

# Enoncé du TP 4 Système

C. Pain-Barre

INFO - IUT Aix-en-Provence

version du 12/11/2012

 Démarrer les PC sous Linux.


## 1 Manipulation des processus

### Exercice 1

*Gérer des processus avec **jobs**, **ps**, **kill**, **killall** et les caractères de contrôle.*

1. Depuis une nouvelle fenêtre terminal, ouvrir une connexion SSH sur **allegro** en déportant l’affichage graphique avec l’option **-X** :

```
ssh -X etxxxx@allegro
```

 L’option **-X** est nécessaire pour autoriser l’affichage local (sur le PC) des applications graphiques qui vont être exécutées sur **allegro**.

2. À partir de cette connexion SSH, que nous désignerons **allegro1** dans ce qui suit, nous allons exécuter des applications graphiques terminal pour travailler sur les processus. Sur **allegro1**, taper la commande **aterm** (donc en avant-plan)
3. Sur la fenêtre **aterm** qui s’affiche, taper la commande :

```
PS1='allegro2:\w\$_'
```

qui modifie le prompt de cette fenêtre, qu’on appellera **allegro2**. Déplacer **allegro2** de façon à voir aussi **allegro1** (les redimensionner si nécessaire).

4. Pourquoi le prompt ne s’affiche pas dans **allegro1** ? Essayer d’y taper la commande **ls**. Normalement, elle ne sera pas (encore) exécutée.
5. Fermer **allegro2** en y tapant la commande **exit** ou **CTRL-D**. Cette fois, le prompt s’affiche dans **allegro1** ainsi que le résultat de la commande **ls** tapée précédemment.
6. À partir de **allegro1**, taper à nouveau la commande **aterm** (en avant-plan) pour ouvrir un nouvel **aterm**. Sur cet **aterm**, qu’on appellera encore **allegro2**, taper la commande :

```
PS1='\[e]0;allegro2\a]allegro2:\w\$_'
```

qui en modifie à la fois le titre et le prompt. Déplacer à nouveau **allegro2** de façon à voir **allegro1**.

7. Sur **allegro1**, taper **CTRL-Z**. Le processus **allegro2** est alors suspendu (il aurait été terminé si on avait tapé **CTRL-C**) et le bash d’**allegro1** reprend son exécution en indiquant que la commande **aterm** est suspendue.
8. Essayer de taper la commande **tty** dans **allegro2** qui devrait être figée (ne fait plus rien). D’ailleurs ce qui est tapé n’est pas visible.

9. Sur **allegro1**, taper la commande **jobs**. La liste des processus non terminés et lancés depuis le shell tournant sur **allegro1** est affichée. Cette liste ne devrait contenir que la commande **aterm** (ayant lancé **allegro2**). Chaque ligne commence par le numéro du job.
10. Sur **allegro1**, taper la commande **jobs -l** pour faire apparaître le **PID** (numéro de processus) des jobs en cours sur ce shell.
11. Sur **allegro1**, taper la commande **bg**. Le processus **allegro2** reprend son cours (et exécute la commande **tty** tapée précédemment) mais cette fois en parallèle avec **allegro1**, comme si on l'avait exécutée dès le départ en tapant : **aterm &**
12. Il est maintenant possible d'exécuter des commandes dans **allegro1** et **allegro2**. Essayer avec **ls** sur les deux fenêtres.
13. Sur **allegro1**, lancer une nouvelle fenêtre *aterm* en parallèle (arrière-plan) en tapant la commande :

**aterm&** (ou **aterm &**)

On appellera cette fenêtre **allegro3**. Remarquer que le shell en indique le numéro de job et le **PID**. En modifier éventuellement le titre et le prompt avec :

**PS1='\[\e]0;allegro3\a\]allegro3:\w\\$\_'**

et la déplacer pour voir **allegro1** et **allegro2**.

