Unix/Linux II

DUT 1^{re} année

Université de Marne La vallée

1. Protection de fichiers
   - Droits d'accès aux fichiers
   - Visualisation des droits d'accès
   - Modification des droits d'accès
   - Initialisation des droits d'accès
   - Changement de propriétaire et de groupe

2. Métacaractères et expressions régulières
   - Métacaractères
   - Expression régulières
   - Recherche de chaîne dans un fichier : `grep`
   - Recherche d'un fichier `find`
   - La commande `sed`

Chaque fichier (ou répertoire) possède un ensemble d'attributs définissant les droits d'accès à ce fichier pour tous les utilisateurs du système.

3 types d'utilisateurs

le propriétaire	`u`
le groupe	`g`
les autres	`o`

3 types de droits

lécture	`r`
écriture	`w`
exécution	`x`

Pour les fichiers, les droits sont exprimés par une chaîne de 10 caractères :`tuuugggooo`

`t` :type du fichier :

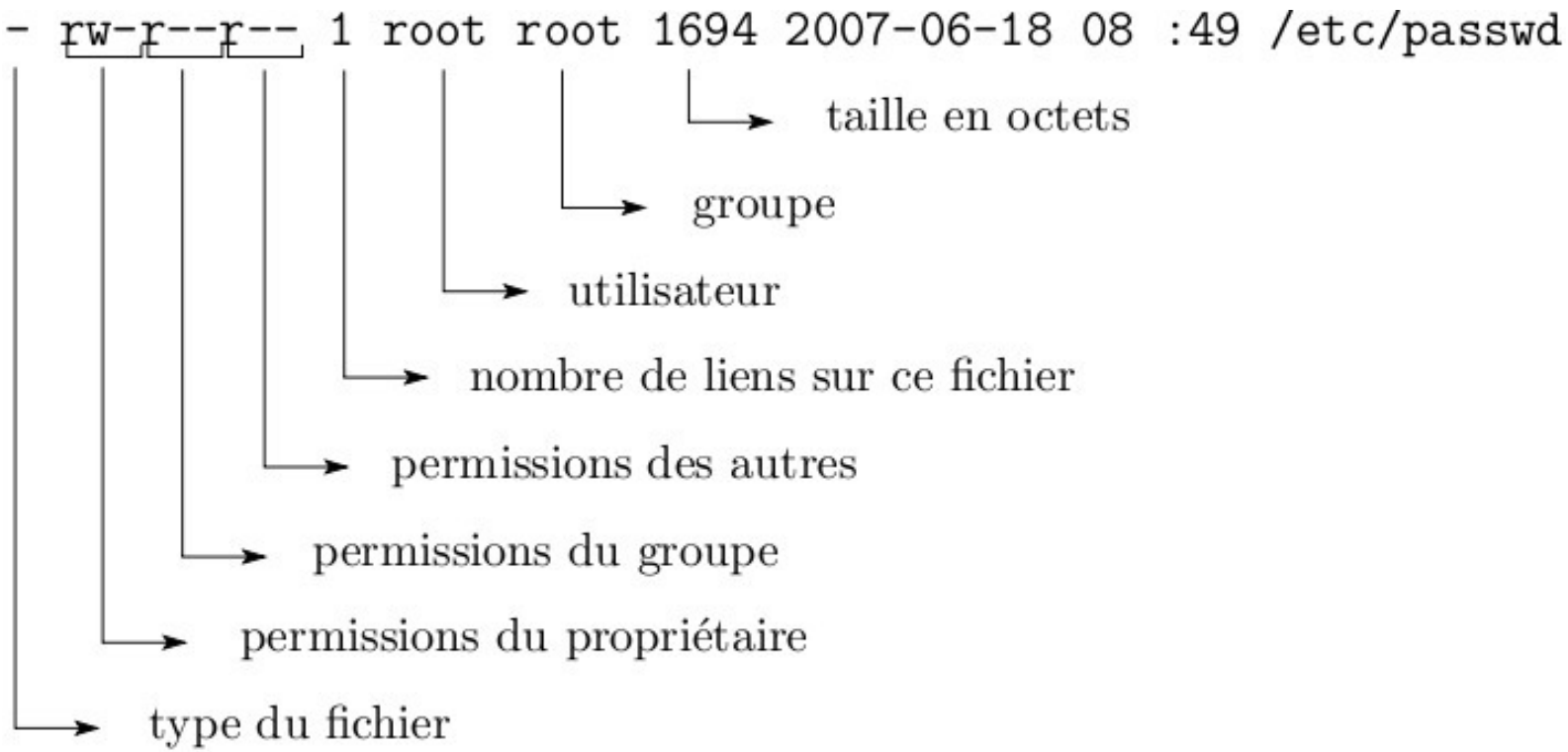
Fichier ordinaire	`-`
Répertoire	`d`
Lien symbolique	`l`
Fichier spécial	`c` ou `b`
Socket	`s`

