

Enoncé du TP 3 Système

C. Pain-Barre

INFO - IUT Aix-en-Provence

version du 1/11/2012

❗ Démarrer les PC sous Linux.

Les exercices sont à faire sur allegro via une connexion SSH.

Certains exercices utilisent des fichiers créés aux TP précédents. Vous pourrez copier les fichiers qui vous manquent depuis le répertoire `~cpb/public/unix`.

Exercice 1

Utiliser **cat** pour afficher des fichiers, et pour en concaténer.

1. Dans votre répertoire `tpunix`, utiliser **cat** pour visualiser le contenu du fichier `simpsons.txt`
2. Recommencer en faisant aussi afficher les numéros des lignes.
3. Sans changer de répertoire, utiliser **cat** pour visualiser le contenu du fichier `/etc/group` qui recense les groupes d'utilisateurs reconnus par le système. Ce fichier étant grand, une bonne partie de son contenu a défilé et ne figure plus sur la fenêtre (mais peut être visualisé en utilisant l'ascenseur¹ si l'application "terminal" est graphique). Remarquer que chaque ligne comporte des renseignements sur le **groupe** dont le nom figure en début de ligne. Ce fichier sera étudié ultérieurement.
4. Taper `CRTL+ALT+F1` pour basculer sur le terminal texte "à l'ancienne" nommé **tty1** qui devrait afficher l'invite d'authentification `login:`. Après s'être logé avec le compte de l'université, ouvrir une nouvelle connexion SSH sur allegro. Nous travaillerons désormais sur **tty1**.


❗ En général, 6 terminaux texte de ce type sont créés au démarrage et sont accessibles via les combinaisons `CRTL+ALT+F1` à `CRTL+ALT+F6`. La session graphique occupe le 7^e terminal et est accessible via les combinaisons `CRTL+ALT+F7`.

5. Utiliser **cat** pour visualiser le contenu du fichier `/etc/passwd` qui recense les utilisateurs reconnus par le système. Ce fichier est lui aussi trop grand pour être visible en entier sur ce terminal. On n'en voit que la fin. Remarquer que chaque ligne comporte des renseignements sur l'**utilisateur** dont le nom figure en début de ligne. Ce fichier sera aussi étudié ultérieurement.
6. Utiliser **cat** pour afficher le fichier `des_lignes.txt`. Puis, une nouvelle fois pour afficher le fichier `acrostiche.txt`. Enfin, une dernière fois pour afficher la concaténation des fichiers `des_lignes.txt` et `acrostiche.txt`. Remarquer qu'en effet, rien ne distingue la fin de `des_lignes.txt` du début d'`acrostiche.txt`.

1. Il y a une limite du nombre de lignes mémorisées par l'application "terminal" et visibles via l'ascenseur. Elle est souvent modifiable via les préférences/profils de l'application.

Exercice 2

Utiliser **less** pour paginer l'affichage.

 On continue sur **tty1**.

1. Utiliser **less** pour paginer l'affichage du fichier `/etc/passwd`. Seul le début du fichier occupe l'espace disponible sur le terminal et **less** attend vos instructions. Faire défiler l'affichage ligne par ligne pendant quelques lignes (avec `Entrée`) puis page par page (avec `Espace`) jusqu'à retrouver la ligne vous concernant. Y jeter un coup d'œil puis quitter **less** (avec `q`).
2. Utiliser une seule commande **less** pour paginer l'affichage des fichiers `a_decouper.txt` et `~/ .bashrc` (le passage au fichier suivant se fait en tapant `:n`)
3. Fermer cette connexion SSH sur allegro en tapant **exit**
4. Terminer le shell de **tty1** en tapant à nouveau **exit** (l'invite d'authentification "login:" doit s'afficher)
5. Taper `CRTL+ALT+F7` pour revenir à la session graphique. Se replacer sur (sélectionner) la fenêtre terminal de la connexion SSH sur allegro.

Entrées-sorties, terminaux et redirections

Le shell bash actuellement actif sur le terminal est **rattaché** à ce dernier. Notamment, si on ferme la fenêtre terminal, le shell se termine aussi. Le terminal est utilisé pour les entrées-sorties du shell, ainsi que des commandes que le shell exécute. En entrée, le terminal fournit les frappes du clavier. En sortie standard et d'erreur, le terminal écrit les caractères à l'écran. Cette situation est illustrée par la figure 1.

