom `b/`
`mv a.txt b/`	Déplace `a.txt` dans `b/`
`mv a/ b/`	Déplace `a/` dans `b/` (on suppose que `b/` existe)
`mv a.txt b.txt`	Renomme `a.txt` en `b.txt`
`mv a/ b/`	Renomme `a/` en `b/` (ici on suppose que `b/` n'existe pas)
`cat a.txt`	Affiche le contenu du fichier `a.txt`
`tree`	Affiche l'arborescence de racine le répertoire courant (c'est-à-dire le répertoire "`.`")
`tree toto/`	Affiche l'arborescence de racine `toto/`
`vi toto.txt`	Édite le fichier `toto.txt` avec vi

Gestion des paquets

`apt-get update`	Met à jour la liste des paquets disponibles à partir des sources du fichier `/etc/apt/sources.list`
`apt-get upgrade`	Remplace chaque paquet installe par la dernière version disponible.
`apt-get dist-upgrade`	Remplace chaque paquet installé par la dernière version disponible, installe les paquets supplémentaires nécessaires et supprime les paquets devenus inutiles.
`dpkg --status tree`	Le paquet tree est-il installé ?
`apt-cache search web browser`	Recherche d'un navigateur
`apt-cache search tree`	Recherche du paquet tree
`apt-cache search tree | grep tree`	Idem mais n'affiche que les lignes contenant le mot « tree ».
`apt-cache search tree | grep ^tree`	Idem mais n'affiche que les lignes commençant (^) par le mot « tree ».
`apt-cache search tree | grep tree$`	Idem mais n'affiche que les lignes terminant ($) par le mot « tree ».

Outils Unix

`wc`

`wc -l toto.txt`
→ nombre de lignes de toto.txt

`wc -w toto.txt`
→ nombre de mots de toto.txt

`wc -m toto.txt`
→ nombre de caractères de toto.txt

`head` et `tail`

`head toto.txt` → 10 premières lignes de `toto.txt`

`tail toto.txt` → 10 dernieres lignes de `toto.txt`

Exemples de combinaisons avec le « pipe »

```
ps ax | grep firefox

cat toto.txt | wc -l

ls * > toto.txt

cat toto.txt | less

cat toto.txt | grep ^blabla
```

→ afficher uniquement les lignes commençant par ≪ blabla ≫

```
sed s/bonjour/bonsoir/g toto.txt | grep bonsoir > selection_substituee.txt
```

2– Droits

Les permissions peuvent être :

`r` → permission en lecture.

`w` → permission en écriture.

`x` → permission d'exécution pour un fichier, permission d'entrer dans un répertoire.

chmod

`u` → *user* (propriétaire)

`g` → *group* (groupe)

`o` → *other* (autres)

`+` → "ajouter le droit"

`-` → "supprimer le droit"

(`-R` pour affecter récursivement)

Chiffres correspondants aux droits recherchés

➔ pour l'utilisateur :
droits d'accès en lecture : `400`
droits d'accès en écriture : `200`
droits d'accès en exécution : `100`

➔ pour le groupe :
droits d'accès en lecture : `40`
droits d'accès en écriture : `20`
droits d'accès en exécution : `10`

➔ pour les autres :
droits d'accès en lecture : `4`
droits d'accès en écriture: `2`
droits d'accès en exécution : `1`

➔ on additionne ensuite les droits pour chacun

`ls -l` → Les droits s'affichent pour l'utilisateur, le groupe et les autres.

Exemple 1 : droit "ugo"

```
chmod +x toto
```
→ rends exécutable `toto`

```
chmod ug+x toto
```
→ rends exécutable `toto` pour l'utilisateur et le groupe

Exemple 2 : droits avec chiffres

Les droits `rwxr-xr-x` pour `toto.txt équivalent` à :

`400+200+100=700` pour l'utilisateur

`40+10=50` pour le groupe

`4+1=5` pour les autres

Soit au total `700+50+5=755` → `chmod 755 toto.txt`

chown → changer le propriétaire d'un fichier (-R changement récursif)
`chown toto fichier` → « toto » est propriétaire de « fichier »

chgrp → changer le groupe propriétaire d'un fichier (-R changement récursif)
`chgrp toto fichier` → « toto » est le "groupe" de « fichier »

3– Gestion des utilisateurs

`/etc/passwd`

→ tout ce qui concerne la gestion et l'authentification des utilisateurs

`/etc/group`

→ la gestion des groupes

`/etc/shadow`

→ Les mots de passe cryptés sont souvent placés dans ce fichier,
par sécurité lisible seulement par root.

