

COMMANDES GNU / LINUX
GESTION DES PERMISSIONS ET DROITS
D'ACCES

Changement des permissions : chmod, chown et chgrp

Les permissions et les bits de permissions : l'inode de chaque fichier contient un champ dont les bits définissent qui peut faire quoi avec ce fichier :

- le bit R, *Read* : s'il est armé, la lecture est autorisée, désarmé, il interdit de lire le fichier.
- le bit W, *Write* : il contrôle de la même manière l'écriture sur le fichier et la possibilité de le supprimer.
- le bit X, *execute* : pour un fichier, il contrôle la possibilité de l'exécuter (si c'est un exécutable ou non). Pour un répertoire, il permet ou il interdit l'accès à ce répertoire.

Certaines combinaisons n'ont pas de sens : un fichier avec w armé et r, et x désarmés ne peut être que modifié ou effacé, mais pas lu ni exécuté, ce qui est utile ! De même x armé et r désarmé vous donnent un fichier que vous pouvez certes exécuter après l'avoir chargé, mais vous ne pouvez pas le charger en lecture, car il n'est pas lisible...

En fait chaque fichier possède trois bits r, trois w et trois x, ce qui fait neuf en tout. Le premier trio rwx définit les permissions pour le propriétaire du fichier (c'est à dire pour toute tâche dont l'UID correspond à l'UID du fichier), le second pour tous les utilisateurs membres du groupe auquel appartient ce fichier, et enfin le troisième pour tous les autres :

Ex : 1s – 1 permet de voir ces bits

Vous savez déclarer les droits des utilisateurs par défaut en utilisant la commande interne `umask` (voir le chapitre 2 : commandes usuelles), mais il faut encore pouvoir modifier les bits de permissions à votre convenance :

`chmod`, *CHange MODe*, cette commande relativement complexe à maîtriser modifie les permissions d'accès de chacun des fichiers indiqués, en suivant l'indication de mode, qui peut être une représentation symbolique du changement à effectuer, ou un nombre octal représentant le motif binaire des nouvelles autorisations :

Ex : `chmod permissions fichiers`

`chmod u+rw,g+rw,orw titi.txt` change les permissions du fichier en mode symbolique

`chmod 650titi.txt` change les permissions du fichier `titi.txt` en mode octal

La manière la plus courante de représenter des permissions est le mode chiffré en octal :

4, 2, 1 la valeur octale est la représentation numérique ou chiffrée des permissions de la forme symbolique (r, w, x).

Plutôt que d'utiliser le mode des permissions sous la représentation de nombres octaux (421), nous allons plutôt nous intéresser à la représentation symbolique. La représentation symbolique a la forme suivante `[ugo]+/[rwx]:`

u, *User*, il s'agit des permissions du propriétaire (400, 200, 100)

g, *Group*, il s'agit des permissions des utilisateurs du groupe auquel appartient le fichier (40, 20, 10) o, *Other*, celles des autres utilisateurs (4, 2, 1).

+, *Plus*, le signe positif ajoute la permission

-, *Moins*, le signe négatif enlève le permission

=, *Egal*, le signe égal fixe une permission

r, w, x, *Read, Write, exécute* la combinaison de permissions concernées :

ex : `chmod u+rwx` donne le droit de lire, écrire et exécuter le fichier `titi.txt` à son propriétaire

`chmod ug+w` autorise le propriétaire ainsi que les utilisateurs du groupe à modifier le fichier

`chmod gorwx toto` interdit tout accès au fichier à tout le monde sauf le propriétaire (et root!)

Vous pouvez également utiliser la lettre a, qui est un raccourcis, pour ugo:

Ex : `chmod a+r` permet à tout le monde de lire le fichier.

Version étendue des permissions :

A ces droits (*) viennent ce rajouter SUID (4000) , SGID (2000) et Sticky-bit (1000) :

SUID, Sticky User IDentificator :

- Fichiers exécutables : le processus résultant d'un fichier exécutable SUID (4000) possède les permissions du propriétaire du fichier.

