Note de cours : les droits de Fichier sous ubuntu [COURS-2026-T5JJ]

📚 Type de document : Cours

Les droits de Fichier sous Ubuntu

Introduction et contexte

📌 Contexte historique ou pratique

📖 **Définition :** : Unix = système d'exploitation inventé dans les années 1970, à l'origine du modèle de permissions utilisateur/groupe/autres utilisé aujourd'hui.

🎯 **Application :** : Ubuntu est une distribution Linux utilisant ce modèle POSIX. Les droits de fichiers sont fondamentaux pour la sécurité, le partage et le fonctionnement correct des services (serveurs, scripts, accès multi-utilisateurs).

• Historique : le modèle rwx (read, write, execute) remonte aux premiers Unix. Il était léger et efficace pour gérer l'accès aux fichiers sur des systèmes multi-utilisateurs.

• Pratique : sur un serveur Ubuntu, une mauvaise permission peut provoquer des failles de sécurité (chmod 777), empêcher l'exécution d'un service (absence du bit d'exécution), ou gêner la collaboration entre utilisateurs (mauvaise appartenance au groupe).

Comprendre le modèle de permissions POSIX (compétence 1)

📌 Concepts fondamentaux

📖 **Définition :** : permission = règle qui détermine quelles opérations (lecture, écriture, exécution) un utilisateur peut effectuer sur un fichier ou répertoire.

📖 **Définition :** : utilisateur/propriétaire = compte système auquel appartient le fichier (owner).

📖 **Définition :** : groupe = ensemble d'utilisateurs; chaque fichier appartient à un groupe.

📖 **Définition :** : autres = tous les autres utilisateurs n'appartenant pas au propriétaire ni au groupe.

📖 **Définition :** : rwx = read (r), write (w), execute (x).

📖 **Définition :** : mode = représentation des permissions d'un fichier (symbolique ou octale).

📖 **Définition :** : inode = structure de données qui contient les métadonnées d'un fichier (permissions, propriétaire, timestamps, pointeurs vers les blocs).

📖 **Définition :** : type de fichier = '-' fichier ordinaire, 'd' répertoire, 'l' lien symbolique, 'c' caractère, 'b' bloc, 's' socket, 'p' FIFO.

- ls -l : affiche le mode et les métadonnées.

> 💡 **Exemple :** : sortie "drwxr-xr-x 2 alice dev 4096 2026-01-01 dir" signifie :

- 'd' = répertoire
- 'rwx' = propriétaire (alice) a lecture, écriture, exécution
- 'r-x' = groupe (dev) a lecture et exécution
- 'r-x' = autres ont lecture et exécution

📌 **Exemple** : interprétation complète

> 💡 **Exemple :** : "rw-r--r--" (fichier) :

- propriétaire : read+write (rw-)
- groupe : read (r--)
- autres : read (r--)
- En octal : 644 (4+2 = 6 pour propriétaire, 4 pour groupe, 4 pour autres)

> 💡 **Exemple :** : "rwxr-x---" → octal 750.

📌 **Méthode** : visualiser les permissions

1. Utiliser ls -l /chemin/fichier
2. Utiliser stat -c "%A %a %U %G %n" fichier pour obtenir représentation symbolique, octale, propriétaire, groupe, nom.
3. Utiliser namei -lx /chemin/vers/fichier pour inspecter chaque composant de chemin (utile pour droits sur répertoires parents).

📌 **Application** : pourquoi ces concepts sont importants

- Gestion de serveurs multi-utilisateurs.
- Protection des scripts sensibles.
- Sécurisation des répertoires partagés.
- Dépannage d'un service qui ne démarre pas à cause d'un fichier non lisible.

📌 **Attention** : pièges de base

- Les permissions des fichiers sont indépendantes du propriétaire : même si un fichier appartient à root, d'autres peuvent avoir des droits.
- L'absence du bit 'x' sur un répertoire empêche l'accès (pas seulement l'exécution).
- Les permissions listées par ls -l pour les liens symboliques sont souvent celles du lien lui-même ; l'accès dépend du fichier cible.

