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

Linux : TP 3

Halim Djerroud

révision 1.1

Convention : Nous travaillerons dans un dossier dédié du répertoire personnel :

```
mkdir -p ~/tp3_permissions && cd ~/tp3_permissions
```

Objectif : comprendre et manipuler les permissions d'un fichier sous Linux.

1. Créez un fichier vide nommé `test.txt` puis affichez ses permissions par défaut.
2. Donnez **lecture seule** pour tous (`r--r--r--`) en mode symbolique.
3. Faites la même opération en mode octal.
4. Donnez **lecture+écriture** au propriétaire seulement (`rw-------`).
5. Ajoutez le **droit d'exécution** au groupe uniquement.
6. Vérifiez le résultat avec :

   ```
   stat test.txt
   ```

7. **Mise en situation** : créez un second fichier `secret.txt`, donnez-lui les droits suivants : `-rw-r----` (lecture/écriture pour le propriétaire, lecture seule pour le groupe, rien pour les autres).

   Pour tester avec un autre utilisateur, créez un nouveau compte utilisateur (à faire avec `sudo`) :

   ```
   sudo adduser testuser
   # définir un mot de passe quand demandé
   ```

   Ensuite, ouvrez un nouveau terminal et connectez-vous avec ce nouvel utilisateur :

   ```
   su - testuser
   # ou directement se connecter depuis l'écran de login
   ```

   Depuis ce nouvel utilisateur, essayez de lire le fichier `secret.txt` :

   ```
   cat /home/<votre_user>/secret.txt
   ```

   Observez que l'accès est refusé si vous n'appartenez pas au groupe autorisé.

Exercice 2 : Symbolique vs Octal

1. Reprenez le fichier `test.txt`. Modifiez ses permissions pour obtenir exactement : `-rw-r---`.
   — Faites-le une première fois en **mode symbolique**.
   — Puis refaites-le en **mode octal**.
   Vérifiez le résultat avec `ls -l`.
2. Quelles commandes permettent d'obtenir les permissions suivantes ? (écrire la version symbolique ET la version octale) :
   (a) `-rwxr-xr-x`
   (b) `-rw-rw-r--`
   (c) `-r--------`
3. Créez un script nommé `hello.sh` contenant une simple commande `echo "Hello"`.
   — Donnez-lui les permissions minimales pour que seul le propriétaire puisse l'exécuter.
   — Vérifiez que les autres utilisateurs ne peuvent pas l'exécuter.
4. Question de réflexion :
   — Quelle est la différence entre un fichier `-rw-r--r--` et un fichier `-rwxr--r--` ?
   — Pourquoi ne faut-il pas donner systématiquement le droit `x` à tous les fichiers ?

Exercice 3 : Comprendre `umask`

Rappel théorique : Sous Linux, la commande `umask` définit un *masque de permissions par défaut*. - Lorsqu'on crée un fichier, le système part de permissions maximales (666 = `rw-rw-rw-`) et retire ce qui est indiqué par la `umask`. - Lorsqu'on crée un répertoire, le système part de 777 = `rwxrwxrwx` et applique le même principe.

Exemple : - `umask 022` signifie que le groupe et les autres n'ont pas le droit `w`. - Ainsi, un fichier aura par défaut `644` (`rw-r-r-`) et un dossier `755` (`rwxr-xr-x`).

1. Affichez la `umask` actuelle et notez sa valeur :

```
umask
```

2. Créez un fichier `a.txt` et un dossier `dirA`. Comparez leurs permissions par défaut :

```
touch a.txt
mkdir dirA
ls -ld a.txt dirA
```

3. Changez temporairement `umask` à `002`, puis créez `b.txt` et `dirB`. Comparez :

```
umask 002
touch b.txt
mkdir dirB
ls -ld b.txt dirB
```

4. Revenez à votre `umask` initiale (*re-saisissez la valeur notée au début*).

Questions de réflexion :

- Pourquoi le système n'autorise-t-il pas le droit `x` par défaut sur les fichiers ordinaires, même si la `umask` ne l'interdit pas ?
- Quelle différence pratique entre `umask 022` et `umask 002` dans un projet de groupe où plusieurs utilisateurs partagent un répertoire ?
- Que se passerait-il si vous mettiez `umask 000` (aucune restriction) ? Quels problèmes de sécurité cela pourrait poser ?
- Peut-on définir une `umask` différente pour chaque utilisateur ? Où cela peut-il être configuré ?