SGID, Sticky Group IDentificator :

- Fichiers exécutables : le processus résultant d'un fichier exécutable SGID (2000) possède les permissions du groupe du propriétaire du fichier.
- Répertoires : les fichiers créés dans le répertoires ont pour groupe du

propriétaire le groupe du propriétaire du répertoire.

Sticky-Bit :

- Fichiers exécutable : le processus résultant d'un fichier exécutable Sticky-bit (1000) reste en mémoire et son chargement est rapide.
- Répertoires : la suppression d'un fichier dans un répertoire Sticky-bit n'est possible que pour le propriétaire.

Comment écrire la permission "étendue" ?

Le SUID s'écrit S (à la place du x du propriétaire) si le propriétaire n'a pas le droit d'exécuter le fichier,

sdans le cas contraire

Le SGID s'écrit S (à la place du x du groupe propriétaire) si le groupe propriétaire n'a pas la permission d'exécuter le fichier, s dans le cas contraire

Le Sticky bit s'écrit T (à la place du x du droit "des autres utilisateurs") si les autres n'ont pas la permission d'exécuter le fichier, tdans le cas contraire.

(*) Note : ces permissions n'ont pas d'effets sur les systèmes de fichiers de type FAT.

Lorsque vous possédez un compte sur une station UNIX ou GNU/Linux (comme la vôtre :), le système ne vous connaît pas par votre nom mais par votre *User IDentificator* (UID), numéro qui vous est attribué au moment de la création du compte : rappelez vous, lorsque vous avez utilisé la commande useradd.

Cette commande permet aussi d'attribuer avec l'option u l'UID de l'utilisateur à

créer.

En plus des utilisateurs, les systèmes UNIX gère également la notion de groupe : chaque groupe, identifié bien sûr par son *Group Identifier* (GID), définit simplement un ensemble d'utilisateurs (c'est à dire un ensemble d'UIDs). Lorsque vous créez un compte, vous avez aussi également la possibilité de choisir le GID du groupe auquel appartient l'utilisateur soit en l'attribuant à un groupe existant avec l'option `g` soit en le créant (option `idem`). Il y a plusieurs écoles : certains préconisent qu'il faut créer un groupe pour chaque utilisateur, d'autres disent de rassembler tous les utilisateurs dans un groupe dédié appelé `users`.

Pour rappel l'UID et le GID de chaque utilisateur sont stockés dans le fichier `/etc/passwd`. Lorsque vous vous "loguez" grâce à `l'invite`, le système d'authentification va consulter ce fichier pour vérifier la validité de votre nom et de votre mot de passe et en cas de succès, il renvoie l'UID et le GID de l'utilisateur autorisé à se connecter. Ce fichier est d'ailleurs celui qui définit le nom de l'utilisateur : vous pouvez renommer un utilisateur en éditant ce fichier (mais ne modifiez surtout pas son UID !)

Un utilisateur peut bien évidemment appartenir à plus d'un groupe. A cette fin, le fichier `/etc/group` contient, pour chaque groupe, son GID et les noms d'utilisateurs membres (sauf ceux déclarés par défaut dans `/etc/passwd`). Il faut savoir qu'à tout moment, chaque fichier, chaque tâche, chaque périphérique, etc ... possède exactement un UID et un GID. En plus de posséder un UID et un GID, chaque fichier est aussi défini par un bit, *Binary digiT* de permission spécial : le bit `setuid`. A quoi sert-il ?

Sous UNIX, il y a une règle d'or en ce qui concerne les privilèges, la règle d'endossement : quand un processus est lancé, il acquiert les droits de celui qui

l'a lancé (évidemment, cela crée des problèmes). Ainsi, si un utilisateur veut modifier son mot de passe, il ne le pourra pas si on se contente de cela ; en effet, le fichier `/etc/passwd` n'est accessible en écriture que pour le root ! D'où l'intérêt du bit `setuid`. En temps normal, il est positionné à 0, et la règle d'endossement s'applique. Mais quand il est positionné à 1, il est lancé avec les droits de son propriétaire, c'est-à-dire les droits de root, et l'utilisateur pourra modifier son mot de passe.

