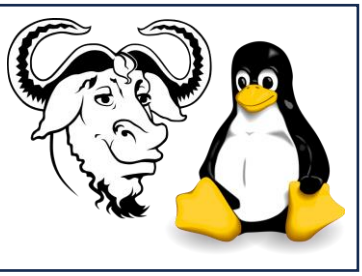
TP n°2 : Rappels et approfondissements



Objectifs du TP :

Utiliser un ordinateur sous Linux, sans interface graphique, uniquement en ligne de commande. Ce TP commence par quelques rappels un peu plus détaillés qui vous permettront de :

- Savoir utiliser les commandes relatives à l'arborescence de fichiers (`cd`, `ls`, `cp`, `rm`, `ln`, ...).

Ensuite, nous introduirons des éléments nouveaux que vous n'avez pas encore abordés :

- Comprendre la notion de droits sur fichiers et répertoires et la commande chmod.
- Savoir utiliser l'éditeur vi.

1. Qu'est-ce que le shell?

Le shell (« coquille ») est un programme « interpréteur de commandes » qui assure l'interface entre l'utilisateur et le « noyau » (kernel) du système d'exploitation. C'est aussi un langage de programmation (programmes = « scripts shell »). Il existe plusieurs langages shell, séparés en deux familles : les Bourne shell et les C shell.

Le shell par défaut sous *Linux* s'appelle **bash** (« **B**ourne **A**gain **Sh**ell »). Le shell attend la saisie d'une ligne de commande et sa validation (touche « Entrée »), pour analyser sa syntaxe puis l'exécuter.

Pour bien connaître le shell et ses commandes, on peut (avec courage !) consulter le manuel :

```
> man bash
```

Les commandes shell doivent être tapées dans un terminal.

- Rappel : dans un terminal, quand le système est prêt à accepter une commande, est affiché une « invite de commande » (Ou « prompt »). Cette invite de commande se termine souvent par un `$` (sauf si vous êtes root, dans ce cas c'est un `#`). D'autres fois les terminaux utilisent le symbole `>`.

Vous pouvez avoir des informations sur le terminal avec la commande :

```
> stty -a
```

Notions d'**entrée, sortie et erreur standard** : ces termes sont souvent utilisés sous Linux :

- L'« entrée standard » par défaut est le clavier. Elle correspond au périphérique où une commande shell lit ses données d'entrée,
- La « sortie standard » par défaut est l'écran. Elle correspond au périphérique où une commande shell écrit ses données de sortie,
- L'« erreur standard » par défaut est aussi l'écran. Elle correspond au périphérique où une commande shell écrit ses messages d'erreur.

2. Quelques commandes générales (rappels et nouvelles commandes)

Dans ce TP, les commandes indiquées par une ☆ doivent être absolument mémorisées. Ce sont des commandes que tout « linuxien » doit connaître sans se référer à la documentation. Ce n'est qu'une indication, en pratique toutes les commandes présentées sont importantes et pourront tomber au contrôle 😊 .

- La commande **echo** : affiche un texte ou la valeur d'une variable sur la sortie standard. Ex. :

☆
```
$ echo Bonjour tout le monde
$ echo « Bonjour tout le monde »
$ echo $PATH
```

- Tapez une commande inexistante pour voir la réaction de Linux à une mauvaise commande. Ex :

```
$ bidon
```

- La commande **who** : affiche qui est connectée sur votre machine.

```
$ who
```

- La commande **hostname** : affiche le nom de votre machine.

☆
```
$ hostname
```

- La commande **whoami** : affiche qui vous êtes (ça a un intérêt si vous pouvez vous connecter sous plusieurs identités distinctes).

```
$ whoami
```

- La commande **date** : affiche la date et l'heure du système.

☆
```
$ date
```

- La commande **which** : localise le chemin du fichier exécutable d'une commande. Ex. :

☆
```
$ which date
```

- La commande **man** : affiche l'aide à propos d'une commande. À l'intérieur des pages d'aide, plusieurs commandes sont possibles :
  - Touche « barre d'espace » pour passer à la page suivante,
  - Touche « b » pour revenir à la page précédente,
  - Taper « /truc » pour rechercher le mot « truc » (touche « n » pour l'occurrence suivante),
  - Touche « q » pour quitter *man*.

Ex. :

☆
```
$ man date
```

- La commande **history** : affiche l'historique des commandes. Connaissant cet historique, plusieurs commandes sont possibles :
  - `!!` : pour rappeler la dernière commande,
  - `!12` : pour rappeler la commande N°12 de l'historique,
  - `!ma` : pour rappeler la dernière commande commençant par « ma »,
  - `!-2` : pour rappeler l'avant-dernière commande.

☆
```
$ history
```

Remarque : on peut aussi rappeler les commandes précédentes à l'aide de la touche « ↑ ».

- La commande **alias** : affiche les « alias » de commandes connus dans votre session. Un alias est un raccourci pour une commande (ex. : on tape *h* plutôt que *history*).

☆
```
$ alias
```

Remarque : on peut créer un alias interactivement (pendant la durée de la session) à l'aide de la commande *alias nom='valeur »*. Nous verrons plus tard dans le TP comment rendre cet alias durable. Ex : créer un alias « h » pour « history »

```
$ alias h='history'
```

3. L'arborescence de fichiers et les commandes liées

Le système de fichiers de *Linux* présente une structure arborescente. La racine de l'arborescence est toujours le répertoire « / » (*root directory*) (voir schéma dans le TP précédent).