Le super-utilisateur (`root`) a tous les droits

Pour visualiser les droits, on utilise `ls -l`.
Présence du droit si une lettre `r`, `w` ou `x`. Abscence du droit si -

Exemple

`ls -l /etc/passwd` donne



On peut représenter la protection d'un fichier par trois chiffres (un nombre octal).

Exemple :`rwxrw-r-x` est représenté par 765.
Une lettre est $\equiv$ à 1 et un tiret à 0
$\Rightarrow$
`rwxrw-r-x` $=$ 111 110 101 $=$ 765
car
$111 = 2^2 + 2^1 + 2^0 = 4 + 2 + 1 = 7$
$110 = 2^2 + 2^1 = 4 + 2 = 6$
$101 = 2^2 + 2^0 = 4 + 1 = 5$

Droits	valeur octale
`---`	0
`--x`	1
`-w-`	2
`-wx`	3
`r--`	4
`r-x`	5
`rw-`	6
`rwx`	7

`chmod` : changer le mode de protection d'un fichier

Syntaxe

```
chmod mode nom_fichier
```

Deux modes d'utilisation :

Mode absolu : `mode` représenté par un nombre octal

Exemple `chmod 765 fich`

```
nicolas@lancelot: /Bureau$ touch ess.txt
nicolas@lancelot: /Bureau$ ls -l ess.txt
-rw-rw-r- 1 nicolas nicolas 0 août 29 23:42 ess.txt
nicolas@lancelot: /Bureau$ chmod 764 ess.txt
nicolas@lancelot: /Bureau$ ls -l ess.txt
-rwxrw-r- 1 nicolas nicolas 0 août 29 23:42 ess.txt
```

mode symbolique :

`mode` indique de quelle façon les droits d'accès doivent être modifiés. Il se décompose en `[qui] op accès` où

- `qui` (optionnel) indique quelles classes sont concernées par `chmod`, est composé de une ou plusieurs lettres (`u`, `g` et `o`). Si aucune lettre alors tous les types d'utilisateurs sont concernés (ou `a` pour all).

- `op` peut être :
  `+` pour ajouter des droits d'accès
  `-` pour enlever des droits d'accès
  `=` pour affecter des droits d'accès.
- `accès` est une combinaison des lettres `r`, `w` et `x` qui spécifient les types d'accès.

Exemple : `chmod a=r,u+w fich`

Exemple précédent :

```
nicolas@lancelot: $ ls -l ess.txt
-rwxrw-r- 1 nicolas nicolas 0 sept. 1 11:50 ess.txt
nicolas@lancelot: $ chmod u-x,g-w ess.txt
nicolas@lancelot: $ ls -l ess.txt
-rw-r-r- 1 nicolas nicolas 0 sept. 1 11:50 ess.txt
```

Au moment de créer un fichier, des droits d'accés par défaut sont donnés à ce fichier. La commande `umask` seule permet de consulter ces droits d'accés. La valeur retournée joue le rôle de masque sur les droits d'accès d'un fichier à sa création. Les droits d'accès sont obtenus aprés l'opération logique suivante : `mode & (~ masque)` où `&` est le ET logique et `~` est le NON logique.
Pour une commande de création de fichier, le mode par défaut est `rw-rw-rw-`. Le masque est appliqué à ce mode.

Exemple :

`umask` donne 022 soit en codage binaire 000 010 010

le mode par défaut est `rw-rw-rw-` soit en codage binaire 110 110 110

`mode`	110	110	110
`~ masque`	111	101	101
`mode & (~ masque)`	110	100	100

soit `rw-r--r--`.

Le mode par défaut pour la création d'un répertoire est `rwxrwxrwx`.

