Enoncé du TP 2 Système

C. Pain-Barre

INFO - IUT Aix-en-Provence

version du 27/9/2013

Exercice 1 (à traiter en TD)

Déterminer les permissions d'un utilisateur sur un fichier en étudiant les informations fournies par **ls**, *et assimiler les possibilités offertes par ces droits.*

Supposons qu'un utilisateur tape la commande suivante :

```
$ ls -al
drwxr---wx  3 truc    bidule   12288 18 oct.  16:21 .
drwxr-xr-x  4 root    root      4096 23 août  2010 ..
-r-xrw-rwx  1 truc    machin     333 22 mars  08:52 fic
drw-r---wx  2 machin  bidule    4096  8 sep.  10:10 rep
```

1. Qui est le propriétaire du répertoire de travail ?
2. Quelles sont les permissions de l'utilisateur truc appartenant aux groupes machin et bidule sur `fic` ?
3. Quelles sont les permissions de l'utilisateur bidule appartenant aux groupes machin et truc sur `fic` ?
4. Quelles sont les permissions de l'utilisateur bidule appartenant aux groupes machin et truc sur `rep` ?
5. Que peut-on dire des utilisateurs qui peuvent supprimer `fic` ?
6. Est-il possible que `rep` ait pu être créé tel quel dans le répertoire de travail ?
7. Quels sont les droits effectifs de l'utilisateur machin appartenant aussi au groupe machin sur `rep` ?

> ✍ Démarrer les PC sous Linux. Tous les exercices sont à faire via une connexion SSH sur allegro.
>
> Rappels :
>
> - le prompt indique la machine sur laquelle bash s'exécute ;
> - une connexion SSH sur allegro s'établit à partir d'un bash tournant sur le PC par la commande :
>
> **ssh** *etxxxx*`@allegro`
>
> où *etxxxx* est votre nom d'utilisateur sur allegro.

Exercice 2

Étude et utilisation de **umask** *qui limite les permissions des prochains fichiers créés*

1. Se placer dans votre répertoire `tpunix`
2. Afficher le contenu de ce répertoire avec les informations détaillées
3. En principe, un masque de création de fichiers par défaut est fixé pour les utilisateurs, via un fichier de configuration (étudié plus tard). Exécuter **umask** (seul) pour afficher la valeur actuelle du masque :
   (a) à partir de ce qu'a écrit **umask** (le 0 de gauche doit être ignoré), en déduire quelles seront les permissions des prochains fichiers ordinaires de données (texte, image, etc.) créés. L'écrire sous la forme **rwxrwxrwx**.