Commandes de base pour gérer permissions et propriété (compétence 2)

📌 Concepts fondamentaux

📖 **Définition :** : chmod = commande pour modifier les permissions (symbolique ou octale).

📖 **Définition :** : chown = changer le propriétaire et/ou le groupe d'un fichier.

📖 **Définition :** : chgrp = changer le groupe d'un fichier.

📖 **Définition :** : sudo = exécuter avec élévation de privilèges (nécessaire pour chown et certains chmods).

📖 **Définition :** : recursive (-R) = option pour appliquer une commande à tous les fichiers et sous-répertoires.

📌 **Exemple** : chmod en octal et symbolique

- Octal :

> 💡 **Exemple :** : chmod 644 fichier → propriétaire rw-, groupe r--, autres r--.

> 💡 **Exemple :** : chmod 755 script.sh → propriétaire rwx, groupe rx, autres rx (typique pour scripts exécutables).

- Symbolique :

> 💡 **Exemple :** : chmod u+rwx,g+rx,o+rx fichier équivaut à chmod 755 fichier.
>
> 💡 **Exemple :** : chmod g-w fichier retire l'écriture pour le groupe.

📌 **Méthode** : comment utiliser chmod/chown/chgrp étape par étape

4. Inspecter : ls -l fichier ; stat fichier.
5. Déterminer la configuration désirée (octal ou symbolique).
6. Exécuter :

- chmod 750 fichier
- sudo chown alice:dev fichier
- chgrp dev fichier (si vous êtes membre du groupe ayant permission de changer le groupe)

7. Vérifier : ls -l fichier ; stat fichier.

📌 **Application** : opérations courantes

• Rendre un script exécutable : chmod +x script.sh ou chmod 755 script.sh.

• Mettre à jour le propriétaire après extraction d'archive : sudo chown -R deploy:deploy /var/www/app.

• Partager un répertoire entre membres d'un groupe : chown :sharedgroup /srv/shared ; chmod 2770 /srv/shared (voir setgid plus bas).

📌 **Attention** : erreurs fréquentes avec ces commandes

• Utiliser chmod -R 777 / (catastrophique) ; n'utiliser jamais -R 777 sans comprendre.

• chown sans sudo échouera généralement si vous n'êtes pas propriétaire.

• chgrp échoue si vous n'êtes pas membre du groupe cible (selon configuration).

Bits spéciaux et comportements avancés (compétence 3)

📌 Concepts fondamentaux

📖 **Définition :** : setuid (SUID) = bit spécial (octal 4000) qui fait exécuter un binaire avec l'UID du propriétaire du fichier.

📖 **Définition :** : setgid (SGID) = bit spécial (octal 2000) qui fait exécuter un binaire avec le GID du fichier; pour les répertoires, force l'héritage du groupe.

📖 **Définition :** : sticky bit = bit spécial (octal 1000) qui pour les répertoires empêche la suppression d'un fichier par un utilisateur autre que le propriétaire du fichier, le propriétaire du répertoire ou root.

📖 **Définition :** : permission effective = ensemble réel des droits appliqués au moment de l'accès (peut être influencé par SUID/SGID/ACLs/AppArmor).

📖 **Définition :** : mask (pour ACL) = limite maximale des droits qui peuvent être accordés aux groupes/ACL.

📌 **Exemple** : valeurs numériques spéciales

- chmod 4755 /usr/bin/passwd → setuid + rwxr-xr-x (4000 + 0755 = 4755).

• chmod 2750 /srv/shared → setgid + rwxr-x--- (2000 + 0750 = 2750).

• chmod 1777 /tmp → sticky + rwxrwxrwx (1000 + 0777 = 1777).