Nous allons maintenant voir comment on utilise tout ceci pour gérer la sécurité. Les UID et GID (*) de valeur 0 sont ceux dont les droits ne sont pas vérifiés et sont par conséquent réservés au root. Vous pouvez donc changer le nom du root en modifiant le nom associé à UID 0, ce qui est vraiment une très mauvaise idée !

Nous avons vu que chaque fichier (entre autres) possède un UID et un GID. En tapant `ls -l`, vous affichez directement le nom de l'utilisateur et du groupe qui correspondent à l'UID et au GID plutôt que les valeurs numériques. L'UID du fichier est ce qu'on appelle habituellement le "propriétaire" du fichier et le GID du fichier le "groupe" du fichier.

Il faut d'ailleurs bien comprendre que ces deux champs sont indépendants : un fichier peut très bien faire partie d'un groupe dont le propriétaire n'est pas membre. Lire l'UID et le GID d'un fichier, est utile, mais il faut aussi pouvoir les modifier :

`chown`, *CHange OWNeR* permet (entre autres) de modifier le propriétaire d'un fichier : donne un fichier à quelqu'un d'autre. Toutefois, seul le root peut l'utiliser, les simples utilisateurs n'ont aucun autre moyen de se débarrasser d'un fichier que de l'effacer :

```
chown nom_de_propriétaire fichiers
```

Ex : `chown toto titi.txt` change de propriétaire : le nouveau propriétaire est toto Vous pouvez entrer directement l'UID au lieu du nom et faire beaucoup d'autres :

man `chown` :

Ex : `chown 501 titi.txt`

`chgrp`, *CHange GRouP* symétrique à la commande `chown` : modifie le groupe d'un fichier.

Contrairement à `chown`, elle peut être appelée par un simple utilisateur, à condition qu'il soit membre du groupe auquel il veut donner le fichier. Le root n'a naturellement aucune restriction de ce type : `chgrp nom_de_groupe fichiers`

Ex : `chgrp toto titi.txt` change de groupe : le nouveau groupe est toto

Vous pouvez entrer directement l'UID au lieu du nom et faire beaucoup d'autres : man `chgrp` : ex : `chgrp 501 titi.txt`

(*) Note : en fait, les explications ci-dessus sont simples car en effet hormis l'uid, il existe deux autres données le RUID, *Real UID*, l'UID du propriétaire du fichier (il est immuable) et l'EUID, *Effective UID*, l'UID qui est pris en compte lors de l'exécution du *processus* (process).

Il détermine les droits d'accès du process au système. Normalement, RUID = EUID => le process hérite des droits de l'appelleur, la règle de l'endossement. Mais, quand le bit `setuid` est positionné par un process, l'euid du process a pour valeur l'uid du propriétaire du fichier.

Donc, si le process est setuid root, il aura pour ruid celui de l'utilisateur qui l'a lancé, mais l'euid du root. Ainsi, dans le cas de /etc/passwd qui doit être modifié par un utilisateur voulant changer de password, comme /etc/passwd a le bit setuid root, le kernel modifie l'euid de /etc/passwd qui devient celui du root afin que l'utilisateur puisse y avoir accès en écriture pour modifier son mot de passe (rappelons que le ruid est immuable).

En effet, un process n'a pas besoin d'avoir toujours l'EUID root. Il a donc la possibilité de modifier son euid. En fait, quand l'euid du programme est modifié (/etc/passwd en l'occurrence), son ancien euid (qui est égal au ruid de l'utilisateur qui a lancé le programme) est mémorisé dans un bit de sauvegarde (le dernier à apprendre) : le SUID, *Saved UID*.