Vous pouvez "donner" un fichier vous appartenant à un autre utilisateur.

```
chown nouveau_propriétaire nom_fichier
```

ou changer le groupe auquel le fichier est rattachÃ©

```
chgrp nouveau_groupe nom_fichier.
```

Si vous êtes à la recherche d'un fichier qui commence par la lettre *a*, en faisant `ls`, vous voudriez voir que les fichiers commençant par *a*. De même si vous voulez appliquer une commande à certains fichiers mais pas à d'autres.
C'est le but des métacaractères, ils vous permettent de faire une sélection de fichiers suivant certains critères.

Les métacaractères

- sont des caractères génériques permettant de désigner un ensemble d'objets et
- s'appliquent aux arguments des commandes qui désignent des noms de fichiers.

Le Shell permet de générer une liste de noms de fichier en utilisant les caractères spéciaux suivants :

`*`	toutes chaînes de caractères, y compris la chaîne vide ex : `a*b` tous les noms de fichiers commençant par `a` et finissant par `b`
`?`	caractère quelconque ex : `a?b` tous les noms de fichier commençant par `a`, suivi d'un caractère et finissant par `b`
`[...]`	un caractère quelconque $\in$ à la liste donnée entre crochets Le `-` permet de représenter un intervalle. ex : `a[a-z0-9A-Z]b` désigne tous les noms de fichiers commençant par `a` suivi d'un caractère alphanumérique et finisant par `b`
`[!...]`	une liste de caractères à exclure ex : `a[!a-z]b` tous les noms de fichiers commençant par `a` suivi d'un caractère autre qu'un caractère alphabétique en minuscule et finissant par `b`

Exemple Si le répertoire courant contient :
`fich1.bin fich1.txt fich2.txt fich10.txt fichier.txt readme zzz` Alors :

`fich1*`	
`fich*.txt`	
`fich[0-9]*.txt`	
`???`	

Exemple Si le répertoire courant contient :
`fich1.bin fich1.txt fich2.txt fich10.txt fichier.txt readme zzz` Alors :

`fich1*`	`fich1.bin fich1.txt fich10.txt`
`fich*.txt`	
`fich[0-9]*.txt`	
`???`	

Exemple Si le répertoire courant contient :
`fich1.bin fich1.txt fich2.txt fich10.txt fichier.txt readme zzz` Alors :

`fich1*`	`fich1.bin fich1.txt fich10.txt`
`fich*.txt`	`fich1.txt fich2.txt fich10.txt fichier.txt`
`fich[0-9]*.txt`	
`???`	

Exemple Si le répertoire courant contient :
`fich1.bin fich1.txt fich2.txt fich10.txt fichier.txt readme zzz` Alors :

`fich1*`	`fich1.bin fich1.txt fich10.txt`
`fich*.txt`	`fich1.txt fich2.txt fich10.txt fichier.txt`
`fich[0-9]*.txt`	`fich1.txt fich2.txt fich10.txt`
`???`	

Exemple Si le répertoire courant contient :
`fich1.bin fich1.txt fich2.txt fich10.txt fichier.txt readme zzz` Alors :

`fich1*`	`fich1.bin fich1.txt fich10.txt`
`fich*.txt`	`fich1.txt fich2.txt fich10.txt fichier.txt`
`fich[0-9]*.txt`	`fich1.txt fich2.txt fich10.txt`
`???`	`zzz`

Les expressions régulières (comme les métacaractères) sont aussi des suites de caractères permettant de faire des sélections.
Une expression régulière peut être aussi simple qu'un mot exact à rechercher, par exemple `'Bonjour'`, ou aussi complexe que `'^ [a-zA-Z]'` qui correspond à toutes les lignes commençant par une lettre minuscule ou majuscule.
La syntaxe des expressions régulières utilise les notations suivantes :

`c`	correspond au caractère c
`\c`	banalise le métacaractère c
	ex : `\.`, `\*`, ...
`.`	caractérise n'importe quel caractère
`[...]`	n'importe quel caractère de l'ensemble spécifié
	- pour définir un intervalle

`[^ ...]`	n'importe quel caractère hors de l'ensemble spécifié
`^`	caractérise le début de ligne (`^` $\neq$ `[^ ...]`)
	ex : `^ abc` désigne une ligne commençant par `abc`
`$`	caractérise la fin de ligne
	ex : `abc$` désigne une ligne finissant par `abc`
`^ $`	ligne vide
`*`	0 à n fois le caractère la précédent
	ex : `a*` représente de 0 à n fois `a`
	`aa*` représente de 1 à n fois `a`
	`.*` désigne n'importe quelle chaîne même vide
`\{n \}`	nombre de répétition `n` du caractère placé devant

Exemple : `[0-9]\{4 \} $` : du début à la fin du fichier `$`, recherche les nombres `[0-9]` de 4 chiffres `\{4 \}`

La commande **grep** permet de rechercher une chaîne de caractères dans un fichier.