Ci-après vous trouverez un rappel rapide des principales commandes de navigation et de manipulation des répertoires et fichiers

Dans les exemples donnés le répertoire courant est */home/isen/tplinux*.

Commande	Description	Exemple	Résultat
ls	Affiche le contenu du répertoire courant	`ls`	`test.txt  tp1`
ls nom_rep	Affiche le contenu du répertoire nommé	`ls /home`	`dupont isen`
ls -l	Affiche le contenu du répertoire courant avec les détails (droits, date de modification, etc.)	`ls -l`	`-rw-rw-r-- 1 isen isen 8 mars 23 11:23 test.txt` `drwxrwxr-x 2 isen isen 4096 mars 23 11:23 tp1`

ls -al	Affiche le contenu d'un répertoire avec les fichiers et répertoires cachés (+détails)	`ls -al ~`	`drwxr-xr-x 21 isen isen 4096 mars 16 15:13 .`
			`drwxr-xr-x 3 root root 4096 févr. 5 14:38 ..`
			`-rw-r--r-- 1 isen isen 220 août 25 2014 .profile`
			`-rw-r--r-- 1 isen isen 3637 août 25 2014 .bashrc`
			`drwxr-xr-x 2 isen isen 4096 août 25 2014 doc`
			`drwxrwxr-x 3 isen isen 4096 mars 23 11:23 tplinux`

pwd	Affiche le nom du répertoire courant	`pwd`	`/home/isen/tplinux`
cd nom_rep	Change le répertoire courant	`cd /tmp`	Le répertoire courant est maintenant /tmp
cd ..	Remonte vers le répertoire parent	`cd ..`	Le répertoire courant est maintenant /home/isen
cd ou cd ~	Se place sur votre répertoire personnel ou home directory, en général /home/nom_user	`cd`	Le répertoire courant est maintenant /home/isen
cd ~nom_user	Se place sur le home directory de l'utilisateur	`cd ~dupont`	Le répertoire courant est maintenant /home/dupont

Commande	Description	Exemple	Résultat
`touch fic`	Crée le fichier (vide) *fic*	`touch test2`	Le fichier vide `test2` est créé dans le répertoire courant
`cp fic1 fic2`	Copie le fichier *fic1* dans un nouveau fichier *fic2*	`cp test2 ../testnew`	Le fichier `testnew`, copie du fichier `test2`, est créé dans le répertoire `/home/isen` (parent)
`cp fic rep`	Copie le fichier *fic* dans le répertoire *rep*	`cp test2 ..`	Le fichier `test2`, copie du fichier `test2`, est créé dans le répertoire `/home/isen` (parent)
`cp -r rep1 rep2`	Copie (récursivement) le contenu du répertoire *rep1* dans un nouveau répertoire *rep2*	`cp -r tp1 tp2`	Le répertoire `tp2`, copie du répertoire `tp1`, est créé dans le répertoire courant. Si `tp1` contient des sous-répertoires et des fichiers, ils sont copiés dans `tp2`.
`mv fic1 fic2`	Renomme le fichier *fic1* en *fic2*	`mv test.txt t.txt`	Le fichier `test.txt` est renommé `t.txt` (son contenu n'est pas modifié)
`Mv rep1 rep2`	Renomme le répertoire *rep1* en *rep2*	`mv tp1 tp0`	Le répertoire `tp1` est renommé `tp0` (son contenu n'est pas modifié)
`mv fic rep`	Déplace le fichier *fic* dans le répertoire *rep*	`mv t.txt tp1`	Le fichier `t.txt` est déplacé dans le répertoire `tp1` (il n'existe plus dans le répertoire courant)
`rm fic`	Détruit le fichier *fic*	`rm test2`	Le fichier `test2` est détruit (irréversible : pas de corbeille!)
`rm -r rep`	Détruit le répertoire *rep* et son contenu	`rm -r tp2`	Le répertoire `tp2` est détruit et tous les fichiers et sous-répertoires contenus (récursivement). **Danger!**
`mkdir rep`	Crée un nouveau répertoire (vide) *rep*	`mkdir tp2`	Le répertoire `tp2` est créé dans le répertoire courant
`rmdir rep`	Détruit le répertoire *rep* (autorisé si vide)	`rmdir tp2`	Le répertoire `tp2` est détruit. Ne fonctionne pas si `tp2` n'est pas vide, faire `rm -r` dans ce cas.

Rappel : fichiers et répertoires cachés :

Certains fichiers et répertoires ont un nom commençant par « . » (point). Ils sont en général situés dans le « *home directory* » de chaque utilisateur. Ce sont des fichiers de configuration personnels (ou des répertoires contenant des fichiers de configuration). Dans l'exemple d'arborescence ci-dessus, on trouve les fichiers cachés *.bashrc* et *.profile*.

On dit qu'ils sont cachés, car ils ne sont pas visibles par la commande « *ls* » simple. Pour les voir, il faut ajouter à « *ls* » l'option « *-a* » (cf. tableau ci-dessus).

On peut cacher n'importe quel fichier ou dossier en le renommant et en rajoutant un point devant.