Exercice 4 : Propriétaire & groupe (`chown`, `chgrp`)

1. Affichez votre UID/GID et vos groupes.
2. Créez `projet/` et un fichier `projet/notes.txt`. Vérifiez leur propriétaire et groupe.
3. Créez un nouveau groupe nommé `tpgroup`, puis ajoutez votre utilisateur à ce groupe (vous devrez peut-être vous reconnecter).
4. Changez le groupe du dossier `projet` vers `tpgroup`. Vérifiez.
5. Changez le propriétaire du fichier `notes.txt` vers un autre utilisateur (si possible, avec `sudo`). Vérifiez :

```
sudo chown autre_user projet/notes.txt
ls -l projet/notes.txt
```

Questions de réflexion :

- Quelle différence entre changer le **propriétaire** (`chown`) et changer le **groupe** (`chgrp`) ?
- Dans quel cas pratique un utilisateur changerait-il le groupe d'un fichier ?
- Pourquoi seul `root` (ou le propriétaire) peut modifier propriétaire/groupe d'un fichier ?
- Que se passerait-il si vous changiez par erreur le propriétaire de tout votre `$HOME` vers un autre utilisateur ?

Exercice avancé (facultatif) : Le bit SGID

1. Activez le **bit SGID** sur `projet/` et créez de nouveaux fichiers dedans. Observez le groupe attribué.

```
chmod g+s projet
mkdir projet/docs
touch projet/docs/todo.txt
ls -ld projet projet/docs
ls -l projet/docs/todo.txt
```

Explication : le bit SGID
- SGID = *Set Group ID*.
- Sur un **répertoire** : les nouveaux fichiers héritent du groupe du dossier.
- Très utile pour le travail collaboratif (projets partagés).

Questions de réflexion :
- Pourquoi est-il utile de mettre un répertoire en SGID dans un projet collaboratif ?
- Que se passerait-il si vous ne mettiez pas le SGID mais que plusieurs utilisateurs écrivaient dans le même dossier ?

Exercice avancé (facultatif) : Bits spéciaux (SUID, SGID, sticky)

Rappel théorique : En plus des permissions classiques (rwx), certains **bits spéciaux** existent :
- **SUID** (Set User ID) : un binaire s'exécute avec les privilèges de son propriétaire (souvent root).
- **SGID** (Set Group ID) : similaire, mais pour le groupe. Sur un répertoire, tous les fichiers créés héritent du groupe du répertoire.
- **Sticky bit** : sur un répertoire partagé, il empêche les utilisateurs de supprimer les fichiers des autres (ex. `/tmp`).

Attention : **Ne pas** essayer de mettre SUID sur vos propres scripts : c'est dangereux et souvent bloqué par le système. Nous allons uniquement observer et expérimenter dans des dossiers de test.

1. **Observer SUID/SGID sur des binaires système** : Listez quelques binaires connus et repérez le `s` dans les colonnes :

   ```
   ls -l /usr/bin/passwd
   ls -l /bin/su 2>/dev/null || ls -l /usr/bin/su 2>/dev/null
   ```

2. **Sticky bit** sur un répertoire partagé : Créez un répertoire comme `/tmp`, appliquez le sticky bit, et observez :

   ```
   mkdir -p shared_tmp
   chmod 1777 shared_tmp
   ls -ld shared_tmp
   ```

3. **Mettre/retirer SGID et sticky** sur un répertoire de test :

   ```
   mkdir lab && touch lab/a lab/b
   chmod g+s lab
   chmod +t lab
   ls -ld lab
   # Retirer
   chmod g-s lab
   chmod -t lab
   ls -ld lab
   ```

Questions de réflexion :
- Pourquoi la commande `passwd` a-t-elle le bit SUID activé ?
- Quelle différence entre SGID appliqué à un fichier exécutable et à un répertoire ?
- Pourquoi le sticky bit est-il indispensable dans `/tmp` ?
- Quels seraient les risques si un administrateur mettait SUID sur un programme non prévu pour cela ?

Exercice 6 : Rechercher par permissions (`find -perm`)

Rappel : L'option `-perm` de `find` permet de rechercher des fichiers selon leurs permissions classiques (`r`, `w`, `x`). Exemples : - `-perm -u=x` → fichiers exécutables par le propriétaire. - `-perm -g=w` → fichiers avec écriture pour le groupe. - `-perm 644` → fichiers dont les droits sont exactement `rw-r-r-`.

1. Recherchez dans `/tp3_permissions` tous les fichiers que le propriétaire peut exécuter.
2. Recherchez les fichiers où le groupe a la permission d'écriture.
3. Recherchez tous les fichiers qui ont exactement les permissions `rw-r-r-` (644).
4. Recherchez tous les fichiers où les `others` (autres) ont le droit d'exécution.