   (b) en déduire aussi celles des prochains répertoires créés
   (c) Taper **vi** `essai1.txt` pour éditer un nouveau fichier `essai1.txt`. Entrer en mode insertion (avec **i**) pour saisir une ligne de texte quelconque, puis sortir du mode insertion (avec Esc) afin de sauver le fichier et quitter **vi** (avec **:wq**)
   (d) créer un nouveau répertoire `rep1`
   (e) Taper **ls -ld essai1.txt rep1** pour afficher les informations détaillées sur `essai1.txt` et sur `rep1` (pas sur son contenu)
   (f) Observer leurs permissions et comparer avec vos déductions aux questions a. et b.
4. Modifier le masque des permissions de telle sorte qu'il n'y ait aucune interdiction pour le propriétaire et aucune permission pour le groupe et les autres puis :
   (a) avec **vi**, créer un nouveau fichier `essai2.txt` contenant une nouvelle ligne de texte quelconque, puis sauver et quitter **vi**
   (b) créer un nouveau répertoire `rep2`
   (c) afficher les informations détaillées sur `rep1`, `essai1.txt`, `rep2` et `essai2.txt`.
   (d) vérifier que les permissions de `essai2.txt` et `rep2` correspondent à ce qui a été demandé en 4
   (e) remarquer que les permissions de `essai1.txt` et `rep1` n'ont pas changé car le masque n'a pas d'incidence sur les fichiers/répertoires existants
5. Simplement par déduction, déterminer quelles sont les différences entre un masque à `077` (qu'il fallait utiliser à la question 4) et un masque à `066` :
   (a) sur les permissions accordées lors de la création des fichiers ordinaires de données d'une part ;
   (b) sur les permissions accordées lors de la création des répertoires d'autre part.
6. Vérification de ces différences :
   (a) Positionner le masque à la valeur `066`
   (b) avec **vi**, créer un nouveau fichier `essai3.txt` contenant une ligne de texte, le sauver et quitter **vi**
   (c) créer un nouveau répertoire `rep3`
   (d) afficher les informations détaillées sur `rep1`, `essai1.txt`, `rep2`, `essai2.txt`, `rep3` et `essai3.txt`
   (e) Vérifier vos réponses à la question 5 en remarquant que `essai2.txt` et `essai3.txt` ont les mêmes permissions alors que celles de `rep2` et `rep3` diffèrent.
   (f) Observer à nouveau que les permissions des autres fichiers n'ont pas changé.
7. Fermer la connexion SSH en tapant **exit** (ou CTRL-D). Le prompt qui s'affiche ensuite doit être celui de votre PC.
8. Ouvrir une nouvelle connexion SSH sur allegro en tapant **ssh** *etxxxx*`@allegro` (le prompt doit vous permettre de vérifier que la connexion a réussi)
9. Vérifier la valeur du masque. Celui-ci a dû revenir à `0022`. Pourquoi ?

Exercice 3

Modification manuelle des permissions des fichiers avec **chmod**

Dans votre répertoire `tpunix` :

1. Utiliser le mode absolu de **chmod** pour fixer les permissions de `essai1.txt` à **`rwx------`**. Vérifier en demandant les informations détaillées sur ce fichier.
2. Utiliser le mode symbolique (sans l'opérateur =) de **chmod** pour fixer les permissions de `essai2.txt` à **`rw-rw---x`** (c.-à-d. ajouter les droits de lecture/écriture pour le groupe et le droit d'exécution pour les autres). Vérifier en demandant les informations détaillées sur ce fichier.
3. Modifier la protection du fichier `essai3.txt` en interdisant l'écriture pour le propriétaire et en ajoutant la lecture aux autres (pas au groupe). Vérifier en demandant les informations détaillées sur ce fichier.
4. Éditer `essai3.txt` sous **vi**. Insérer une ligne de texte et sauver sans quitter (avec **`:w`**). Remarquer la réaction de **vi**.
5. Quitter vi en sauvant quand même (si ! si ! c'est possible car **vi** est notre ami ☺, il faut taper **`:wq!`**)

> ⓘ Que **vi** sauve quand même le fichier en dépit des permissions est assez inattendu mais pas totalement anormal car le fichier vous appartient. Il lui suffit en effet de modifier temporairement les permissions du fichier, de le sauver, puis de remettre ses permissions comme elles étaient. Cela dit, c'est un effort de **vi** que ne font généralement pas les autres utilitaires.

Exercice 4

Utilisation de **mkdir** *et de* **chmod** *pour créer un environnement de travail comprenant un répertoire public, accessible à tous, et un répertoire totalement privé*

1. Se placer dans son répertoire d'accueil
2. Créer les répertoires `public` et `prive`
3. Modifier les permissions de `prive` de façon à ce que le propriétaire ait tous les droits et que le groupe et les autres n'en aient aucun.

> ✍ Ainsi, personne d'autre que vous ne pourra accéder aux fichiers de votre répertoire `prive`.

4. Si le masque vaut `022`, `public` a été créé avec des permissions permettant au groupe et aux autres de lire son contenu, de s'y placer et d'accéder (référencer) un fichier qu'il contient. Vérifier que les permissions de `public` sont conformes à cet objectif. Les modifier si ce n'est pas le cas (et remettre le masque à `022`). Vérifier aussi que tout le monde a au moins le droit de traverser votre répertoire d'accueil et y remédier si ce n'est pas le cas.