4. Substitution de « méta-caractères » dans les noms de fichiers et répertoires

Quand on spécifie des noms de fichiers (ou des chemins), *Linux* permet de substituer une partie du nom par un métacaractère ou caractère « joker » (*wildcard*). Ces métacaractères sont :

« * » : correspond à n'importe quelle chaîne de caractères, y compris la chaîne vide,

« ? » : correspond à n'importe quel caractère,

[plusieurs caractères] : correspond à un des caractères entre les crochets.

Ex :

`[ABab]` correspond à un « a » ou un « b » minuscule ou majuscule. On peut aussi définir un intervalle.

Ex. :

`[a-z]` correspond à un caractère minuscule alphabétique quelconque.

Dernière possibilité : le complément `[!...]`

Ex :

`[!abc]` correspond à n'importe quel caractère sauf « a », « b » ou « c » minuscules.

Le développement (*globbing*) est l'opération qui transforme un motif générique (=nom contenant des métacaractères) en une liste de fichiers correspondant à ce motif.

Remarque 1 : par défaut (c'est-à-dire si vous n'avez pas changé les options de « *globbing* »), les métacaractères * et ? ne se substituent pas à un « . » (point) au début d'un nom de fichier. Cela implique que pour les fichiers cachés, le « . » doit toujours être écrit explicitement.

Remarque 2 : la recherche de correspondances par « *globbing* » ne doit pas être confondue avec la recherche par expression régulière (*regex*), qui n'a pas du tout les mêmes conventions pour la signification des métacaractères. Nous verrons le regex ultérieurement.

5. Autres commandes de manipulation des répertoires et fichiers

5.1. Écriture dans un fichier texte sans ouvrir d'éditeur

Pour écrire des textes simples dans un fichier, il n'est pas nécessaire de passer par un éditeur. Il existe deux commandes appropriées pour réaliser cette action :

- La commande `echo "texte " > fichier` crée un fichier contenant « *texte* » (cette commande, et la suivante font intervenir la notion de redirection de sortie que nous verrons lors du TP2). Ex :

```
$ echo "bla
bla
bla" > test.txt
```

- La commande `cat > fichier <saut de ligne> texte <Ctrl D>` crée aussi un fichier contenant « *texte* » (ici, *<saut de ligne>* signifie que vous avez tapé sur la touche « *Entrée* » et *<Ctrl D>* signifie que vous avez tapé en même temps sur les touches « *Control* » et « *D* »).

```
$ cat > test2.txt
bla
bla
bla<Ctrl-D>
```

5.2. Lien symbolique vers un fichier (ou un répertoire)

Linux permet d'avoir plusieurs noms pour un même fichier (ou répertoire), un peu de la même façon que les « raccourcis » sous Windows. Ce « raccourci » s'appelle un **lien** vers un fichier (ou un répertoire). Ce lien peut être matériel (*hard link*) ou symbolique (*symbolic link* ou *symlink*). Le lien symbolique est le plus utilisé, car le lien matériel a plusieurs limitations et inconvénients.

Le *symlink* apparaît pour le système comme un fichier d'un type particulier. Le résultat de « *ls -l* » fait apparaître un « `l` » en premier caractère au lieu d'un « `-` » (fichier) ou « `d` » (répertoire), et le nom du fichier (ou répertoire) réel « ciblé » est indiqué en fin de ligne, après le nom du *symlink*.

On dit qu'on « déréférence » le *symlink* quand on accède au fichier (ou répertoire) réel « ciblé » par le lien. On peut faire sur un *symlink* toutes les commandes comme sur un fichier « normal » :

- La plupart des opérations (ouverture, lecture, écriture) sur un *symlink* le déréférencent automatiquement et opèrent sur la « cible » du lien (le fichier ou répertoire réel).
- Par contre, la suppression (*rm*) ou le déplacement/renommage (*mv*) portent sur le lien lui-même et n'affectent pas le fichier « cible ». Si on supprime le *symlink*, le fichier « cible » n'est pas supprimé.
- Si on supprime le fichier « cible », le *symlink* n'est pas supprimé, mais il ne pointe plus sur rien. On dit que le «*lien est cassé*» (*broken link*). Une tentative d'ouvrir en lecture un lien cassé conduit à un message d'erreur de type « fichier non trouvé ».

La création d'un lien symbolique se fait par la commande *ln* (*link*) avec l'option *-s*

(*symbolic*). La forme la plus fréquente de la commande est :

`ln -s fic_ou_rep_cible nom_du_lien`

Exemples :

- « `ln -s fic lien` » crée le lien `lien` vers le fichier « cible » `fic`. Ex : ici on crée *test1.txt*, qui est un fichier (fictif) dans le répertoire courant qui pointe vers le fichier réel *tp1/test1* :

```
$ ln -s tp1/test1 test1.txt
```

La commande « `ls -l` » va afficher :

```
lrwxrwxrwx  1 isen isen  9 mars 24 17:45 test1.txt -> tp1/test1
```

Si le lien est cassé, par ex. si on fait « *rm tp1/test1* », la commande « *ls -l* » va afficher :

```
lrwxrwxrwx  1 isen isen  9 mars 24 17:45 test1.txt -> tp1/test1
```

C'est-à-dire que seule la couleur rouge/fond noir (sous *Ubuntu* ou *Debian*) indique que le fichier « cible » n'existe plus.