Questions de réflexion :
- Quelle différence entre `find -perm 644` et `find -perm -644` ?
- Pourquoi est-il important de vérifier les fichiers accessibles en écriture par « others » ?
- Donnez un exemple de situation où vous voudriez rechercher tous les fichiers exécutables dans un projet.

Exercice 7 : Redirections de base

Rappel : Sous Linux, un programme communique avec 3 flux standards :
- `stdin` (entrée standard, fd 0) – par défaut le clavier.
- `stdout` (sortie standard, fd 1) – par défaut l'écran.
- `stderr` (erreur standard, fd 2) – par défaut l'écran aussi.

Les redirections permettent de connecter ces flux à des fichiers ou à d'autres commandes.

1. Créez un fichier `liste.txt` contenant la sortie de la commande `ls`.
2. Ajoutez à la fin de `liste.txt` le résultat de la commande `date`.
3. Affichez uniquement le nombre de lignes de `liste.txt` en redirigeant son contenu vers `wc -l`.
4. Essayez d'afficher un fichier inexistant, puis redirigez le message d'erreur dans un fichier `erreurs.txt`.

Questions de réflexion :
- Quelle différence entre `>` et `>>` ?
- Pourquoi est-il utile de séparer la sortie standard et la sortie d'erreur ?
- Dans quel cas concret redirigeriez-vous les erreurs vers un fichier plutôt que de les laisser à l'écran ?

Exercice 8 : Redirections et pipes avec `dictionnaire.txt` et `livres.csv`

Préparation : Téléchargez les fichiers (voir TP2) de données suivants dans votre répertoire de TP :

```
wget https://perso.halim.info/iut_25_26/RT/R108_GnuLinux/dictionnaire.txt
wget https://perso.halim.info/iut_25_26/RT/R108_GnuLinux/livres.txt
```

Vérifiez leur présence avec `ls -l` puis familiarisez-vous avec leur contenu.

1. **Extraction et sauvegarde** : À partir de `dictionnaire.txt`, extrayez la colonne des catégories, triez-les et enregistrez le résultat dans un fichier `categories.txt`.
   - Quelle différence entre `>` et `>>` si vous relancez la commande deux fois ?
   - Vérifiez combien de catégories différentes existent dans ce fichier.
2. **Chaînage de commandes** :
   - Comptez le nombre de mots français du dictionnaire qui commencent par la lettre `M`.
   - Comptez le nombre de mots anglais qui contiennent la lettre `o`.
   - Expliquez pourquoi l'usage du pipe est plus efficace que de créer plusieurs fichiers temporaires.
3. **Analyse sur les livres** :
   - Dans `livres.csv`, extrayez uniquement les titres des livres de type `Science-Fiction` et redirigez-les dans `sf.txt`.
   - Affichez ensuite le contenu de `sf.txt` tout en l'enregistrant dans un fichier `sf.log` grâce à `tee`.
   - Quelle est la différence entre `tee sf.log` et `tee -a sf.log` ?
4. **Comptage avancé** :

— Combien de livres ont été publiés avant 1950 ?
— Parmi les livres publiés entre 1950 et 1999, combien appartiennent au genre `Roman` ? (Utilisez une combinaison de `grep`, `cut`, `wc` et éventuellement des regex).

5. **Réflexion** :
   — Donnez un exemple de situation concrète (hors TP) où vous auriez besoin de rediriger les erreurs dans un fichier.
   — Expliquez pourquoi on préfère parfois utiliser `tee` plutôt qu'une simple redirection avec `>`.

Exercice 9 : Here-doc et Here-string

Rappel théorique :
— Un **Here-document** (`<<`) permet de fournir un bloc de texte multi-lignes comme entrée standard (`stdin`) d'une commande. Le texte est saisi entre deux délimiteurs choisis (souvent `EOF`).
— Un **Here-string** (`<<<`) est similaire mais pour une seule ligne de texte.
— Le mot-clé `EOF` n'a rien de spécial en lui-même : il pourrait être remplacé par `FIN`, `END`, ou tout autre identifiant. Il rappelle simplement la notion de `End Of File`, que l'on obtient aussi en saisie manuelle avec le raccourci `CTRL+D`.