📌 **Méthode** : définir et retirer bits spéciaux

8. Pour ajouter setuid : chmod u+s binaire (ou chmod 4755 binaire).

9. Pour retirer setuid : chmod u-s binaire.

10. Pour ajouter setgid : chmod g+s répertoire (ou chmod 2755).

11. Pour ajouter sticky : chmod o+t répertoire (ou chmod 1777 /tmp).

12. Vérifier avec ls -l : les positions S/s/T/t indiquent les bits spéciaux (S majuscule = setuid/setgid présent mais bit d'exécution absent; s minuscule = setuid/setgid et exécution présents; T/T pour sticky).

📌 **Application** : cas d'utilisation des bits spéciaux

• SUID sur /usr/bin/passwd pour permettre à un utilisateur non-root de changer son mot de passe (programme s'exécute sous uid root).

• SGID sur un répertoire partagé pour que tous les fichiers créés héritent du groupe du répertoire, simplifiant la collaboration.

• Sticky bit sur /tmp pour empêcher la suppression de fichiers d'autres utilisateurs.

📌 **Attention** : risques et précautions

• SUID sur des binaires non sûrs peut créer des failles d'élévation de privilèges. Eviter SUID sur des scripts interprétés (souvent ignoré par le kernel ou dangereux).

• SGID mal configuré cause des fichiers appartenant à un groupe non souhaité.

• Sticky bit seul ne protège pas contre la lecture/écriture — il protège la suppression dans le répertoire.

• Toujours auditer les binaires SUID sur le système : find / -perm /4000 -type f -exec ls -ld {} \;

ACL (Access Control Lists) et attributs étendus

📌 Concepts fondamentaux

📖 **Définition :** : ACL = mécanisme permettant d'accorder des permissions finement définies par utilisateur ou par groupe au-delà du modèle classique owner/group/others.

📖 **Définition :** : default ACL = ACL appliquée par défaut aux nouveaux fichiers créés dans un répertoire.

📖 **Définition :** : xattr (attributs étendus) = métadonnées supplémentaires stockées sur les fichiers (utilisées par SELinux/AppArmor, etc).

📖 **Définition :** : POSIX ACL = implémentation courante sur Linux (supportée par ext4, xfs, btrfs, etc).

📌 **Exemple** : commandes getfacl/setfacl

- getfacl fichier → montre ACL d'un fichier.
- setfacl -m u:alice:rw fichier → ajoute à alice les droits rw.
- setfacl -m g:dev:rx fichier → ajoute au groupe dev les droits rx.

- setfacl -m d:g:dev:rw /srv/shared → définit une default ACL pour le répertoire.
- setfacl -b fichier → supprime toutes les ACL.

> 💡 **Exemple :** : sortie getfacl

file: rapport.txt

owner: bob

group: dev

user::rw-

user:alice:r--

group::r--

mask::r--

other::r--

📌 **Méthode** : définir des ACL pratiques

13. Vérifier que le système de fichiers supporte les ACLs : mount | grep acl ; tune2fs -l /dev/sdXN | grep "Default mount options" ; ou essayer getfacl.

14. Définir une ACL utilisateur : sudo setfacl -m u:alice:rw fichier

15. Définir une ACL groupe : sudo setfacl -m g:qa:rwx répertoire

16. Définir ACL par défaut pour un répertoire : sudo setfacl -m d:u:carole:rwx /srv/project

17. Vérifier : getfacl /srv/project ; ls -l (un + apparait après la permission si ACL présente: rwxrwxr-x+).

📌 **Application** : exemples d'usage

• Collaboration multi-équipe : permettre à Alice d'éditer certains fichiers sans la rendre propriétaire.

• Héritage de permissions sur un répertoire partagé via default ACL.

• Limitation fine d'accès sur des répertoires contenant des données sensibles.

📌 **Attention** : pièges ACL

• Les ACL interagissent avec le mask : si le mask est plus restrictif, il limitera les droits effectifs des entrées de groupe et des utilisateurs nommés.

• Restauration depuis archive tar peut ne pas préserver les ACLs par défaut (utiliser tar --acls).