Structure de `/etc/passwd`
Ce fichier comprend 7 champs, séparés par le symbole « : »

- nom de connexion (encore appelé nom d'utilisateur ou login)
- ancienne place du mot de passe crypte
- numéro d'utilisateur **uid**, sa valeur est le véritable identifiant pour le système Linux ;
  l'uid de root est `0`,
  le système attribue conventionnellement un uid à partir de `500` aux comptes créés.
- numéro de groupe **gid**, dans lequel se trouve l'utilisateur par défaut ; le gid de root est `0`,
  les groupes d'utilisateurs au-delà de `500`
- nom complet, il peut être suivi d'une liste de renseignements personnels
- rép. personnel (c'est également le rép. de connexion)
- shell, interpréteur de commandes (par défaut `/bin/bash`)

Structure de `/etc/group`
Ce fichier comprend 4 champs, séparés par le symbole « : »

- nom du groupe
- `x` pour remplacer un mot de passe non attribué maintenant
- numéro de groupe, c-à-d l'identifiant **gid**
- la liste des membres du groupe

`useradd, usermod, userdel`

→ gestion des comptes utilisateur

`groupadd, groupmod, groupdel`

→ gestion des groupes

`passwd`

→ changer le mot de passe d'un utilisateur

useradd → outils de création d'un compte d'utilisateur

```
useradd -g group1 toto
```

→ Créer « toto » de groupe primaire « group1 »

`useradd -G group1 toto`

→ Créer « toto » de groupe secondaire « group1 »

`useradd -G group1,group2 toto`

→ Créer « toto » de groupe secondaire « group1 » et « group2 »

usermod → modifier un utilisateur

« `-l` » → renomme l'utilisateur

« `-g` » →change de groupe

`usermod -g group1 toto`

→ Modification du groupe primaire d'un utilisateur

`usermod -a -G group1 toto`

→ Ajout d'un groupe secondaire a un utilisateur existant

4– Scripts Bash

Substitution et blocage de substitution

`___ $ ma_variable=toto`
`___ $ echo "bonjour $ma_variable"`
`bonjour toto`
	Les guillemets effectuent la substitution, du nom d'une variable préfixée par un « `$` », par son contenu.

`___ $ echo 'bonjour $ma_variable'` `bonjour $ma_variable'`	Les quotes « `'` » bloquent la substitution.

Évaluations

`$(cmde)`	Évalue la commande et affiche son résultat.
`$`cmde``	Idem.
`$((expression))`	Évalue l'expression arithmétique et affiche le résultat.
`$(( x ))`	Convertit `x` en numérique.

Exemples de base

```
nom=toto
echo $toto
```

`1.sh`

```
#!/bin/bash

for i in $( ls ); do
    echo item: $i
done
```

Script Bash uniligne :

```
for i in $( ls ); do echo $i; done
```

`2.sh`

```
#!/bin/bash

for i in {1..50}
do
    mkdir dossier$i
done
```

`3.sh`

```
#!/bin/bash
```

```
mkdir dossier
cd dossier

for i in {1..50}
do
    echo "blabla $i blabla" > fichier_$i.txt
done
```

4.sh

```
#!/bin/bash

read "votre nom : " nom

if [ $nom = "Toto" ]
then
    echo "Bonjour Toto !"
elif [ $nom = "Bobo" ]
then
    echo "Bonjour Bobo !"
elif [ $nom = "Gogo" ]
then
    echo "Bonjour Gogo !"
else
    echo "Bonjour Mr. X !"
fi
```

Conditionnelles

Chaînes de caractères	Nombres	Vrai si
`x = y`	`x -eq y`	$x = y$
`x != y`	`x -ne y`	$x \neq y$
`x < y`	`x -lt y`	$x < y$
`x <= y`	`x -le y`	$x \leq y$
`x > y`	`x -gt y`	$x > y$
`x >= y`	`x -ge y`	$x \geq y$
`-n x`	—	x est non nul
`-z x`	—	x est nul