- « `ln -s rep lien` » crée le lien *lien* vers le répertoire `rep`. Ex. : ici on crée *documents*, qui est un sous-répertoire (fictif) dans le répertoire courant qui pointe vers le répertoire réel *../doc* (répertoire frère de *tplinux*) :

```
$ ln -s ../doc documents
```

La commande « *ls -l* » va afficher :

```
lrwxrwxrwx  1 isen isen  9 mars 24 18:01 documents -> ../doc/
```

5.3. Quelques autres commandes sur les fichiers

- La commande ***du*** (*disk usage*) : affiche l'espace disque pris par chaque répertoire, récursivement en descendant à partir du répertoire courant. Avec l'option *-a* (*all*), elle affiche aussi les fichiers contenus dans chaque répertoire. L'option *-h* (*human readable*) permet d'afficher les tailles de façon plus lisible (ex. 1K 23M 2G) au lieu d'afficher en nombre de blocs. Ex. :

```
$ du -ha
```

Le résultat de la commande va ressembler à :

```
4,0K  ./tp1/test1
8,0K  ./tp1
0     ./test1.txt
0     ./tp2/test1
4,0K  ./tp2
4,0K  ./test.txt
```

```
20K    .
```

- La commande ***wc*** (*word count*) : Elle sert à compter les lignes (option *-l*), les mots (option *-w*) ou les caractères (option *-c*) dans un fichier. Elle est le plus souvent utilisée pour compter les lignes. Sans option, elle compte les trois. Ex. :

```
$ wc -l test.txt
```

- La commande ***file*** : sert à connaître le type du fichier. Le résultat de la commande indique précisément le type de contenu du fichier (ex. : *ASCII text*) . Ex. :

```
$ file test.txt
```

- La commande ***diff*** (différence) : permet de trouver les différences entre deux fichiers ou deux répertoires.
  - « ***diff fic1 fic2*** » affiche les lignes différentes entre les fichiers *fic1* et *fic2*.

```
$ diff test1.txt test2.txt
```

Exemple détaillé :

```
$ echo "Un
2
Trois
4" > f1.txt
$ echo "Un
Deux
Trois" > f2.txt
$ diff f1.txt f2.txt
2c2
< 2
---
> Deux
4d3
< 4
```

Explication :

A chaque différence trouvée, *diff* résume la différence de la façon suivante :

« N° ligne 1er fichier » « une lettre (a : *add*, c : *change* ou d : *delete*) » « N° ligne 2d fichier ». Ex. :

`2c2` signifie : la ligne 2 de f1 doit être changée pour correspondre à la ligne 2 de f2

`4d3` signifie : la ligne 4 de f1 doit être détruite pour correspondre après la ligne 3 de

f2

Ensuite, les différences sont indiquées :

La ligne concernée par une différence est affichée à la suite d'un < pour le 1er fichier et d'un > pour le 2d fichier, éventuellement séparés par une ligne `- - -`.

Remarque : il existe une variante de la commande qui affiche les différences de façon plus synthétique. Il faut ajouter l'option *-u* (*unified*) :

```
$ diff -u f1.txt f2.txt
--- f1.txt 2015-03-25 14:44:05.337899491 +0100
+++ f2.txt 2015-03-25 14:44:40.879214900 +0100
@@ -1,4 +1,3 @@
Un
-2
+Deux
Trois
-4
```

- « `diff rep1 rep2` » affiche les fichiers différents entre les répertoires *rep1* et *rep2*, et s'il trouve des fichiers de même nom, il les compare ligne à ligne.

```
$ diff tp1 tp2
```

- La commande **cat** déjà vue précédemment pour afficher le contenu d'un fichier ou créer un nouveau fichier peut servir aussi à **concaténer** deux fichiers (ou plus) à l'aide d'une redirection de sortie (cf. TP suivant). Ex. :

☆
```
$ cat test1.txt test2.txt > test1plus2.txt
```

- La commande **find** sert à chercher un(des) fichier(s) à partir d'un point dans l'arborescence. Elle peut avoir une syntaxe assez complexe (on peut préciser une action à exécuter sur tous les fichiers trouvés, avec l'option `-exec`).

Ex. 1 : la commande suivante permet de chercher tous les fichiers se terminant par « *.sh* » en parcourant l'arborescence à partir de « */etc* » :

☆
```
$ find /etc -name "*.sh"
```

Ex. 2 : la commande suivante permet de chercher tous les fichiers « normaux » (`-type f`) en parcourant l'arborescence à partir du répertoire courant « `.` » et, pour chacun, d'afficher le type de son contenu (`-exec file {} \;`), cf. plus haut commande *file* :

☆
```
$ find . -type f -exec file {} \;
```

Ex. de résultat de cette commande, dans le répertoire courant */home/isen/tplinux* :

```
./tp1/test1 : ASCII text
./tp2/test1: empty
./test.txt: ASCII text
```

6. Droits d'accès aux fichiers et répertoires

6.1. Notion de propriétaire (owner)

Tout fichier ou répertoire *Linux*/*Unix* possède un propriétaire. Au départ, c'est l'utilisateur qui a créé le fichier, mais *root* peut le « donner » à un autre utilisateur.

Un fichier ou répertoire appartient aussi à un groupe. On définit ainsi les possibilités d'action du groupe sur ce fichier. Ce groupe est souvent le groupe d'appartenance du propriétaire, mais ce n'est pas obligatoire.

Une règle générale simple s'applique à la création de tout nouveau fichier (ou répertoire) :

1. Son propriétaire est l'utilisateur qui l'a créé,
2. Son groupe est le groupe primaire de ce même utilisateur.

Sous *Linux*, seul *root* peut changer le propriétaire d'un fichier.