14. Sur **allegro1**, demander la liste des processus avec la commande **ps**, sans utiliser d'option. Les *aterms* **allegro2** et **allegro3** apparaissent car ils sont rattachés au même terminal que bash, qui les a lancés (la fenêtre **allegro1** elle-même n'y est pas rattachée). Remarquer que les bash s'exécutant dans ces *aterms* n'apparaissent pas car ils sont rattachés à d'autres terminaux.
15. Sur **allegro1**, demander la liste des processus avec l'option **l** (*l* minuscule sans le tiret). Vérifier que ces *aterms* ont bien le bash (ou sh) d'**allegro1** comme père.
16. Recommencer avec l'option **lx** (ce qui, selon la version de **ps** peut donner à peu près le même résultat que l'option **l**, si ce n'est que les processus n'ayant pas de terminal de rattachement sont aussi affichés). Repérer le bash (ou sh) qui a été lancé par **allegro2** et celui lancé par **allegro3**. Chacun est rattaché à un terminal distinct. Utiliser la commande **tty** dans **allegro2** et **allegro3** pour vérifier le terminal utilisé par leur shell.
17. Depuis **allegro2**, utiliser **kill** pour envoyer un signal "gentil" à l'*aterm* **allegro3** (et non pas au bash qui s'y exécute) pour le tuer. Si ça ne suffit pas, utiliser un signal plus "violent".
18. Depuis **allegro1**, exécuter à nouveau un *aterm* en parallèle (arrière-plan).
19. Dans **allegro2**, lancer le programme `~cpb/public/unix/casse-pieds`. Celui-ci ne veut pas s'arrêter de lui-même. Trouver un moyen de le tuer sans utiliser `CTRL-\`, et sans faire disparaître la fenêtre **allegro2** (ni aucune autre fenêtre) ni tuer le shell qui s'y exécute.
20. Depuis **allegro1**, utiliser **killall** pour tuer tous les *aterm* qui vous appartiennent (uniquement ceux-là).

**i** Dans ce qui suit, nous n'aurons plus besoin d'utiliser l'option **-X** de **ssh** pour se connecter à allegro.

## Exercice 2

Utilisation de **pstree** et de **top**.


1. Ouvrir deux fenêtres **gnome-terminal** sur le PC.
2. Sur l'un des deux, taper **man pstree** afin d'étudier le manuel de **pstree**.


3. Agrandir au maximum l'autre fenêtre, et y exécuter **ps** sans option afin d'afficher la hiérarchie des processus. Utiliser l'ascenseur pour remonter au début de la hiérarchie de l'affichage et faire apparaître le processus **init**. Retrouver votre processus **ps** (petit-fils d'un **gnome-terminal**).
4. Dans le man de **ps**, étudier son option **-h**. L'utiliser sur la grande fenêtre et remarquer qu'elle fait ressortir en gras la "branche" à laquelle appartient le processus **ps**.
5. Observer la fin du synopsis de **ps**, puis demander d'afficher seulement la hiérarchie des processus qui vous appartiennent.
6. Exécuter la commande **top**. Vous observez ainsi en temps réel l'occupation de la CPU. La liste des processus affichés est triée par ordre décroissant du pourcentage d'occupation de la CPU. Les critères de tri sont modifiables. **top** est interactive et attend les commandes de l'utilisateur : **h** pour de l'aide ; **q** pour quitter.
7. Si ce n'est pas encore fait, quitter **man** (de **ps**) en tapant **q**, puis fermer la grande fenêtre terminal.

## 2 Utilisateurs et groupes


### Exercice 3

Gestion des groupes par **gpasswd**, **groups**, **newgrp** et **chgrp**


-  Pour pouvoir réaliser correctement cet exercice, il faut travailler en collaborant et en parallèle avec au moins 1 autre étudiant.

 Les groupes ne sont exploitables que sur allegro. La configuration actuelle des PC ne permet pas de bénéficier correctement de leurs fonctionnalités. Mais c'est une notion importante de la sécurité sur Unix.