• Complexity creep : trop d'ACLs nuit à la maintenance ; documenter les ACLs et garder une politique simple.

• Certains outils graphiques ou scripts s'attendent au modèle traditionnel et ignorent les ACLs.

umask, création de fichiers et comportements par défaut

📌 Concepts fondamentaux

📖 **Définition :** : umask = valeur masquée soustraite des permissions par défaut lors de la création d'un fichier ou répertoire.

📖 **Définition :** : valeur par défaut typique = 022 (fichiers 644, répertoires 755), ou 002 pour environnements de groupe collaboratif (fichiers 664, dirs 775).

📖 **Définition :** : ingénierie inversée du mask = permissions_par_defaut = 666 pour fichiers / 777 pour répertoires moins umask.

📌 **Exemple** : calculs umask

- umask 022 :
- fichiers : 666 - 022 = 644
- répertoires : 777 - 022 = 755
- umask 002 :
- fichiers : 666 - 002 = 664
- répertoires : 777 - 002 = 775

📌 **Méthode** : modifier et persister umask

18. Voir umask courant : umask (dans shell).
19. Définir pour session : umask 002

20. Persister pour utilisateur : ajouter umask 002 dans ~/.profile ou ~/.bashrc (selon le shell).

21. Persister globalement : modifier /etc/profile, /etc/login.defs (nota : /etc/login.defs contient souvent UMASK pour shells de login).

22. Pour services systemd, utiliser /etc/systemd/system/service.service.d/override.conf ou Environment=UMASK=002 dans le fichier d'unité.

📌 **Application** : scenarios pratiques

• Environnement de développement collaboratif → umask 002 pour que les fichiers créés appartiennent au groupe et soient modifiables par les coéquipiers.

• Serveur web → umask 022 pour empêcher écriture par groupe/autres.

📌 **Attention** : erreurs liées à umask

• umask ne modifie pas les permissions des fichiers existants.

• Certains programmes (par ex. scp/rsync) peuvent créer fichiers avec permissions spécifiques indépendamment du umask.

• Umask dans systemd peut être ignoré si le service définit explicitement les permissions.

Permissions et types de fichiers spécifiques

📌 Concepts fondamentaux

📖 **Définition :** : permission sur répertoire = read (listage), write (création/suppression), execute (traversal/accès).

📖 **Définition :** : permission sur lien symbolique = généralement ignorée ; le kernel suit le lien vers la cible et applique les permissions de la cible.

📖 **Définition :** : hard link = entrée supplémentaire pointant vers le même inode ; les permissions sont liées à l'inode, pas à l'entrée.

📖 **Définition :** : systèmes de fichiers non POSIX = FAT, NTFS (via driver), exFAT ; gèrent différemment les permissions (mount options influencent uid/gid/umask/fmask/dmask).

📌 **Exemple** : comportement des répertoires

• Un répertoire sans 'x' (par exemple rw-) : même si vous avez 'r' (listage), vous ne pouvez pas entrer dans le répertoire ni ouvrir des fichiers par chemin.

• Pour accéder à /home/alice/docs/fichier, il faut avoir le droit d'exécution (x) sur /home et /home/alice et /home/alice/docs selon l'arborescence.

📌 **Méthode** : monter un système de fichiers non-POSIX avec permissions

• Pour vfat : mount -t vfat -o uid=1000,gid=1000,umask=0022 /dev/sdb1 /mnt/usb

• Pour ntfs-3g : mount -t ntfs-3g -o permissions /dev/sdb1 /mnt/ntfs (permet de simuler POSIX permissions mais attention aux limitations).

📌 **Application** : implications pratiques

• Brancher une clé USB formatée FAT : permissions seront gérées au montage ; tous les fichiers peuvent apparaître appartenir à l'utilisateur qui monte.

• Copier des fichiers depuis Windows vers Linux : vérifier et ajuster les permissions après copie.