6.2. Notion de droits (permissions)

À chaque fichier ou répertoire est associée une liste de droits ou *permissions*, qui déterminent ce que chaque utilisateur a le droit de faire du fichier.

Seul le propriétaire du fichier et *root* peuvent changer les droits.

6.3. Catégories d'ayants droit

Pour chaque fichier, le monde de ses utilisateurs potentiels est partagé en trois catégories :

1. **u** : l'**u**tilisateur (*user*) propriétaire, bien souvent son créateur, mais qui n'a pas forcément tous les droits sur lui !
2. **g** : son **g**roupe propriétaire, ensemble d'utilisateurs auxquels on peut accorder des droits différents des autres (attention, il n'inclut pas l'utilisateur lui-même).
3. : les autres (*others*) utilisateurs (c-à-d qui ne font pas partie du groupe précédent).

6.4. Les droits classiques (norme POSIX)

Il y en a trois, représentés par les lettres **r**, **w** et **x** (appelés *flag*). Ils ne sont pas récursifs (ne s'appliquent pas aux sous-répertoires). En voici la signification :

Symbole	Sur un fichier normal	Sur un répertoire
-	Absence de droit	Absence de droit

r	Lecture (lire, copier le contenu)	Visibilité (listage) des fichiers directement contenus
w	Écriture (modifier le contenu)	Création, renommage et destruction des fichiers directement contenus (même si on n'a pas de droit d'écriture sur les fichiers)
x	Exécution	Entrée dans le répertoire (avec *cd*)

6.5. Représentation des droits

Cet ensemble de 3 droits sur 3 catégories se représente de la façon suivante : on écrit côte à côte les droits r, w puis x pour le propriétaire (u), le groupe (g) et les autres utilisateurs (o). Lorsqu'un *flag* est attribué à une catégorie, on l'écrit (r, w ou x), sinon on écrit « - ». Ex. :

`rwxr-xr--` signifie droits r, w et x pour le propriétaire, r et x pour le groupe et r pour les autres

De plus, une première lettre, apparaissant avant les 9 lettres ci-dessus, indique le type de fichier. En voici la liste (les deux premiers sont les plus fréquents) :

`-` : fichier classique (régulier)

`d` : répertoire

`l` : lien symbolique

`c` : périphérique de type caractère

`b` : périphérique de type bloc

`p` : fifo (*pipe*)

`s` : socket

Pour connaître les droits sur un fichier, on utilise la commande `ls -l` (ou l'alias `ll`), ex :

```
$ ls -l test.txt
-rw-r—r-- 1 isen isen 124 jan 22 2014 test.txt
```

Les informations qui apparaissent après les droits sont :

- Nombre de liens sur le fichier : 1 signifie que le fichier n'a aucun lien matériel (*hard link*) qui pointe vers lui, 2 (ou plus) qu'il existe un lien matériel (ou plus) vers lui,
- Le nom du propriétaire du fichier,
- Le nom du groupe propriétaire,
- La taille du fichier (en octets),

- La date de dernière modification,
- Le nom du fichier.

Remarque : la commande *stat* permet d'obtenir plus d'informations sur un fichier (ou répertoire) que `ls -l`. Ex :

```
$ stat test.txt
```

La notation octale

C'est une autre façon commune pour représenter les droits. La notation octale consiste en une valeur numérique en base 8, sur 3 (ou 4) *digits*.

On définit un codage binaire où `rwx` correspond à `111` (= 7 en octal). Il suffit donc de déclarer pour chacune des catégories d'utilisateur (*user*, *group*, *others*) un chiffre entre 0 et 7 auquel correspond une séquence de droits d'accès. Par exemple :

- `777` donne `111 111 111` soit `rwx rwx rwx`
- `605` donne `110 000 101` soit `rw- --- r-x`
- `644` donne `110 100 100` soit `rw- r-- r--`
- `666` donne `110 110 110` soit `rw- rw- rw-`

Voici un résumé de la signification de chaque valeur octale (un *digit*) :

0 --- no permission

1 --x execute

2 -w- write

3 -wx write and execute

4 r-- read

5 r-x read and execute

6 rw- read and write

7 rwx read, write and execute

Remarque : Une astuce permet d'associer rapidement une valeur à la séquence de droits souhaitée. Il suffit d'attribuer les valeurs suivantes pour chaque type de droit :

- Lecture (`r`) correspond à 4
- Écriture (`w`) correspond à 2
- Exécution (`x`) correspond à 1

Puis on additionne ces valeurs selon qu'on attribue ou non le droit correspondant. Par exemple, rwx vaut 7 (4+2+1), r-x vaut 5 (4+1) et r-- vaut 4.

6.6. Commandes de changement des droits

Changer les droits (*permissions*) sur les fichiers et répertoires :

chmod [options] *mode*[,*mode*] fichiers

Le paramètre *mode* permet de déterminer les nouveaux droits d'accès. On peut agir de façon absolue, en fixant les nouveaux droits qui remplacent les anciens (symbole = ou notation octale), ou bien agir par ajout (+) ou retrait (–) par rapport aux droits existants.