1. Sur **allegro1** (ou une autre connexion **ssh** sur allegro), taper **id** et observer votre identité, votre groupe primaire et la liste des groupes dont vous êtes membre.
2. Taper **groups** pour n'afficher que la liste de vos groupes.
3. Pager le fichier `/etc/passwd` et rechercher la ligne qui début par votre nom d'utilisateur (vous pouvez utiliser `/chaîne` dans **less** pour rechercher la chaîne). Vérifier que votre *uid* et votre *gid* (groupe primaire) correspondent bien à ce qu'a mentionné **id**.
4. Pager le fichier `/etc/group`. Retrouver votre groupe primaire ainsi que les groupes auxquels vous appartenez (figurez dans la liste des membres) en recherchant votre nom d'utilisateur (dans **less**, on continue une recherche avec **n** (*next*)).

 allegro est configuré pour que vous soyez administrateurs du groupe qui porte votre nom d'utilisateur 😊.

5. Avec **gpasswd**, ajouter 2 ou 3 membres à votre groupe. S'assurer que quelqu'un vous ajoute aussi dans son groupe.


 Les groupes des processus d'un utilisateur sont définis lors de son login (authentification), conformément à `/etc/passwd` et `/etc/group`. La modification de ces fichiers n'impacte pas les groupes des processus actuels ni leurs descendants. Ainsi, l'ajout des membres à votre groupe n'est pas effectif sur leur(s) shell(s) en cours. Il le sera lorsque ces utilisateurs ouvriront une nouvelle connexion SSH sur allegro (ce qu'ils n'ont pas à faire pour le moment).

6. Rechercher votre groupe dans `/etc/group` (utiliser **grep** est une idée) et consulter la liste de ses membres pour vérifier les ajouts. Puis, utiliser **groups** pour vérifier les groupes de ces utilisateurs.
7. Taper **groups** sans argument. Les groupes auxquels on vous a ajouté depuis que cette connexion SSH est établie n'apparaissent pas dans la liste affichée. Sur cette connexion, vous ne faites pas partie de ces groupes !

 Sans argument, **groups** affiche les groupes du processus courant. Avec des arguments, il consulte les fichiers `/etc/passwd` et `/etc/group`, ce qui peut donner un résultat différent pour le même utilisateur.

Taper ensuite **groups** suivi de votre nom d'utilisateur. Observer que la liste affichée est actualisée (mais vous ne faites toujours pas partie de vos nouveaux groupes sur cette connexion).

8. Fermer la connexion SSH et en ouvrir une nouvelle. Utiliser **groups** pour vérifier que vos groupes sont à jour.
9. Utiliser **gpasswd** pour supprimer tous les membres, sauf un, de votre groupe. Les informer oralement de la suppression. S'assurer que quelqu'un vous ait gardé dans son groupe. Vérifier vos suppressions en consultant la ligne de votre groupe dans `/etc/group`.
10. Taper à nouveau **groups** (sans argument). Observer que les groupes desquels on vous a supprimé depuis l'établissement de cette connexion apparaissent toujours. Sur cette connexion, vous en faites toujours partie !
11. Fermer la connexion SSH et en ouvrir une nouvelle. Utiliser **groups** pour vérifier que vos groupes sont à jour.
12. Affecter (temporairement) un mot de passe à votre groupe.


 Si vous le souhaitez, vous serez libre de donner un mot de passe définitif à votre groupe à la fin du TP.

13. Utiliser **newgrp** afin de lancer un shell avec comme groupe primaire (ou par défaut) un groupe auquel on vous a ajouté.
14. Créer<sup>1</sup> un fichier vide appelé `vide.txt` en tapant :  

```
touch vide.txt
```
15. Observer que le groupe de ce fichier est celui choisi en argument de **newgrp**
16. Modifier le groupe de ce fichier en utilisant **chgrp** pour qu'il appartienne à votre propre groupe
17. Vérifier que le changement du groupe a bien eu lieu puis supprimer `vide.txt`.