📌 **Attention** : pièges spécifiques

• Croire que chmod fonctionne sur vfat/ntfs de la même manière ; ce n'est pas le cas.

• Ne pas oublier les droits sur les répertoires parents lors du dépannage d'accès.

• Les liens symboliques conservent leurs propres métadonnées mais les permissions de l'accès sont celles de la cible.

Applications pratiques

📌 Cas d'usage concrets

• Partage de projet entre développeurs :

• Créer groupe "devteam", ajouter utilisateurs : sudo groupadd devteam ; sudo usermod -aG devteam alice

• Créer dossier partagé : sudo mkdir /srv/devproject ; sudo chown :devteam /srv/devproject ; sudo chmod 2770 /srv/devproject ; expliquer résultat (SGID + rwx pour owner/groupe).

> 💡 **Exemple :** : un employé crée un fichier, il hérite du groupe devteam grâce au SGID, et les autres membres du groupe peuvent modifier le fichier.

• Serveur web sécurisé :

• Fichiers web en propriétaire root:root mais groupe www-data, permissions 640, scripts exécutables selon besoin.

• S'assurer que le processus web (www-data) a exactement les droits nécessaires.

• Répertoire /tmp et sticky bit :

• chmod 1777 /tmp assure que seul le propriétaire d'un fichier peut le supprimer, même si d'autres ont write dans /tmp.

• Gestion de mots de passe :

• /usr/bin/passwd possède setuid root pour permettre changement de mot de passe.

📌 Exercices pratiques

• Exercice 1 : Inspecter et décrire le mode de /etc/passwd, /tmp, /var/www.

23. ls -l /etc/passwd ; stat /etc/passwd

24. Interpréter les bits.

• Exercice 2 : Créer un dossier partagé pour le groupe "projectA" et configurer SGID et umask 002 pour les nouveaux fichiers.

25. sudo groupadd projectA

26. sudo mkdir /srv/projectA ; sudo chown :projectA /srv/projectA ; sudo chmod 2775 /srv/projectA

27. Configurer les utilisateurs (usermod -aG projectA user)

28. Tester création de fichiers et vérifier le groupe.

• Exercice 3 : Définir une ACL permettant à l'utilisateur "bob" de lire/écrire un fichier appartenant à alice sans changer la propriété.

29. setfacl -m u:bob:rw alicefile

30. getfacl alicefile pour vérifier.

• Exercice 4 : Rechercher tous les fichiers setuid du système et les documenter.

31. sudo find / -perm /4000 -type f -exec ls -ld {} \; > setuid_list.txt

32. Analyser chaque binaire pour comprendre si setuid est légitime.

Points d'attention et erreurs courantes

⚠️ **Attention :** : chmod -R 777 / donne accès complet à tous, compromettant la sécurité.

⚠️ **Attention :** : setuid sur script shell est généralement ignoré ou dangereux; privilégier des binaires C audités.

⚠️ **Attention :** : oublier l'existence des répertoires parents lors d'un problème d'accès.

⚠️ **Attention :** : ACLs complexes non documentées rendent la maintenance difficile.

⚠️ **Attention :** : confondre exécution pour fichier (run) et exécution pour répertoire (traversal).

⚠️ **Attention :** : les permissions sur un fichier ne protègent pas contre la lecture si l'utilisateur est root.

⚠️ **Attention :** : montage de systèmes de fichiers Windows (vfat/ntfs) peut masquer le véritable modèle de permissions POSIX.

⚠️ **Attention :** : backups et restaurations peuvent perdre UID/GID et ACLs — vérifier options d'archivage (tar --numeric-owner --acls --xattrs).

Résumé et points clés à retenir

⚠️ : Le modèle de base est owner/group/others avec rwx ; comprendre cette matrice est primordial.

- Utilisez ls -l, stat, getfacl pour diagnostiquer.
- chmod (symbolique ou octal), chown, chgrp sont vos outils de base.
- umask influence les permissions par défaut lors de la création de fichiers.
- Bits spéciaux (SUID, SGID, sticky) ont des usages puissants mais des risques de sécurité.