1. Créez un fichier `poeme.txt` en utilisant un here-doc :

```
cat <<EOF > poeme.txt
Roses are red,
Violets are blue,
Linux is powerful,
And so are you.
EOF
Ctrl + D
```

2. Affichez la sortie d'une commande en lui donnant directement une chaîne avec un here-string :

```
wc -w <<< "Linux est génial"
```

Questions de réflexion :
— Que se passe-t-il si vous changez `EOF` par `FIN` dans l'exemple ?
— Quelle différence entre utiliser un here-doc et rediriger avec `>` puis éditer manuellement un fichier ?
— Donnez un cas concret où un here-doc est plus pratique qu'un fichier temporaire.

Exercice 10 : Lien physique (hard link)

Rappel théorique : Un **lien dur** (hard link) est une autre entrée dans le système de fichiers qui pointe vers le même **inode** qu'un fichier existant.
— Les deux fichiers sont en réalité le même objet : ils partagent le même contenu et les mêmes droits.
— Supprimer un des deux noms ne supprime pas le contenu tant qu'il reste au moins un lien actif.
— Un lien dur ne peut pas pointer vers un répertoire (sauf cas particuliers réservés au système).
— Un lien dur ne peut pas traverser deux systèmes de fichiers différents.

1. Créez un fichier `original.txt` avec du texte.
2. Créez un **lien dur** nommé `copie_hard.txt`.
3. Vérifiez avec `ls -li` que les deux fichiers pointent vers le même inode. Notez aussi la colonne indiquant le **nombre de liens**.
4. Modifiez `copie_hard.txt` (par exemple ajoutez une ligne) puis affichez le contenu de `original.txt`. Que remarquez-vous ?
5. Supprimez `original.txt` et affichez le contenu de `copie_hard.txt`. Que se passe-t-il ?

Questions de réflexion :
— Quelle est la différence entre un **lien dur** et une **copie classique** d'un fichier ?
— Quelle est la différence entre un **lien dur** et un **lien symbolique** ?
— Pourquoi les liens durs sont-ils interdits sur les répertoires pour les utilisateurs normaux ?
— Dans quel cas pratique un administrateur pourrait-il utiliser un lien dur ?

Exercice 11 : Liens symboliques (symlinks)

1. Créez un fichier `notes.txt` avec du contenu.
2. Créez un lien symbolique nommé `lien_sym.txt` pointant vers `notes.txt`.
3. Comparez avec `ls -l` et `ls -li` les différences entre l'original et le lien symbolique.
4. Supprimez `notes.txt`, puis essayez d'afficher le contenu de `lien_sym.txt`. Que remarquez-vous ?

Questions de réflexion :
— Quelle différence fondamentale avec un lien dur (Exercice précédent) ?
— Dans quel cas pratique préférer un lien symbolique à un lien dur ?

Exercice 12 : Types de fichiers et commandes `file`/`stat`

Rappel : Sous Linux, tout est fichier. Les principaux types sont :
— `-` fichier ordinaire
— `d` répertoire
— `l` lien symbolique
— `b` périphérique bloc
— `c` périphérique caractère
— `p` FIFO (tube nommé)
— `s` socket

1. Identifiez le type des fichiers suivants :

```
file /etc/passwd
file /bin/ls
file /dev/null
```

2. Créez un fichier vide `exemple.txt`, puis observez ses métadonnées :

```
touch exemple.txt
stat exemple.txt
```

3. Que représentent les informations suivantes dans `stat` ?
   — inode
   — UID / GID
   — liens

Mini-projet 1 : Gestion d'un espace collaboratif sécurisé

Contexte : Vous êtes administrateur d'un petit laboratoire universitaire. Vous devez créer un espace collaboratif où plusieurs utilisateurs peuvent travailler sur des fichiers communs tout en respectant les règles suivantes : - Chaque projet a son propre répertoire. - Les fichiers doivent être accessibles uniquement aux membres du projet. - Les droits doivent être configurés de manière à éviter les erreurs (ni trop restrictifs, ni trop permissifs). - Les utilisateurs doivent pouvoir ajouter des fichiers sans casser les droits des autres. - Certains fichiers sont confidentiels et doivent être protégés par des permissions spécifiques.

Travail demandé : Réalisez les étapes suivantes, documentez vos commandes et expliquez vos choix.

1. **Mise en place des utilisateurs et groupes**
   — Créez deux utilisateurs : `alice` et `bob`.
   — Créez un groupe nommé `projetgrp`.
   — Ajoutez `alice` et `bob` au groupe.
   — Vérifiez leur appartenance avec `id` et `groups`.
2. **Création de l'arborescence du projet**

```
projet_securise/
|-- docs/
|-- src/
|-- secrets/
```

- Le dossier `docs` servira aux documents partagés. - Le dossier `src` servira aux scripts et programmes. - Le dossier `secrets` contiendra des fichiers accessibles uniquement à l'administrateur.