Syntaxe : `grep [option] motif nom_fichier`

Les options sont les suivantes :

- `-v` affiche les lignes ne contenant pas la chaîne
- `-c` compte le nombre de lignes contenant la chaîne
- `-n` chaque ligne contenant la chaîne est numérotée
- `-x` ligne correspondant exactement à la chaîne
- `-l` affiche le nom des fichiers qui contiennent la chaîne

Exemple : le fichier carnet-adresse :

```
olivier:29:0298333242:Brest
marcel:13:0466342233:Gardagnes
myriam:30:0434214452:Nimes
yvonne:92:013344433:Palaiseau
```

On peut utiliser les expressions régulières avec **grep**. Si on tape

```
grep ^ [a-d] carnet-adresse
```

On va obtenir tous les lignes commençant par les caractères compris entre **a** et **d**. Dans notre exemple, on n'en a pas, d'où l'absence de sortie.

```
grep Brest carnet-adresse
```

Permet d'obtenir les lignes contenant la chaîne de caractère `Brest` :

```
olivier:29:0298333242:Brest
```

Il existe aussi les commandes **fgrep** et **egrep** équivalentes.

La commande **find** permet de retrouver des fichiers à partir de certains critères.

Syntaxe :

```
find <répertoire de recherche> <critères de recherche>
```

critères de recherche :

`-name`	recherche sur le nom du fichier
`-perm`	recherche sur les droits d'accès
`-link`	recherche sur le nombre de liens
`-user`	recherche sur le propriétaire
`-group`	recherche sur le groupe auquel appartient le fichier
`-type`	recherche sur le type
`-size`	recherche sur la taille
`-atime`	recherche sur la date de dernier accès en lecture
`-mtime`	recherche sur la date de dernière modification du fichier
`-ctime`	recherche sur la date de création du fichier

On peut combiner les critères avec des opérateurs logiques :

- `critère1 critère2` ou `critère1 -a critère2` ≡ au ET logique,
- `!critère` ≡ NON logique,
- `\(critère1 -o critère2\)` ≡ OU logique,

L'option `-print` est indispensable pour obtenir une sortie.

Remarque

La commande `find` est récursive, *i.e.* scruter dans les répertoires, et les sous répertoires qu'il contient.

Recherche par nom de fichier

Pour chercher un fichier dont le nom contient la chaîne de caractères `toto` à partir du répertoire `/usr` :

```
find /usr -name toto -print
```

- Si le(s) fichier(s) existe(nt) ⇒ sortie : `toto`
- En cas d'échec, vous n'avez rien.

Pour rechercher tous les fichiers se terminant par `.c` dans le répertoire `/usr` :

```
find /usr -name " *.c " -print
```

⇒ toute la liste des fichiers se terminant par `.c` sous les répertoires contenus dans `/usr` (et dans `/usr` lui même).

Recherche suivant la date de dernière modification

Ex :Les derniers fichiers modifiés dans les 3 derniers jours dans toute l'arborescence (/) : `find / -mtime 3 -print`

Recherche suivant la taille

Ex :Connaître dans toute l'arborescence, les fichiers dont la taille dépasse 1Mo (2000 blocs de 512Ko) :

```
find / -size 2000 -print
```

Recherche combinée

Ex :Chercher dans toute l'arborescence, les fichiers ordinaires appartenant à olivier, dont la permission est fixée à 755 :

```
find / -type f -user olivier -perm 755 -print
```

Ex : Recherche des fichiers qui ont pour nom `a.out` et des fichiers se terminant par `.c` :

```
find . \ ( -name a.out -o -name " *.c " \ ) -print
```

Commandes en option :

En dehors de `-print` on dispose de l'option `-exec`. Le `find` couplé avec `exec` permet d'exécuter une commande sur les fichiers trouvés d'après les critères de recherche fixés. Cette option attend comme argument une commande, suivie de {}\ .