La syntaxe de *mode* suit un des deux modèles suivants :

1. **Symbolique : [u g o a] [+ - =] [r w x]** avec :
   a. ***u, g, o*** : les 3 catégories d'utilisateurs (*user*, *group*, *others*) et ***a*** (*all* =*ugo*). Par défaut *a* (mais en tenant compte du *umask*, cf. plus bas).
   b. **+ - =** : l'action d'ajouter, de retirer ou de fixer un droit, qui s'applique à chaque catégorie séparément.
   c. ***r, w, x*** : les 3 attributs pour chaque catégorie d'utilisateur.
2. **Octal : ddd** ou **dddd** où d est un chiffre octal (0 à 7)

La principale **option** est ***-R*** (Récursif) : elle permet de modifier les droits de tous les sous-répertoires et de leur contenu.

Exemples :

`$ chmod a+r fic`	Droit de lecture ajouté pour tous
`$ chmod a-x fic`	Droit d'exécution enlevé pour tous
`$ chmod a+rw fic`	Droit de lecture et d'écriture ajouté pour tous
`$ chmod ug+rw fic`	Droit de lecture et d'écriture ajouté pour le propriétaire et le groupe
`$ chmod ug=rx fic`	Met les droits de lecture et d'exécution seulement (pas d'écriture) pour le propriétaire et le groupe
`$ chmod u=rw,go= fic` `$ chmod 600 fic`	Met les droits de lecture et d'écriture pour le propriétaire, tous les droits sont enlevés pour tout autre utilisateur
`$ chmod a-rwx fic` `$ chmod 000 fic`	Enlève tous les droits pour tout le monde
`$ chmod +rwx fic`	Ajout des droits rwx pour tout le monde (mais avec application *umask*, cf. §4.7 ci-après). Différent de a+rwx ou de a=rwx.
`$ chmod -R u+w,go-w rep`	Change les droits du répertoire *rep* et de l'arborescence sous *rep* pour rajouter un accès en écriture pour le propriétaire, et interdire l'accès en écriture pour tout autre utilisateur

Exercices :

- Créer un fichier *test* (vide ou non). Quels sont ses droits (notation caractère et octale) ?
- Lui rajouter les droits d'exécution pour tous,
- Enlever tous les droits pour le groupe. Quels sont maintenant ses droits (notation caractère et octale) ?
- Écrire l'équivalent de « `chmod u=rwx,g=rw,o=r test` » en notation octale. Appliquer.
- Faire « `chmod g=r test` ». Quels sont maintenant les droits de *test* (notation caractère et octale) ?
- Remettre sur *test* les mêmes droits qu'à sa création.
- Créer un répertoire *rep1*, puis, sous celui-ci, un répertoire *rep2* et un fichier *test2*. Quels sont les droits de *rep1* et *rep2* (notation caractère et octale) ?
- Avec un *chmod* récursif, changer tous les droits de *rep1* et de son contenu : enlever tous les droits au groupe et aux autres. Vérifiez.

6.7. Droits à la création

A la création d'un fichier ou d'un répertoire, des droits ***par défaut*** sont attribués à l'utilisateur propriétaire, au groupe propriétaire et aux autres.

Le principe en est le suivant : les droits maximums accordés sont 666 (`rw-rw-rw-`) pour un fichier et 777 (`rwxrwxrwx`) pour un répertoire. Sur ces valeurs, on applique un « masque » qui est défini dans un fichier de configuration : c'est le rôle de la commande **`umask`** (*user mask*).

Les droits résultants sont calculés par soustraction du masque aux droits maximum. Par exemple, si le masque est 027 en octal :

1. Création d'un répertoire :

```
  777 = 111 111 111 droits max =     rwx rwx rwx
- 027 = 000 010 111 masque =        --- -w- rwx
= 750 = 111 101 000 droits effectifs = rwx r-x ---
```

2. Création d'un fichier :

```
  666 = 110 110 110 droits max =     rw- rw- rw-
- 027 = 000 010 111 masque =        --- -w- rwx
= 640 = 110 100 000 droits effectifs = rw- r-- ---
```

- La commande ***umask*** :
  - « *umask* » seul affiche le masque actif dans votre session :

```
$ umask
```

  - « *umask masque* » modifie le masque pour la durée de la session. Ex :

```
$ umask 027
```

- Masque par défaut

Sous *Linux*, le masque par défaut est fixé dans */etc/bashrc* (à 022 ou 002) pour les distributions *Red Hat* (*Centos*, *Fedora*), dans */etc/profile* (à 022) pour les distributions *Debian (Ubuntu)*.

Chaque utilisateur peut changer (de façon durable) ce masque par défaut : il faut

ajouter la ligne `umask masque` dans le fichier de configuration personnel `~/.bash_profile` (ou `~/.profile`).

Exercice :

- Quel est le masque actif dans votre session?
- Modifier ce masque pour 027. Testez la création d'un fichier et d'un répertoire.

6.8. Les droits spéciaux : suid, sgid, sticky bit

Il existe d'autres droits spéciaux (dits aussi *étendus*), rendant possible une gestion plus poussée des permissions.

6.8.1. Droit suid (ou setuid) :

Ce droit s'applique aux fichiers exécutables, il permet de donner temporairement à un utilisateur les droits du propriétaire du fichier, durant son exécution (et non les droits de l'utilisateur qui le lance). Le nom *suid* ou *setuid* est une abréviation de « ***set user id** upon execution* ».

Par exemple, le programme *passwd*, qui permet à un utilisateur de modifier son mot de passe, appartient à l'utilisateur *root* et possède le droit *suid*. En effet, ce programme doit pouvoir écrire dans le fichier */etc/passwd* (ou */etc/shadow*), dans lequel seul *root* peut écrire.