1. La commande **touch** a pour rôle premier de changer la date de dernière modification d'un fichier et la mettre à la date courante (ce qui peut être pratique plus tard pour les compilations). Un effet de bord de **touch** est que si un fichier en argument n'existe pas, il est créé (vide).

18. Taper **exit** pour quitter le shell qui a été lancé par **newgrp** et revenir au shell de **allegro1** (toujours sur allegro).
19. Utiliser **newgrp** afin de lancer un shell avec comme groupe par défaut un groupe auquel **vous n'appartenez pas** mais qui dispose d'un mot de passe (que l'administrateur du groupe vous aura communiqué).
20. On peut vérifier que **newgrp** a réussi. Taper **groups** qui devrait afficher que (dans ce shell) vous faites partie de ce groupe et il est votre groupe par défaut.
21. Créer un autre fichier vide par **touch vide2.txt**. Regarder le groupe de ce fichier.
22. Supprimer **vide2.txt**.
23. Taper **exit** pour quitter le shell qui a été lancé par **newgrp** et revenir au shell **allegro1** (sur allegro).
24. Créer un répertoire **groupe** dans votre répertoire d'accueil et modifier les permissions de **groupe** pour autoriser l'exécution et la lecture aux membres de votre groupe, mais enlever tous les droits aux autres.
25. Copier le fichier **cigale.txt** dans **groupe**. Vérifier que cette copie de **cigale.txt** est bien de votre groupe et modifier ses permissions pour qu'elle soit accessible en lecture/écriture aux membres du groupe. Comprendre pourquoi les permissions des autres n'ont aucune importance mais enlever quand même tous les droits aux autres.
26. Copier le fichier **simpsons.txt** dans **groupe**. Vérifier qu'il est bien de votre groupe. Modifier ses permissions pour ne laisser que la lecture aux membres du groupe, et aucun droit pour les autres.
27. Copier les fichiers **amphigouri.txt** et **dates.txt** dans **groupe**. Vérifier qu'ils sont bien de votre groupe et modifier leurs permissions pour ne laisser aucun droit aux membres du groupe et aux autres.
28. Vérifier en demandant ensuite à :
  - (a) un non-membre de votre groupe de lire ou de se placer dans votre répertoire **groupe**, ce qu'il ne pourra pas faire car il n'a pas le droit d'exécution sur **groupe**
  - (b) un membre de votre groupe de lire le contenu du répertoire **groupe**. Cela devrait marcher.
  - (c) un membre de votre groupe de lire le fichier **groupe/cigale.txt**. Cela devrait marcher.
  - (d) un membre de votre groupe de modifier le fichier **groupe/cigale.txt**. Cela devrait marcher.

 Remarquer que, bien que modifié par un autre utilisateur, **cigale.txt** reste votre propriété. Modifier un fichier ne veut pas dire se l'approprier. . .

- (e) un membre de votre groupe de modifier le fichier **groupe/simpsons.txt**. Il ne le peut pas, que ce soit par des redirections ou avec **vi**.
  - (f) un membre de votre groupe de lire le contenu de **groupe/amphigouri.txt**, ce qui ne devrait pas marcher, car il ne possède pas les droits de lecture sur **amphigouri.txt**.
29. Ajouter les droits d'écriture aux membres du groupe sur le répertoire **groupe**, puis :
- (a) Demander à un membre du groupe de modifier le fichier **groupe/simpsons.txt** en utilisant les redirections **>** ou **>>**. Par exemple, en exécutant **echo salut > ~vous/groupe/simpsons.txt** ou **echo salut >> ~vous/groupe/simpsons.txt**. Il ne le peut pas car ces redirections nécessitent le droit de modification du fichier cible s'il existe déjà.
  - (b) Demander à un membre du groupe de supprimer le fichier **groupe/dates.txt**. Bien qu'il n'ait aucun droit dessus, il le peut !!!
  - (c) Demander à un membre du groupe de modifier le fichier **groupe/simpsons.txt** en utilisant **vi**. Cette fois, malgré les avertissements de **vi**, il le peut !!! Constaté que le fichier modifié a changé de propriétaire (il a en fait été supprimé puis recréé par **vi**).