Par conséquent, pour limiter les attaques (en effet, si le programme garde son euid root n'importe quel utilisateur invité qui exécute /etc/passwd aura les droits en écriture dessus !), le programme va modifier son euid au plus vite en ruid. L'ancien euid (celui du root) est placé dans le suid et le ruid est copié dans l'euid. Puis, plus tard, quand les *appels système* (syscalls) nécessitent des privilèges correspondant au propriétaire du fichier, on remplace le suid dans l'euid (gain de temps).

EXERCICES

EXERCICE 1 :

1- En tant que root Créez un répertoire nommé « plan » contenant un fichier test.txt

Qui contient « bonjour tout le monde »

Et créer les groupes suivants avec leur utilisateurs membre :

grads = user3, user4 ; porfs =user5 ; interns = user6

```
# mkdir /plan ; echo bonjour tout le monde > /plan/test.txt
```

2- Changez-les permissions du dossier plan

- Le propriétaire et « root » a contrôle totale
- Le groupe propriétaire « Sales » dont ses membres peuvent accéder au répertoire sans modifier son contenu
- Tout le monde il a juste le droit de lire les attributs du répertoire

```
# chown root.Sales /plan
```

```
# chmod 754 /plan
```

```
# su - user1 (testez l'accès au répertoire cd /plan , ls -ld /plan , test.txt )
```

```
# su - user2 (testez l'accès au répertoire cd /plan , ls -ld /plan ; cat /plan/test.txt )
```

3- En tant que root créez dans la racine un 2em répertoire « trade » avec les droits par défaut

- Propriétaire root ; Groupe sales ; Tout le monde

```
# mkdir /trade ; chown root.sales /trade
```

4- Switchez en tant que user1 et créez un fichier file1 au sien du répertoire trade

Visualisez les attributs

```
# su - user1
# touch /trade/file1
# ls -ld /trade ; ls -l /trade
```

5- Switcher en tant que user2 et créez un fichier file2 au sien du répertoire trade

Visualisez les attributs

```
# su - user2
# touch /trade/file2 ; ls -ld /trade ; ls -l /trade
```

6- Switchez en tant que root et positionnez la permission SGID sur le répertoire trade

```
# su - root
# chmod 2777 /trade
```

7- Switchez en tant que user1 et créez un fichier nomme file3 dans le répertorie trade

Listez les attributs

Switcher en tant que user2 et créez un fichier nomme file4 dans le répertorie trade

Visualiser les attributs qu'est-ce que vous constatez ?

```
# su - user1
```

```
# touch /trade/file3 ; ls -ld /trade ; ls -l /trade
# su - user2
# touch /trade/file4 ; ls -ld /trade ; ls -l /trade
```

EXERCICE 2 :

- 1- en tant que root Créez un répertoire dans la racine nommé « Shares » avec les droits tout le monde contrôle totale et positionnez le sticky-bit

```
# mkdir /Shares ; chmod 777 /Shares
# chmod 1777 /Shares
```

- 2- Switcher en tant que user1 créez au sien de Shares un répertoire nomme classe1 et dans le répertoire classe1 créez un fichier nommé lab01.txt - visualisez les attributs

```
# su - user1
# mkdir /Shares/classe1 ; touch /Shares/classe1/lab01.txt
# ls -lR /Shares
```

- 3- Switcher en tant que user2 créez au sien de Shares un répertoire nomme classe2 et au sien de classe2 créez un fichier nommé lab02.txt - visualisez les attributs

```
# su - user2
# mkdir /Shares/classe2 ; touch /Shares/classe2/lab02.txt
# ls -lR /Shares
```

- 4- En tant que user2 essayez de supprimer le dossier classe1 que ce que vous constatez ?