Représentation :

Dans la représentation des droits avec la notation « caractère », un fichier portant le droit *suid* est représenté par un `s` (ou `S`) à la place du x du propriétaire : par ex : `-rwsr-xr-x`.

Dans la notation octale, tous les droits spéciaux apparaissent sur un premier *digit* (avant celui du propriétaire). Le droit *suid* vaut `4` : les droits en octal seront `4755` pour l'exemple ci-dessus.

Exercice : trouver le fichier exécutable de la commande *passwd* et voir ses droits.

6.8.2. Droit *sgid* (ou *setgid*)

Ce droit peut s'appliquer soit aux fichiers exécutables, soit aux répertoires.

1. Sur un fichier exécutable :

Il fonctionne de la même façon que *suid*, mais pour un groupe. Le nom *sgid* ou *setgid* est une abréviation de « ***set group id** upon execution* ».

2. Sur un répertoire :

Il signifie quelque chose de complètement différent : quand un répertoire a le droit *sgid*, tous les fichiers (et sous-répertoires) créés dans ce répertoire appartiendront par défaut au même groupe que le répertoire.

Représentation :

Dans la représentation des droits avec la notation « caractère », un fichier ou répertoire portant le droit *sgid* est représenté par un `s` (ou `S`) à la place du `x` du groupe : par ex : `-rwxr-sr-x`.

Le droit *sgid* vaut `2` : les droits en octal seront `2755` pour l'exemple ci-dessus.

Exercice : trouver le fichier exécutable de la commande *wall* et voir ses droits.

6.8.3. Droit « sticky bit »

Le droit « *sticky bit* » (« bit collant ») s'applique (sous *Linux*) uniquement aux répertoires.

Quand un répertoire a le droit « *sticky bit* », les utilisateurs ayant le droit *w* sur le répertoire ne peuvent pas détruire (ou renommer) les fichiers qui ne leur appartiennent pas (contrairement au comportement normal).

Représentation :

Dans la représentation des droits avec la notation « caractère », un fichier ou répertoire portant le droit « *sticky bit* » est représenté par un `t` (ou `T`) à la place du `x` des autres (*others*) : par ex : `drwxr-xr-t`.

Le droit « *sticky bit* » vaut `1` : les droits en octal seront `1755` pour l'exemple ci-dessus.

Exercice : voir les droits du répertoire */tmp*.

6.9. Changement de propriétaire

La propriété d'un fichier (ou répertoire) est accordée au niveau d'un utilisateur (*user*) et d'un groupe. Le changement de propriétaire peut se faire indépendamment pour l'utilisateur et le groupe.

- Changer l'utilisateur propriétaire

```
chown [options] user[:groupe] fichier_ou_rep
```

Sous *Linux* cette commande ne peut être lancée que par *root*. Par défaut (si le groupe n'est pas précisé), cette commande laisse le groupe propriétaire inchangé.

La principale option est *-R* (Récursif) : elle permet de modifier les propriétaires de tous les sous-répertoires et de leur contenu.

Exemple :

```
$ sudo chown -R isen:isen /tmp/backup
```

- Changer le groupe propriétaire

```
chgrp [options] groupe_dest fichier_ou_rep
```

Cette commande peut être lancée par *root* mais aussi par le propriétaire s'il fait partie du groupe de destination.

La principale option est *-R* (Récursif) : elle permet de modifier les groupes propriétaires de tous les sous-répertoires et de leur contenu.

Exemple :

```
$ chgrp -R vboxsf /media/sf_partage/test
```

7. Éditer un fichier

Dans le TP « *Commandes Linux : 1ère approche* », l'éditeur en « mode texte » qui vous a été présenté était *nano*, choisi pour sa simplicité. Cependant, selon les distributions *Linux/Unix*, *nano* n'est pas toujours disponible. Par contre, sauf exception, vous trouverez toujours *vi* (ou sa version améliorée *Vim*).

Nous allons donc dans ce TP apprendre à utiliser *vi*. Sur la plupart des distributions *Linux*, on trouve plutôt *Vim* (*Vi IMproved*) qui est un peu plus convivial que *vi* de base.

```
$ vi test.txt
```

7.1. Guide de survie pour l'éditeur vi (ou Vim)

Il faut savoir que *vi* à l'origine a été conçu pour être utilisé avec des claviers très restreints (pas de touches flèches, pas de touche *Control* ou *Alt*, ni *Inser* ou *Suppr*, etc.) : en gros, toutes les commandes *vi* se font à l'aide des touches alphabétiques, des touches « `:` », « `!` », « `/` » et « `Echap` ». De plus, un grand nombre de commandes ont un seul caractère (ce qui ne facilite pas toujours leur mémorisation !).

Lorsque l'on débute avec *vi*, il est primordial de comprendre que *vi* possède deux modes de fonctionnement principaux : le mode « commande » ou « normal » (dans lequel *vi* démarre) et le mode « insertion ».

Pour passer du mode « commande » au mode « insertion » : touche « `i` »,

Pour passer du mode « insertion » au mode « commande » : touche « **Echap** ».

La frappe d'une touche donnée (par exemple « `x` ») aura une signification différente selon le mode dans lequel vous vous trouvez : si vous êtes en mode « commande », la touche « `x` » va effacer le caractère sous le curseur, si vous êtes en mode « insertion » elle va insérer la lettre « `x` » à l'endroit où est le curseur.