- (d) Demander à un membre du groupe de renommer le fichier `groupe/amphigouri.txt` en `groupe/truc.txt`, ce qu'il doit pouvoir faire. Observer que seul le nom du fichier a changé (en fait, c'est le répertoire `groupe` qui a été modifié).
- (e) Plus fort encore, demander à un membre du groupe de modifier le fichier `groupe/truc.txt` avec **vi**. Bien qu'illisible par **vi**, il peut le faire !!! Lors de la sauvegarde, il sera lui aussi supprimé et recréé par **vi**, ce qui explique le changement de propriétaire.
- (f) Demander à un membre du groupe de copier son fichier `dates.txt` dans votre répertoire `groupe` et de placer les permissions de lecture aux membres du groupe mais pas aux autres.
- (g) Essayer de lire ce fichier. Cela ne devrait pas marcher si vous n'appartenez pas à son groupe.
- (h) Demander à son propriétaire de modifier le groupe de `dates.txt` pour qu'il appartienne à un groupe commun avec vous.
- (i) Essayer à nouveau de le lire. Cela devrait être possible.
- (j) Si vous n'avez pas les droits d'écriture sur `dates.txt`, vous ne pouvez pas le modifier par une redirection (**vi**, quant à lui, le fera si vous le forcez, en le supprimant et en le recréant, et il deviendra votre propriété). Essayer de le modifier par une redirection.

30. Supprimer `groupe` et son contenu.

31. Supprimer tous les membres de votre groupe (consulter `/etc/group`).



**À partir de maintenant, vous êtes maître de votre groupe. Vous pouvez y ajouter/supprimer des membres. Mais surtout, évitez de laisser les droits de modification au groupe et aux autres sur vos fichiers et répertoires ! ! !**

## Exercice 4

*Travail sur les utilisateurs et les groupes. Dédurre les accès et droits à des fichiers/répertoires à partir de commandes exécutées.*

Supposons que sur un système, seuls les utilisateurs réels `u1`, `u2` à `u6` existent et que `u1` est administrateur des groupes `g1` à `g5` dont il est pour le moment le seul membre. Les utilisateurs autres que `u1` appartiennent tous uniquement au groupe `users`. `u1` tape les commandes suivantes depuis son répertoire d'accueil :

```
$ id
uid=1000(u1) gid=1000(users) groupes=1000(users),1001(g11),1002(g12),1003(g13),
1004(g14),1005(g15),1006(g16),1007(g17)
$ umask
022
$ ls -al
drwxr-xr-x  31 u1  users      4096 oct  3 18:30 ./
drwxr-xr-x  26 root root      4096 oct  4 19:20 ../
-rwxrw-rwx   1 u1  users      1870 sep 17 2002 f1
-rw-rw-r-x   1 u1  users        528 sep 13 2005 f2
-rwxrwxr--   1 u1  users      1321 oct  6 2010 f3
$ gpasswd -a u2 g1
$ gpasswd -a u3 g1
$ gpasswd -a u4 g1
$ gpasswd -a u5 g1
$ gpasswd -a u2 g2
```

```
$ gpasswd -a u3 g2
$ gpasswd -a u4 g2
$ gpasswd -a u3 g3
$ gpasswd -a u4 g3
$ gpasswd -a u2 g4
$ gpasswd -a u4 g4
$ gpasswd -a u2 g5
$ mkdir r1 r2 r2/r3
$ chmod 705 r1
$ chmod 774 r2
$ chmod 750 r2/r3
$ chgrp g2 r1
$ chgrp g1 r2
$ chgrp g4 r2/r3
$ chgrp g3 f2
$ chgrp g5 f3
$ mv f1 r1
$ mv f2 r2
$ mv f3 r2/r3
```