> ✍ Ainsi, tout le monde aura accès aux fichiers de votre répertoire `public`.

> 💣 **Attention ! ! ! N'autorisez personne à écrire dans vos répertoires, à moins que vous ayez une très bonne raison à cela...**

5. Copier dans `public` les fichiers `essai2.txt` et `cigale.txt` de `tpunix`
6. Expliquer pourquoi `essai2.txt` de `public` a été créé avec les permissions **`rw-r----x`** ?
7. Copier dans `prive` les fichiers `essai1.txt` et `essai3.txt` de `tpunix`

Exercice 5

Vérification de l'environnement de travail

1. Demander à l'un de vos voisins d'ouvrir un nouveau terminal sur sa session, de se connecter par SSH sur allegro, puis d'afficher le contenu de votre répertoire `public`. Il doit pouvoir le faire.
2. Lui demander d'éditer avec **vi** votre fichier `cigale.txt` de `public` pour vérifier qu'il peut le lire, puis de quitter **vi**.
3. Lui demander d'afficher le contenu de votre répertoire `prive`. Cette fois, il ne doit pas pouvoir le faire.
4. Lui demander d'éditer avec **vi** votre fichier `essai3.txt` de `prive` pour vérifier qu'il ne peut pas le lire (car il ne doit pas pouvoir traverser `prive`). Il peut ensuite quitter **vi**.
5. Lui demander d'afficher le contenu de votre répertoire d'accueil, qui est en principe accessible en lecture pour tout le monde. Il peut ensuite fermer cette connexion SSH. S'il n'était pas accessible en lecture au groupe et aux autres, cela changerait-il quelque chose aux questions précédentes de l'exercice ?

Exercice 6

Illustration avec **echo** *du traitement de la ligne de commandes et de la protection par le backslash et les chaînes. Il s'agit de se familiariser avec le traitement de la ligne de commandes qu'opère bash (découpage en mots, substitutions, protections) avant d'exécuter la commande en lui communiquant les arguments/paramètres résultant de ce traitement. La commande* **echo** *servira à illustrer ces traitements.*

1. Comparer et expliquer le résultat des commandes suivantes (respecter les espaces qui sont représentés ici par le caractère ␣), notamment en déterminant le nombre d'arguments passés à **echo** :
   (a) **`echo␣Je␣fais␣␣␣partie␣␣des␣␣␣␣␣␣arguments.`**
   (b) **`echo␣"Je␣fais␣␣␣partie␣␣des␣␣␣␣␣␣arguments."`**
   (c) **`echo␣Je␣fais"␣␣␣"part'ie␣␣d'e"s␣␣␣␣␣␣ar"guments.`**
2. Se placer dans son répertoire `tpunix`
3. Afficher son contenu (sans détail)
4. Comme les espaces, bash traite spécialement les caractères spéciaux non protégés. Notamment, il remplace les motifs par la liste des fichiers qui correspondent, remplace la séquence **`$PATH`** (substitution de variable) par le contenu de la variable **`PATH`**, et remplace la séquence **`!!`** (substitution de l'historique) par la commande précédente. Plusieurs protections sont possibles mais n'ont pas le même effet. Comparer et expliquer le résultat des commandes suivantes :
   (a) **`echo --- * --- $PATH --- !!`**
   (b) **`echo --- \* --- \$PATH --- \!!`**
   (c) **`echo '--- * --- $PATH --- !!'`**
   (d) **`echo "--- * --- $PATH --- !!"`**
   (e) **`echo "--- * --- \$PATH ---" \!!`**
5. Utiliser les 3 protections dans une seule commande **echo** pour faire afficher le texte suivant :
   `~ C'est * bon * pour tout ? \" \' !!! * ~ $PATH " ' ??`
6. Créer avec **vi** un fichier nommé `*.txt`, y insérer quelques caractères, le sauver puis quitter. Nous le supprimerons plus tard.