Quelques commandes essentielles (en mode « commande ») :

- Touches pour se déplacer : `h`, `j`, `k`, `l` (gauche, bas, haut, droite). En fait, dans les versions récentes de *vi* (et dans *Vim*), les flèches (←,↓,↑,→) fonctionnent.
- Commandes d'effacement : `x` (sous le curseur), `dd` (toute la ligne du curseur).
- Commandes d'insertion (elles font toutes passer en mode « insertion », mais en démarrant différemment par rapport au curseur) : `i` (insertion avant le curseur), `a` (insertion après le curseur), `o` (insertion sur une ligne vide créée sous le curseur).
- Commandes d'enregistrement du fichier et de sortie de *vi* :
- `:q` (sortie sans enregistrer), `:q!` (sortie sans enregistrer même si le fichier a été modifié), `:wq` (sortie en enregistrant), `:w` (enregistrer sans sortir).
- Recherche de chaînes de caractères : /chaîne cherche la première occurrence de chaîne après le curseur, touche n pour occurrence suivante.
  Ex : on veut rechercher la chaîne isen : `/isen`

Quelques options utiles (à activer en mode « commande ») :

La plupart des options sont des booléens (*on* ou *off*). Pour changer les options, on

utilise la commande `:set` (peut être raccourcie en `:se`).

- Option pour afficher les numéros de lignes : **`:set number`** (raccourci « `:se nu` »). Pour les masquer : `:set nonumber` (raccourci « `:se nonu` »).
- Option pour afficher le mode : **`:set showmode`** (raccourci « `:se smd` »). Pour le masquer : `:set noshowmode` (raccourci « `:se nosmd` »). Cette option ne fonctionne pas sur certaines versions.

7.2. Tutoriel pour l'éditeur Vim

Tapez la commande `$ vimtutor`. Si elle n'existe pas, il faut installer *vimtutor*, avec la commande suivante :

- sous *Ubuntu* et autres distributions *Debian* :

```
$ sudo apt install vim vim-common vim-runtime
```

- sous *Fedora*, *Centos* et autres distributions *RedHat* :

```
$ sudo yum install vim-common vim-minimal vim-enhanced vim-X11
```

Ce tutoriel en français est très bien fait et très complet.

7.3. Annexe : rajouter un utilisateur dans les « sudoers »

La commande *sudo* permet de changer d'identité (généralement pour passer « super-utilisateur ») uniquement pendant le temps d'exécution d'une commande. Le nom *sudo* est une abréviation de « ***s**ubstitute **u**ser **do*** », c'est-à-dire « exécuter en se substituant à l'utilisateur ».

Il est considéré comme une bonne pratique, quand on doit lancer une commande nécessitant des droits super-utilisateur (*root*), de le faire avec *sudo*, et pas en se connectant en tant que *root* puis lancer la commande. C'est-à-dire qu'il vaut mieux faire :

```
$ sudo commande
```

Plutôt que :

```
$ su -
# commande
# exit
```

Pour être autorisé à lancer une commande avec sudo, un utilisateur doit faire partie d'un groupe autorisé (selon les distributions : *wheel*, *sudo*, *admin*,...) ou être autorisé à titre personnel.

La configuration de *sudo*, et notamment les groupes et *users* autorisés, est stockée dans le fichier */etc/sudoers* (qui n'est modifiable que par *root*). Il pourrait être modifié à l'aide d'un éditeur quelconque, mais il existe un outil spécialisé, *visudo*, qui doit être préféré, car il vérifie les modifications.

Dans certaines distributions, notamment *Ubuntu*, l'utilisateur qui a installé le

système est mis automatiquement dans les « *sudoers* », dans d'autres distributions (*Debian*, *Centos*,…) ce n'est pas le cas. Les manipulations indiquées ci-dessous ont pour but de vous mettre dans les « *sudoers* », si vous ne l'êtes pas déjà.

Les commandes décrites ci-dessous consistent à : 1) vérifier si vous (utilisateur *isen*) êtes listé dans les « *sudoers* », 2) sinon rajouter *isen* dans les « *sudoers* », 3) tester une commande avec *sudo*.

- Vérifier si vous êtes listé dans les « *sudoers* » :

```
$ sudo -l
```

- Si vous n'êtes pas listé pas dans les « *sudoers* », vous devez vous passer *root* pour effectuer les modifications nécessaires :

```
# su -
# visudo
```

visudo ouvre le fichier */etc/sudoers* avec un éditeur *vi*. Vous devez parcourir le fichier jusqu'à trouver une ligne contenant `%wheel ALL=(ALL) ALL` (*Centos*, *Fedora*). Si cette ligne commence par # (marque de commentaire), effacez ce # (`x` pour effacer), puis sauvez (`:wq`). Cette modification donne tous les droits pour le groupe *wheel*. NB: si vous êtes **sous *Debian*, ce n'est pas le groupe *wheel* mais le groupe *sudo***.

Vous devez ensuite (toujours en tant que *root*) ajouter l'utilisateur *isen* dans le groupe *wheel*, puis revenir en tant qu'utilisateur *isen*.

```
# usermod -a -G wheel isen
# exit
```

- Tester une commande avec *sudo*, ex (commande impossible sans les droits *root*) :

```
$ sudo cat /etc/shadow
```