Toutes ces commandes ont réussi. Nous voulons savoir quelles sont désormais les permissions des utilisateurs sur ces fichiers/répertoires. Pour cela, compléter le tableau suivant sur une feuille en indiquant dans chaque case les permissions de l'utilisateur de la colonne sur le fichier/répertoire de la ligne. Utiliser une forme symbolique (avec **r**, **w**, **x** et **-**). Bien entendu, les permissions d'un utilisateur sur un fichier sont nulles s'ils ne peut y accéder...

	u2	u3	u4	u5	u6
r1					
f1					
r2					
f2					
r3					
f3					

### 3 Expressions régulières et utilitaires

#### Exercice 5

*Recherche des lignes de fichiers contenant certaines chaînes de caractères avec **grep***

1. Sur allegro, copier le fichier `~cpb/public/unix/fruit.price` dans votre répertoire `tpunix`.
2. Visionner votre copie avec **cat**. Elle contient une liste de fruits en anglais, plusieurs blancs et un prix en euros.
3. À l'aide de **grep**, faire afficher les lignes de ce fichier :
  - (a) contenant l'expression **berries** (il y en a 4)
  - (b) contenant l'expression **appLES** en ignorant la casse (distinction minuscule/majuscule) (il y en a 2)

**i** Par la suite, on considérera que tous les fruits sont écrits en minuscules. Si ce n'était pas le cas, on pourrait utiliser l'option **-i**.



- (c) **ne** contenant **pas** l'expression **apples** (il y en a 16)
- (d) contenant le mot **apples** (il y en a 1)
- (e) dont le fruit (et donc la ligne) commence par la lettre **s** (il y en a 1)
- (f) dont le fruit commence par une voyelle minuscule (il y en a 2)
- (g) dont le fruit commence par la lettre **p** ou une lettre comprise entre **a** et **g** (il y en a 10)
- (h) dont le fruit **ne** commence **pas** par une voyelle (il y en a 16)
- (i) dont le fruit **ne** commence **pas** par une lettre comprise entre **a** et **m** (il y en a 8)
- (j) dont le nom comporte au moins 6 lettres (il y en a 15)
- (k) dont le nom comporte exactement 5 lettres, ni plus ni moins ! (il y en a 2)
- (l) dont le fruit commence par un **p** et comporte un maximum de 7 lettres (il y en a 2)
- (m) dont le prix se termine par **79** (il y en a 2)
- (n) dont le prix **ne** se termine **pas** par **9** (il y en a 5)
- (o) dont le prix se termine par **9** mais pas par **39**, **69**, **79**, ni **89** (il y en a 7)
- (p) dont le prix ne se termine pas par **09**, ni par **39**, ni par **79** (il y en a 15)
- (q) dont le prix commence par un point (il y en a 6). On considère que le seul point présent dans chaque ligne marque la décimale du prix.
- (r) dont le prix est inférieur à 1 euro (il y en a 10). Il n'y a pas besoin de comparaisons numériques pour cela...
- (s) dont le prix est inférieur à 1 euro et dont le nom commence par une lettre comprise entre **a** et **g** (il y en a 3)
- (t) dont le prix est supérieur ou égal à 1 euro mais inférieur strictement à 1.50 euros (il y en a 2)
- (u) dont la première lettre du fruit est comprise entre **a** et **e**, ou dont la deuxième lettre est **a** ou **e** (il y en a 10)

## Exercice 6

Utilisation de **sed** (en particulier combinée à d'autres commandes)