3. **Gestion des permissions**
   — Donnez au groupe `projetgrp` l'accès complet à `docs` et `src`.
   — Activez le bit SGID sur ces dossiers pour que tous les nouveaux fichiers héritent du groupe.
   — Vérifiez que les droits sont corrects avec `ls -ld`.
   — Assurez-vous que seul le propriétaire (root/admin) peut accéder à `secrets`.
4. **Création et manipulation de fichiers**
   — Dans `docs`, créez un fichier `rapport.txt` et configurez ses droits pour que tout le groupe puisse le lire et l'écrire.
   — Dans `src`, créez un script `hello.sh` qui affiche "Hello Projet".
   — Configurez ses droits pour qu'il soit exécutable uniquement par les membres du groupe.
   — Dans `secrets`, créez un fichier `motdepasse.txt` lisible uniquement par root.
5. **Utilisation des liens**
   — Créez un lien dur vers `rapport.txt` nommé `copie_dur.txt` (dans `docs`).
   — Créez un lien symbolique dans le répertoire personnel de `alice` pointant vers `rapport.txt`.
   — Supprimez l'original et observez la différence entre le lien dur et le lien symbolique.
6. **Exploration et recherche avec `find`**
   — Recherchez tous les fichiers du projet qui sont exécutables.
   — Recherchez tous les fichiers du projet qui sont accessibles en écriture par "others".
   — Supprimez toute permission d'écriture aux "others" si vous en trouvez.
7. **Redirections et logs**
   — Créez un script `analyse.sh` qui liste tous les fichiers du projet, compte leur nombre et enregistre le résultat dans `resultats.log`.
   — Configurez le script pour que les erreurs soient redirigées dans un fichier séparé `erreurs.log`.
   — Ajoutez une commande dans le script qui affiche les résultats à l'écran tout en les enregistrant grâce à `tee`.
8. **Question finale de réflexion** :
   — Quelles bonnes pratiques de sécurité avez-vous appliquées dans ce projet ?
   — Que se passerait-il si vous donniez systématiquement les droits 777 à tous les fichiers et dossiers ?

Mini-projet 2 : Analyse de journaux système avec redirections et pipes

Contexte : Vous êtes chargé d'analyser des fichiers journaux (`logs`) produits par un serveur Linux. Vous devez extraire des informations utiles, compter des occurrences, et générer un rapport clair en utilisant uniquement les commandes vues : redirections (`>`, `>>`, `<`, `2>`), pipes (`|`), et la commande `tee`.

Préparation : Créez un dossier de travail et téléchargez un fichier de log simulé :

```
mkdir -p ~/tp3_logs && cd ~/tp3_logs
wget https://perso.halim.info/iut_25_26/RT/R108_GnuLinux/logs.txt
```

Le fichier `logs.txt` contient des entrées comme :

```
2024-11-01 12:01:05 INFO User alice logged in
2024-11-01 12:03:15 ERROR Disk quota exceeded for bob
2024-11-01 12:04:55 WARNING Low memory
2024-11-01 12:06:10 INFO User bob uploaded file report.pdf
...
```

Travail demandé :

1. **Première exploration**
   — Comptez le nombre total de lignes dans le fichier.
   — Redirigez le résultat dans un fichier `stats.log`.
   — Ajoutez à la fin du même fichier la date d'exécution (`date`).
2. **Filtrage avec `grep`**
   — Comptez le nombre d'erreurs (`ERROR`) dans le fichier.

— Comptez le nombre de warnings (`WARNING`).
— Stockez ces résultats dans `stats.log` tout en les affichant à l'écran grâce à `tee`.

3. **Analyse par utilisateur**
   — À l'aide de pipes, extrayez uniquement les lignes contenant le mot `User`.
   — Coupez la 4e colonne (le nom d'utilisateur) et comptez le nombre d'occurrences de chaque utilisateur.
   — Enregistrez ce tableau dans `users.log`.
4. **Gestion des erreurs et redirections avancées**
   — Lancez une commande volontairement incorrecte (par ex. `cat fichier_inexistant`).
   — Redirigez uniquement les erreurs dans `erreurs.log`.
   — Vérifiez que le fichier `erreurs.log` contient bien le message d'erreur.
5. **Script d'automatisation** Créez un script `analyse_logs.sh` qui :
   — Compte les lignes totales.
   — Compte les erreurs et warnings.
   — Liste les utilisateurs distincts et leur nombre d'actions.
   — Génère un rapport complet dans `rapport.log`, en l'affichant aussi à l'écran avec `tee`.