Exercice 7

Obtenir des informations sur la nature d'une commande, et consulter sa documentation

1. Exécuter `type echo` pour déterminer la nature (commande interne ou externe) de **echo**
2. Utiliser **help** pour afficher l'aide sur la commande interne **echo**
3. Utiliser à nouveau **type** pour savoir s'il existe aussi la commande externe **echo**
4. Afficher le contenu de la variable `PATH`
5. Afficher le contenu des chemins du `PATH`. Vérifier que **echo** est bien présent dans le chemin indiqué par **type**.
6. Utiliser **man** pour consulter le manuel de la commande externe **echo**. Remarquer la présence de l'option **--help**. Faire défiler les pages jusqu'au bout, puis revenir en arrière jusqu'au début et enfin quitter **man**
7. Exécuter la commande externe **echo** avec son option **--help** pour afficher une aide succincte
8. Obtenir de l'aide sur **umask**
9. Obtenir de l'aide sur **bash**. Faire défiler quelques pages puis quitter.

Exercice 8

Utilisation du caractère spécial tilde *pour les répertoires d'accueil des utilisateurs*

1. Taper la commande `cd ~cpb` qui vous place dans le répertoire de l'utilisateur `cpb`
2. Taper la commande `cd ~/tp/tpunix` qui vous place dans votre répertoire `tpunix`
3. À partir de `tpunix`, faire afficher le contenu du répertoire `public/unix` de l'utilisateur `cpb`
4. Faire afficher le chemin absolu du répertoire d'accueil de `root`

Exercice 9

Utilisation des motifs des noms de fichiers et de **echo** *ou* **ls** *pour faire afficher une liste de fichiers*

1. Se placer dans le répertoire `~cpb/public/unix`
2. Afficher le contenu du répertoire de travail. Il contient de nombreux fichiers.
3. Exécuter la commande : `echo *`
   Elle affiche la liste des entrées (fichiers/répertoires) non cachées du répertoire de travail.
4. Exécuter la commande : `echo d*`
   Remarquer que l'affichage est limité aux entrées qui commencent par **d**
5. Exécuter la commande : `ls d*`
   Remarquer que le résultat est différent : **ls** reçoit la même liste d'arguments (les entrées commençant par **d**) et les affiche si ce sont des fichiers ordinaires mais affiche leur contenu si ce sont des répertoires
6. Exécuter la commande : `ls -d d*`
   On devrait obtenir un résultat similaire à `echo d*`, si ce n'est l'utilisation de couleurs.
7. Comparer le résultat de `echo z*` et de `ls -ld z*`. Comprendre pourquoi **ls** affiche une erreur mais pas **echo**.
8. Utiliser **echo** ou **ls** pour faire afficher la liste des fichiers qui :
   (a) commencent par une voyelle minuscule
   (b) ne commencent pas par une voyelle (minuscule ou majuscule)
   (c) finissent par `.txt` (c'est à dire d'extension `txt`)
   (d) finissent par `e.txt` ou par `s.txt` en n'utilisant qu'un seul motif !
   Aide : devant `.txt` il doit y avoir soit `e` soit `s`.
   (e) contiennent un `a`
9. Que veut dire le motif `*[^a]*` ?
   Après y avoir réfléchi, essayer d'exécuter `echo *[^a]*` pour le vérifier. Remarquer que cela affiche aussi des fichiers dont le nom comporte un `a` ! Comprendre pourquoi.

> ⓘ Pour information (seulement), bash admet l'expression de motifs étendus, permettant par exemple d'exprimer "tous les fichiers dont le nom ne contient pas de `a`". Pour cela, il faut activer l'option *extglob* par la commande `shopt -s extglob`, puis utiliser le motif étendu `!(`*motif*`)` qui est remplacé par la liste triée de tous les fichiers ne correspondant pas au *motif*. Ainsi, `echo !(*a*)` affiche la liste des fichiers dont le nom ne comprend pas de `a`.