### 1. Questions ne nécessitant pas l'utilisation de fichiers script **sed** :

- (a) Faire afficher en majuscules **le contenu** des fichiers d'extension **.txt** de votre répertoire **tpunix**
- (b) Faire afficher en majuscules **la liste** des fichiers d'extension **.txt** de votre répertoire **tpunix** (sans afficher les autres fichiers). Par exemple, en supposant que **ls -l** donne le résultat suivant :

```
$ ls -l
total 24
-rw-r--r--  1 cpb prof 1168 mar 17 08:09 amphigouri.txt
-rw-r--r--  1 cpb prof  677 mar 17 08:09 cigale.txt
-rw-r--r--  1 cpb prof  451 mar 17 08:09 des_lignes.txt
drwxr-xr-x  2 cpb prof 4096 mar 17 08:09 rep1/
drwxr-xr-x  2 cpb prof 4096 mar 17 08:09 rep2/
-rw-r--r--  1 cpb prof   10 mar 17 08:09 unfic
```

il faut afficher :

```
AMPHIGOURI.TXT
CIGALE.TXT
DES_LIGNES.TXT
```



- (c) Faire afficher la liste de **tous** les fichiers de votre répertoire `tpunix` en affichant le nom des fichiers d'extension `.txt` en majuscules (et en laissant les autres noms inchangés). Il faut obtenir par exemple :

```
AMPHIGOURI.TXT
CIGALE.TXT
DES_LIGNES.TXT
rep1/
rep2/
unfic
```

- (d) Faire afficher le contenu de votre fichier `cigale.txt` en ajoutant 5 espaces au début de chaque ligne
- (e) Faire afficher le contenu du fichier `decale.txt` en supprimant les espaces en début de ligne
- (f) Sans utiliser **cut**, faire afficher la liste détaillée des fichiers d'extension `.txt` (uniquement) de votre répertoire `tpunix`, en ne gardant que les permissions (avec le type) et le nom des fichiers, séparés par un espace. Pour simplifier, on suppose que les noms des fichiers ne contiennent pas d'espace. Il faut donc obtenir :

```
-rw-r--r-- amphigouri.txt
-rw-r--r-- cigale.txt
-rw-r--r-- des_lignes.txt
```

- (g) Encore sans utiliser **cut**, faire afficher la liste détaillée **des fichiers ordinaires** de votre répertoire `tpunix`, en ne gardant que les permissions, **sans** le type, et le nom des fichiers séparés par un espace. Au nouveau, on suppose que le nom des fichiers ne contiennent pas d'espace. Il faut obtenir :

```
rw-r--r-- amphigouri.txt
rw-r--r-- cigale.txt
rw-r--r-- des_lignes.txt
rw-r--r-- unfic
```

2. Faire afficher le contenu du fichier `amphigouri.txt` en ajoutant un retour à la ligne à la suite de chaque point (ou suite de points). Bien que ce ne soit pas nécessaire, utiliser un script **sed**.

**i** Selon la version, **sed** permet d'utiliser des séquences telles que `\n` dans la chaîne de remplacement. Si ce n'est pas le cas, il faut saisir un retour à la ligne mais le protéger.

## Exercice 7

### Combinaison de **cut** et **sed**

Nous avons déjà vu que pour **cut**, deux délimiteurs de champ qui se suivent délimitent un champ vide, et aucune de ses options ne permet de remédier à cela. C'est pourquoi, il est parfois intéressant de faire précéder **cut** d'un pré-traitement par **sed** (ou **tr**).

1. Taper la commande **ps lx**
2. On souhaite ne récupérer que les numéros d'utilisateur, le PID et la commande (sans ses éventuelles options ou arguments), en supposant que le nom de la commande ne comprend pas d'espace. Plutôt que de compter les colonnes, combiner **sed** et **cut** afin d'obtenir le résultat souhaité.
3. (Aperçu de **awk**) Le résultat obtenu précédemment devrait être le même qu'avec la commande :

```
ps lx | awk '{ print $2, $3, $13 }'
```