Opérateur	Vrai si
`-d` ***file***	***file*** existe et est un dossier

`-e` ***file***	***file*** existe
`-f` ***file***	***file*** existe et est un fichier courant
`-r` ***file***	Permission en lecture sur ***file***
`-s` ***file***	***file*** existe et est non-vide
`-w` ***file***	Permission en lecture sur ***file***
file1 `-nt` ***file2***	***file1*** est « newer than » ***file2***
file1 `-ot` ***file2***	***file1*** est « older than » ***file2***

Redirection de la sortie standard

Vers un fichier avec création.	`echo "toto" > mon_fichier.txt`
Vers un fichier avec ajout.	`echo "toto" >> mon_fichier.txt`
Vers une commande.	`echo "toto" | grep "o"`

L'exécution séquentielle et en « multi-tâche »

Exécution séquentielle

On sépare les commandes avec le caractère « ; ».

```
___ $ cat mon_fichier.txt ; rm mon_fichier.txt
```

Exécution en parallèle ou en « multi-tâche ».

On sépare les commandes avec le caractère « & ».

```
___ $ gedit & cat mon_fichier.txt
```

Les arguments

- « `$#` » contient le nombre d'arguments ;
- « `$0` » est le nom de la commande ;
- « `$1` » est le premier argument, « $2 » le second, etc ;
- « `$*` » tous les arguments de celui de rang 1 au dernier.

Fonctions avec arguments

```
#!/bin/bash

maFonction()
{
    echo "\$# : $#"
    echo "\$0 : $0"
    echo "\$1 : $1"
    echo "\$2 : $2"
}


echo "\$# : $#"
echo "\$0 : $0"
echo "\$1 : $1"
echo "\$2 : $2"

echo 'appel: maFonction "toto" "gogo"'
maFonction "toto" "gogo"
```

5– Expressions régulières (ou expressions rationnelles)

Une position pour le motif

`^`	Début de chaîne
`$`	Fin de chaîne
`|`	Ceci ou cela, exemple : `a|b`

Un caractère

`.`	N’importe quel caractère
`[ ]`	Un caractère au choix parmi une liste, exemple : `[ABC]`
`[^ ]`	Tous les caractères sauf…, exemple : `[^@]` tout sauf « `@` »
`[a-zA-Z]`	Toutes les lettres minuscules et majuscules

Des quantificateurs, qui permettent de répéter le caractère qui les précèdent

`*`	Zéro, une ou plusieurs fois
`+`	Une ou plusieurs fois
`?`	Zéro ou une fois
`{ n }`	*n* fois
`{ n, m }`	Entre *n* et *m* fois

Des familles de caractères

`[0-9]`	Un chiffre
`[^0-9]`	Tout sauf un chiffre
`[A-Za-z0-9]`	Un caractère alphanumérique
`\n`	Fin de ligne *(newline)*
`\r`	Retour chariot
`\s`	Un espace (*space*)

6– grep

`grep "mot" toto.txt`	Affiche les lignes contenant « mot » dans toto.txt
`grep "^mot" toto.txt`	Affiche les lignes commençant par « mot »
`grep "mot$" toto.txt`	Affiche les lignes terminant par « mot »
`grep "mot1|mot2" toto.txt`	Affiche les lignes contenant « mot1 » ou « mot2 »
`grep "mot[12]" toto.txt`	Idem que précédemment
`grep "[1-8]" toto.txt`	Affiche les lignes contenant un nombre compris entre 0 et 8
`grep [a-zA-Z] toto.txt`	Affiche les lignes contenant un caractère alphabétique
`grep -r "mot" dossier/`	Affiche les lignes contenant « mot » dans l'arborescence partant de `dossier/`

Diverses options de `grep` :

- `-i` → ne pas tenir compte de la casse (majuscules / minuscules) ;
- `-l` → affiche le nom des fichiers où la correspondance se produit ;
- `-n` → connaître les numéros des lignes ;
- `-v` → inverser la recherche : ignorer un mot.