EXERCICE 3: Les Droits ACL

1- Créer le répertoire /opt/research

```
# mkdir /opt/research
```

2- Changer le groupe propriétaire du répertoire /opt/research par le groupe Sales

```
# chgrp grads /opt/research
```

3- Positionner le droit SGID au répertoire /opt/research 2770

```
# chmod 2770 grads /opt/research
```

4- Ajouter le groupe profs avec les droits lecture écriture et exécution sur le répertoire /opt/research

```
# setfacl -m g :profs :rwx /opt/research
```

5- Ajouter le groupe group interns avec les droits lecture et exécution sur le répertoire /opt/research

```
# setfacl -m g :interns :rx /opt/research
```

6- Garnit l'héritage des permissions pour les nouveaux fichier qui vont être créer avec les droits (group profs=rw) et (groupe interns=r).

```
# setfacl -m d:g:profs :rw /opt/research
```

```
# setfacl -m d:g:interns :r /opt/research
```

7- Lister les acl sur le répertoire /opt/research

Epreuve de Synthèse
Durée 1h30

EXERCICE 1 (10 pts)

1. Quelle est l'erreur dans la commande suivante?

```
$ copy file1 file2
```

Answer: Should be "cp" not "copy".

2. Quelle est l'erreur dans la commande suivante?

```
$ RM FILE BILL-FILE
```

UNIX is case-sensitive, so it reports
"RM: Command not found".

3. Quelle est l'erreur dans la commande suivante?

```
$ cp names it | cat
```

Nothing input to pipe.

4. Supposons que les commandes suivantes sont exécutées :

```
$ whoami
```

```
cschung
```

```
$ ls -l
```

```
-rw-r--r-- 1 horner cs 154 Feb 4 16:38 letter1
```

```
-rw-r--r-- 1 horner cs 64 Feb 4 15:00 names
```

```
dr-xr-xr-x 2 horner cs 512 Feb 8 18:01 secret/
```

```
$ ls -l secret
```

```
-rw-r--r-- 1 horner cs 154 Feb 9 17:32 letter1
```

```
$ cp names secret/letter1
```

Que se passe-t-il quand les commandes sont exécutées? Justifier votre réponse

```
cp: cannot create regular file 'secret/letter1': Permission denied  
Because the user cschung does not have write permission on the file  
'secret/letter1'
```

5. Supposons que les commandes suivantes sont exécutées :

```
$ whoami
```

```
cschung
```

```
$ ls -l
```

```
total 4
```

```
-rw-r--r-- 1 cschung cs 29 Oct 11 12:01 letter1
```

```
-rw-r--r-- 1 cschung cs 0 Oct 11 12:37 names
```

```
d----- 2 cschung cs 20 Oct 11 12:38 secret
```

```
$ ls secret/
```

```
ls: secret/: Permission denied
```

```
$ chmod u+rw secret/
```

```
$ ls secret/
```

Que se passe-t-il quand les commandes sont exécutées? Justifier votre réponse

```
ls: secret/letter1: Permission denied
```

```
Because the user cschung does not have execute permission on the directory 'secret'
```

6. Supposons que les commandes suivantes sont exécutées :

```
$ whoami
```

```
horner
```

```
$ ls -l letter1
```

```
-rw-r--r-- 1 cschung cs 29 Oct 11 12:01 letter1
```

```
$ cat letter1
```

```
#!/bin/sh
```

```
echo "Hello World"
```

```
$ chmod 700 letter1
```

Que se passe-t-il quand les commandes sont exécutées? Justifier votre réponse

There is an error message after chmod because horner is not the owner of the file letter1.

Because the user horner is not the owner of the file letter1, and only the owner can change the permission.

7. Donner le sens du script suivant:

```
#!/bin/sh
users=`who | wc -l`
echo "$users";
```

It counts and prints the number of users logged on the machine.

8. Le programme shell suivant doit calculer le factoriel de n (n! pour n>=1) remplir les blancs dans le programme.

```
#!/bin/sh
echo "Enter n: "
_____
output=1
i=1
while [ _____ ]
do
    output=_____
    i=_____
_____
echo "The factorial of $n is $output"
```

```
#!/bin/sh
echo "Enter n: "
read n                                     (1 mark)
output=1
i=1
while [ $i -le $n ]                       (2 marks)
do
    output=`expr $output \* $i`           (2 marks)
    i=`expr $i + 1`                       (1 mark)
done                                       (1 mark)
echo "The factorial of $n is $output"
```