• ACLs offrent une granularité supplémentaire mais complexifient l'administration.

• Toujours appliquer le principe du moindre privilège : donner uniquement les droits nécessaires.

• Documentez et auditez régulièrement les permissions sensibles (setuid, répertoires partagés).

Lexique complet des termes techniques

📖 **Définition :** : owner/propriétaire = compte auquel appartient un fichier.

📖 **Définition :** : group/groupe = identifiant partagé entre utilisateurs.

📖 **Définition :** : others/autres = utilisateurs non permis par owner/group.

📖 **Définition :** : read (r) = permission de lire le contenu d'un fichier ou de lister un répertoire (pour dossier, read = lister les entrées).

📖 **Définition :** : write (w) = permission d'écrire/éditer un fichier ou de créer/supprimer des fichiers dans un répertoire.

📖 **Définition :** : execute (x) = permission d'exécuter un fichier comme un programme ou de traverser un répertoire (cd).

📖 **Définition :** : chmod = change mode, modifie permissions.

📖 **Définition :** : chown = change owner, modifie propriétaire/groupe.

📖 **Définition :** : chgrp = change group, modifie groupe.

📖 **Définition :** : umask = valeur masquée soustraite des permissions par défaut.

📖 **Définition :** : ACL = Access Control List, permissions fines par utilisateur/groupe.

📖 **Définition :** : setuid = bit qui exécute un binaire sous l'UID du propriétaire.

📖 **Définition :** : setgid = bit qui exécute un binaire sous le GID du fichier ; pour répertoires, hérite du groupe.

📖 **Définition :** : sticky bit = empêche suppression par autres utilisateurs (utile sur /tmp).

📖 **Définition :** : inode = structure contenant métadonnées du fichier.

📖 **Définition :** : hard link = référence supplémentaire au même inode.

📖 **Définition :** : symbolic link (symlink) = fichier spécial pointant vers un autre chemin.

📖 **Définition :** : xattr = attributs étendus stockant métadonnées supplémentaires (utilisés par SELinux/AppArmor).

📖 **Définition :** : mask (ACL) = limite maximale des droits accordables via ACL pour les entrées de groupe/utilisateur.

📖 **Définition :** : mount options = paramètres utilisés lors du montage d'un système de fichiers qui influencent le comportement des permissions (ex: uid, gid, fmask, dmask, permissions pour ntfs-3g).

📖 **Définition :** : POSIX = Portable Operating System Interface, norme qui définit le modèle de permissions.

📖 **Définition :** : sudo = permet d'exécuter des commandes en tant qu'autre utilisateur (généralement root), nécessaire pour changer certains propriétaires/permissions.

Annexes : commandes de référence rapide

- Voir permissions : ls -l fichier ; stat fichier ; getfacl fichier
- Changer permissions :
- chmod 644 fichier
- chmod u+rwx,g+rx,o-rw fichier
- chmod -R 750 dossier
- Changer propriétaire/groupe :
- sudo chown alice fichier
- sudo chown alice:dev fichier
- chgrp dev fichier
- Bits spéciaux :
- chmod u+s /usr/bin/quelquechose (setuid)
- chmod g+s /srv/shared (setgid)
- chmod o+t /tmp (sticky)
- ACL :

- getfacl fichier
- setfacl -m u:alice:rw fichier
- setfacl -m d:g:dev:rwx /srv/shared
- setfacl -b fichier (supprimer ACL)
- Rechercher fichiers avec bits :
- find / -perm /4000 -type f -ls (setuid)
- find / -perm /2000 -type d -ls (setgid)
- find / -perm /1000 -type d -ls (sticky)
- Diagnostiquer montage :
- mount | grep /dev/sd
- cat /proc/mounts

Conclusion

👨‍🏫 Professeur : Julien Mievis

📅 Date de création : 13/01/2026