Ex : recherche des fichiers ayant pour nom `core`qu'on efface

```
find . -name core -exec rm {}\
```

Ex : les fichiers ayant pour nom `core` seront détruits, pour avoir une demande de confirmation avant l'exécution de `rm` :

```
find . -name core -ok rm {}\
```

Autres subtilités : Une fonction intéressante de `find` est de pouvoir être utilisé avec d'autres commandes

Ex : `find . -type f -print | xargs grep toto`

Rechercher dans le répertoire courant tous les fichiers normaux (sans fichiers spéciaux), et rechercher dans ces fichiers tous ceux contenant la chaîne `toto`.

`sed` est éditeur ligne non interactif, il lit les lignes d'un fichier une à une (ou provenant de l'entrée standard) leur applique un certain nombre de commandes d'édition et renvoie les lignes résultantes sur la sortie standard. Il ne modifie pas le fichier traité, il écrit tout sur la sortie standard.

Syntax `sed -e 'programme sed' fichier-a-traiter`
ou `sed -f fichier-programme fichier-a-traiter`

On disposez de l'option `-n` qui supprime la sortie standard par défaut, `sed` va écrire uniquement les lignes concernées par le traitement (sinon il écrit tout même les lignes non traitées).

L'option `-e` n'est pas nécessaire quand on a une seule fonction d'édition.

`sed` est une commande très riche (pour plus de détails `man sed`)

La fonction de substitution :s

`s` permet de changer la 1^{re} ou toutes les occurences d'une chaîne par une autre.

Syntaxe :

- `sed "s/toto/TOTO/" fichier` va changer la 1^{re} occurence de la chaîne `toto` par `TOTO`
- `sed "s/toto/TOTO/3" fichier` va changer la 3^{me} occurence de la chaîne `toto` par `TOTO`
- `sed "s/toto/TOTO/g" fichier` va changer toutes les occurences de la chaîne `toto` par `TOTO`
- `sed "s/toto/TOTO/p" fichier` en cas de remplacement imprime les lignes concernées
- `sed "s/toto/TOTO/w resultat" fichier` en cas de substitution la ligne en entrée est inscrite dans un fichier résultat

La fonction de substitution peut être utilisée avec une expression régulière.
`sed -e "s/[Ff]raise/FRAISE/g" fichier` substitue toutes les chaînes `Fraise` ou `fraise` par `FRAISE`

La fonction de suppression :d

La fonction de suppression `d` supprime les lignes comprises dans un intervalle donné.

Syntaxe : `sed "20,30d" fichier`

Cette commande va supprimer les lignes 20 à 30 du fichier `fichier`. On peut utiliser les expressions régulières :

- `sed "/toto/d" fichier` : supprime les lignes contenant la chaîne `toto`
- `sed "/toto/!d" fichier` : supprime toutes les lignes ne contenant pas la chaîne `toto`

En fait les lignes du fichier d'entrée ne sont pas supprimées, elles le sont au niveau de la sortie standard.

Les fonctions : `p`, `l` et`=`

- `p` (print) affiche la ligne sélectionnée sur la sortie standard. Elle invalide l'option `-n`.
- `l` (list) affiche la ligne sélectionnée sur la sortie standard avec en plus les caractères de contrôles en clair avec leur code ASCII (deux chiffres en octal).
- `=` donne le num de la ligne sélectionnée sur la sortie standard.

Ces trois commandes sont utiles pour le débogage, (mise au point des programmes `sed`)
`sed "/toto/=" fichier` : afficher le numéro de la ligne contenant la chaîne `toto`.

Les fonctions : `q, r` et`w`

- `q` (quit) interrompre l'exécution de **`sed`**, la ligne en cours de traitement est affichée sur la sortie standard (uniquement si `-n` n'a pas été utilisée).
- `r` (read) lit le contenu d'un fichier et écrit le contenu sur la sortie standard.
- `w` (write) écrit la ligne sélectionnée dans un fichier.

`sed "/^ toto/w resultat" fichier` : Ecrire dans le fichier `resultat` toutes les lignes du fichier `fichier` commençant par la chaîne `toto`.