7– find

`find dossier/ -name "toto"`	Recherche dans l'arborescence `dossier/` les fichiers et répertoires portant le nom « toto ».
`find dossier/ -name "toto*"`	Idem mais le nom à rechercher est « toto » suivit de « n'importe quoi ».
`find dossier/ -name "*toto*"`	Idem mais le nom à rechercher est « n'importe quoi » suivit de « toto » suivit de « n'importe quoi ».
`find dossier/ -name "toto????"`	Idem mais le nom à rechercher est « toto » suivit de 4 caractères (`?`=1 caractère quelconque).
`find dossier/ -iname "toto"`	Recherche dans l'arborescence `dossier/` les fichiers et répertoires portant le nom « toto » sans tenir compte de la casse.

8– sed

`sed s/bonjour/bonsoir/ toto.txt`	Substitue la première occurrence de « bonjour » par « bonsoir » pour toutes les lignes de `toto.txt`
`sed s/bonjour/bonsoir/g toto.txt`	Substitue toutes les occurrences de « bonjour » par « bonsoir » pour toutes les lignes de `toto.txt`
`sed -i s/bonjour/bonsoir/ toto.txt`	L'option « `-i` » permet d'effectuer la substitution sur le fichier `toto.txt`
`___ $ echo 100 300 | sed 's/[0-9]*/42/g'` `42 42`	Remplace tout nombre par la valeur 42
`___ $ echo 100 300 | sed -E 's/([0-9]+)/val: \1/g'` `val: 100 val: 300`	Le motif entouré de parenthèse est utilisé via la notation « `\1` »
`___ $ echo -100- 300 | sed -nE 's/-([0-9]+)-.*/val:\1/gp'` `val:100`	Le motif est trouvé entre la présence de « - » et « - », qui sont ensuite éliminés du résultat

9– awk

Exemple

```
___ $ ls -l mon_fichier
-rwxr-xr-x 1 pef staff 369 Mar 15 11:13 mon_fichier

___ $ ls -l mon_fichier | awk '{ print $7 " " $6 }'
15 Mar
```

- « `$0` » correspond à la ligne complète.
- L'argument de `awk` est mis entre « `'` » pour éviter les substitutions du shell.

Changement de séparateur

```
___ $ echo "nom_prenom" | awk -F_ '{ print $2 " "$1 }'
prenom nom
```

10– Divers

L'arborescence des fichiers – Debian

Filesystem Hierarchy Standard (« norme de la hiérarchie des systèmes de fichiers ») définit l'arborescence et le contenu des principaux répertoires des systèmes de fichiers des systèmes d'exploitation GNU/Linux et de la plupart des systèmes Unix.

Répertoire	**Contenu**
`bin`	Binaires (exécutables) des commandes essentielles.
`boot`	Fichiers statiques pour le programme d'amorçage.
`dev`	Fichiers des pilotes de périphériques.
`etc`	Configuration système propre a la machine.
`home`	Répertoires personnels des utilisateurs.
`lib`	Bibliothèques partagées et modules noyaux essentiels.
`media`	Points de montage pour les supports amovibles.
`mnt`	Point de montage pour les montages temporaires.
`proc`	Répertoire virtuel pour les informations système (noyaux 2.4 et 2.6).
`root`	Répertoire personnel de l'utilisateur *root*.
`sbin`	Exécutables système essentiels.
`sys`	Répertoire virtuel pour les informations système (noyaux 2.6).
`tmp`	Fichiers temporaires.
`usr`	Hiérarchie secondaire.
`var`	Données variables.
`srv`	Données pour les services fournis par le système.
`opt`	Répertoire pour d'autres logiciels.

Raccourcis du terminal Bash

CTR-a	→ début de ligne
CTR-e	→ fin de ligne
CTR-l	→ efface la console
CTR-k	→ efface à droite

Commandes vi/Vim

`ESC`	→ mode commande
`i`	→ insertion
`a`	→ insertion "after"
`:w`	→ sauvegarde (*write*)
`:q`	→ quitter
`:wq`	→ quitter en sauvant
`:q!`	→ quitter sans sauver
`u`	→ *undo*

`CTR+r`	→ *redo*
`v`	→ sélectionne
`y`	→ copie
`p`	→ colle (*paste*)
d	→ couper