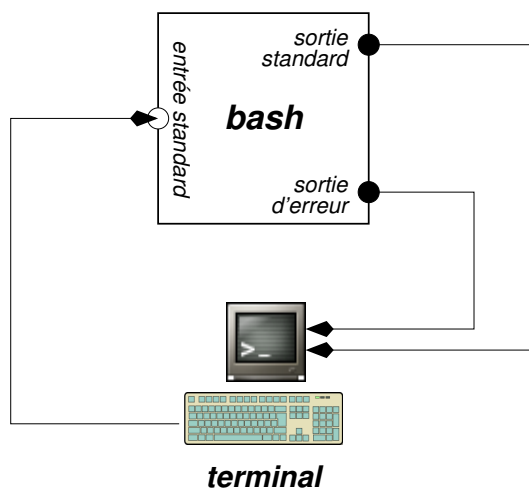


FIGURE 1 – Entrées-sorties du shell sur un terminal

Or, sous Unix tout est fichier ! Y compris les terminaux, qui sont représentés par des fichiers spéciaux de périphériques en mode caractère, placés par le noyau dans l'arborescence de `/dev`, le répertoire (virtuel) des périphériques :

- les **tty1** à **tty7** créés au démarrage correspondent aux fichiers `/dev/tty1` à `/dev/tty7` ;

- les terminaux créés à la demande (applications terminal, connexions SSH, etc.) correspondent aux fichiers de **/dev/pts**. Leur nom est un simple nombre : le premier terminal de ce type créé est **/dev/pts/0** ; le second est **/dev/pts/1**, puis **/dev/pts/2**, etc.

Un utilisateur toto qui dispose du terminal **/dev/pts/0** devrait obtenir des résultats similaires à ceux-ci :

```
$ tty
/dev/pts/0
```

⇒ ce terminal est **/dev/pts/0**

```
$ ls -l /dev/pts/0
crw--w---- 1 toto  tty 136, 0 31 oct.  11:04 /dev/pts/0
```

⇒ c'est un fichier spécial de périphérique en mode caractère.

Le propriétaire du terminal est toto. Il possède les droits :

- de lecture : bash a donc le droit de lire les frappes du clavier ;
- d'écriture : bash a donc le droit d'écrire ses messages (normaux/d'erreur) à l'écran.

Les entrées-sorties du bash de toto sur ce terminal sont illustrées par la figure 2. Les commandes exécutées via ce bash ont par défaut les mêmes entrées-sorties sur ce terminal.

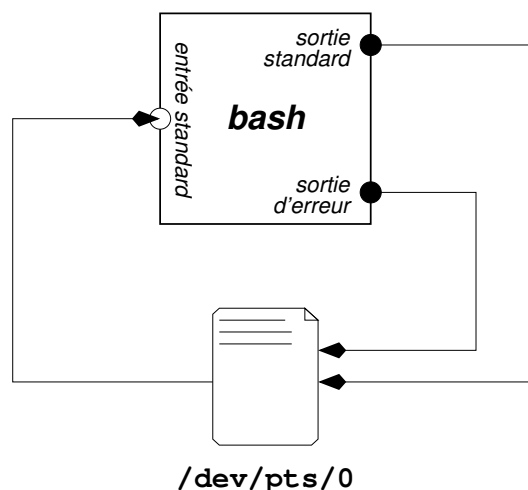


FIGURE 2 – Entrées-sorties du shell d'un utilisateur sur le terminal **/dev/pts/0**

Les redirections et les tubes permettent de les modifier. Dans ce cas, le shell (via le système) réalise la connexion adéquate de l'entrée et/ou sortie(s) avec le fichier ou le tube avant d'exécuter la commande. C'est transparent pour la commande : elle lit via son entrée standard et écrit via ses sorties standard et d'erreur. Selon la redirection, il s'agira d'un terminal ou d'un autre fichier spécial, un fichier ordinaire, un tube, une communication réseau...

Enfin, on voit que le groupe du terminal est **tty**. Ici, le groupe possède le droit d'écriture, ce qui autorise les autres utilisateurs à utiliser **write** ou **wall** pour envoyer un message qui s'affiche sur ce terminal². Le propriétaire d'un terminal peut désactiver cette possibilité en y exécutant **mesg n**. Il l'active avec **mesg y**. Nous y reviendrons plus loin.

2. On étudiera plus tard les mécanismes permettant à **write** et **wall** de fonctionner.

Exercice 3

Étude des entrées-sorties, des terminaux et des redirections.

1. Exécuter **tty** pour connaître le terminal utilisé sur allegro.

i Sur le PC, l'application terminal a créé un terminal (local) qui est relié par la connexion SSH à celui créé sur allegro et indiqué par la commande **tty**.