9. Le programme shell suivant doit calculer la somme des ses arguments et produit le résultat suivant à titre d'exemple:

```
$ calculate 1 2 3
```

```
The sum of 3 numbers is 6
```

```
$ calculate 1 2 3 4 5 6 7 8 9 10
```

```
The sum of 10 numbers is 55
```

Compléter le programme shell suivant sans définir de nouvelles variables.

```
#!/bin/sh
sum=0;
echo -n "The sum of $# numbers is "
while [ $1 ]
```

```
do
    sum=`expr $sum + $1`
    shift
done
echo "$sum"
```

EXERCICE 2 (10 pts)

1. Compte utilisateurs

A) Créer trois compte utilisateurs, `mtrma`, `mtrmb`, and `mtrmc`.

B) Créer un nouveau groupe, `midterm`, avec `GID=5000`,

C) Ajouter chaque utilisateurs, `mtrma`, `mtrmb`, et `mtrmc` dans le groupe `midterm`. Chaque groupe utilisateur doit rester dans son groupe.

D) Changer l'ID du groupe primaire de l'utilisateur `mtrma` à 5001.

-- ANSWER:

A)

```
adduser mtrma
adduser mtrmb
adduser mtrmc
```

B)

```
addgroup -gid 5000 midterm
```

or

```
groupadd -g 5000 midterm
```

C)

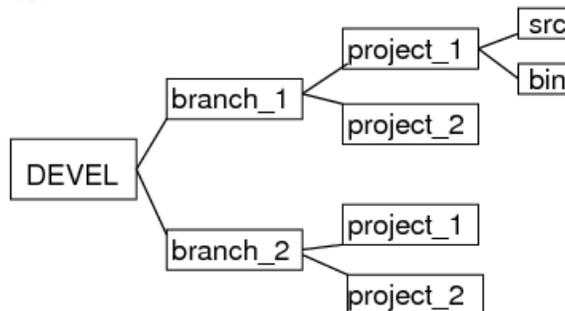
```
usermod -a -G midterm mtrma
usermod -a -G midterm mtrmb
usermod -a -G midterm mtrmc
```

D)

```
groupmod -g 5001 mtrma
```

2. fichiers , droits d'accès

A) Créer l'arborescence suivante:



B) copier le fichier `/etc/hosts` et le répertoire `/etc/default` avec son contenu dans le répertoire `DEVEL/branch_1/project_2`

C) Donner à l'utilisateur `mtrma` la propriété du répertoire `DEVEL/branch_1` ainsi de de son contenu.

Donner à l'utilisateur `mtrmb` la propriété du répertoire `DEVEL/branch_2` ainsi de de son contenu.

D) modifier le droit d'accès du répertoire `DEVEL/branch_1` ainsi que son contenu afin d'être en lecture, écriture et exécutable uniquement pour les trois utilisateurs.

E) modifier le droit d'accès du répertoire `DEVEL/branch_2/project_2` afin d'être en lecture, écriture et

exécutable pour tout le monde.

F) Archiver le répertoire `DEVEL` en créant un fichier de type tar et zip.

A)

```
mkdir -p DEVEL/branch_1/project_1/src  
mkdir DEVEL/branch_1/project_1/bin  
mkdir DEVEL/branch_1/project_2  
mkdir -p DEVEL/branch_2/project_1  
mkdir DEVEL/branch_2/project_2
```

B)

```
cp /etc/hosts DEVEL/branch_1/project_2  
cp -a /etc/default DEVEL/branch_1/project_2
```

C)

```
chown -R mtrma DEVEL/branch_1  
chown -R mtrmb DEVEL/branch_2
```

D)

```
chown -R :midterm DEVEL/branch_1
```

alternatively, you can use command `chgrp`:

```
chgrp -R midterm DEVEL/branch_1
```

then set `rxw` permissions on the directory tree for the user and group only:

```
chmod -R 770 DEVEL/branch_1
```

E)

```
chmod 1777 DEVEL/branch_2/project_2
```

F)

```
tar -zcvf DEVEL.tgz DEVEL
```