10. On peut utiliser un nombre quelconque de motifs, chacun étant remplacé par les fichiers qui correspondent. Exécuter `echo [aeiouy]* *[efi]`. Remarquer que `ici` apparaît deux fois car il correspond aux deux motifs. Remarquer aussi qu'il ne s'agit pas d'un "OU" : si aucun fichier ne correspond à un motif, ce dernier est laissé inchangé (essayer avec `echo [ei]* *[ab]`)

> ⓘ Pour information (seulement), on peut exprimer un "OU" en utilisant un motif étendu tel que `+([aeiouy]*|*[efi])`

11. Exécuter la commande : `echo d* i*`
12. Afficher les informations détaillées sur les mêmes entrées avec **ls**
13. Sans changer de répertoire, faire afficher la liste des fichiers de votre répertoire `tpunix` dont le nom comporte une étoile. Il devrait y en avoir un seul.
14. Supprimer le fichier correspondant en utilisant l'option **-i** de **rm** pour confirmer la suppression et éviter les mauvaises manipulations.
15. Les motifs peuvent être utilisés dans les chemins. Exécuter `ls ~/*/*/*.txt` pour s'en convaincre.

Exercice 10

Utilisation des motifs des noms de fichiers et de **cp** *pour copier un ensemble de fichiers*

1. Se placer dans votre répertoire `tpunix`
2. En restant dans `tpunix`, copier dans votre répertoire de travail, tous les fichiers d'extension `txt` contenus dans le répertoire `public/unix` de l'utilisateur `cpb`.
3. Afficher le contenu de votre répertoire de travail. Il devrait y avoir au moins les fichiers suivants :

```
acrostiche.txt
amphi.txt
amphigouri.txt
cig.txt
```

```
cigale.txt
dates.txt
decale.txt
gouri.txt
simpsons.txt
```

4. Copier tous les fichiers commençant par **c** et d'extension **txt** dans votre répertoire `tp`
5. Supprimer du répertoire `tp`, les fichiers dont le nom comporte un **e** et d'extension **txt**
6. Supprimer les fichiers restant du répertoire `tp` (sans supprimer `tpunix`)

Exercice 11

Synthèse sur les permissions et les manipulations de fichiers

Supposons qu'un utilisateur toto du groupe users tape les commandes suivantes :

```
$ umask
0023
$ ls -al . rep1/fic8
total 12
dr-xrwxr-x 2 titi  users    4096 sep  8 09:09 .
drwxr-x--x 3 titi  titi     4096 sep 10 09:33 ..
drwxr-xrwx 1 titi  users    4096 sep  4 17:30 rep1
--wxr-xrwx 1 truc  chose     362 sep  4 17:31 fic1
-rw-r-x-w- 1 titi  tata      151 sep  4 17:32 fic2
-rwxr----- 1 truc  chose     512 sep  4 17:33 fic3
-r--r--r-- 1 toto  titi     1024 sep  4 17:34 fic4
-rwxrw--w- 1 titi  chose     421 sep  4 17:35 fic5
-rwxrw-r-x 1 titi  titi     1421 sep  4 17:36 fic6
-rw-rw-rw- 1 titi  chose    1421 sep  4 17:37 fic7
-rwxrw-r-x 1 toto  users    8763 sep  4 17:38 rep1/fic8
$ cp -f fic1 fic2
$ mv -f fic3 fic4
$ vi fic5        (tentative (?) de modification de fic5)
$ cp -f fic1 fic6
$ mv -f fic7 rep1/fic8
$ cp -f fic1 fic9
```

Compléter le tableau suivant en indiquant quels sont le propriétaire, le groupe et les permissions (complètes et en utilisant la forme symbolique avec des **r**, **w**, **x** et **-**) des fichiers de la première colonne à l'issu de ces commandes. Si un fichier n'a pas été créé/modifié car une commande a échoué, indiquer la raison de l'échec.

fichier	propriétaire	groupe	permissions
fic2			
fic4			
fic5			
fic6			
rep1/fic8			
fic9			