2. Faire afficher les informations détaillées sur le fichier correspondant.
3. Faire afficher les informations détaillées sur le répertoire `/dev/pts`. Remarquer la présence des terminaux des autres utilisateurs d'allegro, dont le vôtre.

i Le fichier **ptmx** est un cas particulier qu'on ne traitera pas.

4. Utiliser **cat** pour faire afficher le fichier `~cpb/public/unix/sorties.c`.
5. Les programmes peuvent gérer leurs entrées-sorties : ouvrir (ou créer) tout type de fichier en lecture (en entrée), en écriture (en sortie) ou en lecture/écriture (en entrée et en sortie). Ce sont autant d'entrées/sorties qui s'ajoutent à celles par défaut. Ils peuvent aussi fermer des entrées/sorties.


La commande **cat** précédente a reçu en argument le fichier qu'elle a ouvert en lecture. En s'inspirant de la figure 2, dessiner les entrées-sorties de cette commande. Quelles entrées-sorties ont été effectivement utilisées par **cat** au cours de son exécution ?

6. Dessiner les entrées-sorties de la commande :

```
cat < ~cpb/public/unix/sorties.c
```

puis l'exécuter. Remarquer que **cat** n'a pas d'argument. Elle lit son entrée standard, jusqu'à rencontrer la fin de fichier, puis se termine.

7. Exécuter **cat /etc/shadow** pour tenter de visualiser le contenu du fichier `/etc/shadow` qui contient notamment les mots de passe cryptés des utilisateurs reconnus par le système. Cette fois, la tentative de lecture échoue. Notons que **cat** écrit le message d'erreur sur sa sortie d'erreur. On remarque que comme pour de nombreux utilitaires, le message d'erreur est précédé du nom de la commande qui l'écrit. Vérifier vos permissions sur ce fichier.

 On en profite pour rappeler que même sans droit sur un fichier, on peut quand même obtenir les informations détaillées le concernant, tant qu'on peut accéder au répertoire qui le contient. . .

8. Exécuter **cat < /etc/shadow** et remarquer que le message d'erreur provient cette fois de bash qui ne parvient pas à ouvrir le fichier en lecture pour le connecter à l'entrée de **cat**, qui n'a même pas été exécutée !
9. Le fichier `sorties.c` précédent contient le code source d'un programme en langage C qui écrit 2 lignes de texte sur chacune de ses sorties (standard et d'erreur), en les alternant. Sans rentrer dans les détails, il choisit la sortie de ses messages (**stdout** pour les messages normaux et **stderr** pour les messages d'erreur) sans se préoccuper d'où elles mènent en réalité.

i Le fichier `sorties.cxx` contient le code source C++ d'un programme similaire, choisissant le flux de sortie (**cout** ou **cerr**) selon le message.

Le source `sorties.c` a été compilé pour donner l'exécutable `sorties` présent dans le même répertoire. Copier `sorties` dans votre répertoire `tpunix`.

10. Dans votre répertoire `tpunix`, taper `./sorties` pour l'exécuter, ce qui devrait afficher les messages suivants :

```
Cette ligne a été écrite sur la sortie standard
Cette ligne a été écrite sur la sortie d'erreur
Cette ligne a été écrite sur la sortie standard
Cette ligne a été écrite sur la sortie d'erreur
```

où le texte affiché indique la sortie utilisée pour l'écrire.

11. Dessiner les entrées-sorties des commandes suivantes et en déduire leur résultat **avant** de les exécuter pour vérifier. Afficher à la suite de chaque commande le contenu de `fichierXX` :

(a) `./sorties > fichierXX`

(b) `./sorties 2> fichierXX`

(c) `./sorties >& fichierXX` (ou `./sorties > fichierXX 2>&1`)

(d) `./sorties >> fichierXX 2>&1`

12. Afficher les informations détaillées sur le fichier `/dev/null`. Remarquer que c'est un fichier spécial de périphérique en mode caractère, accessible en lecture/écriture pour tout le monde. C'est un terminal fictif représentant le néant : il ne fournit jamais aucune donnée (toujours fin de fichier) quand on l'utilise en lecture, et il ne stocke rien de ce qu'on y écrit (absorbe les données, qui se perdent). Il sert essentiellement dans des redirections pour "éliminer" des entrées-sorties :

(a) Déduire le résultat de la commande `cat < /dev/null` **avant** de l'exécuter pour vérifier.

(b) Déduire le résultat de la commande `./sorties > /dev/null` **avant** de l'exécuter pour vérifier. Afficher ensuite le contenu de `/dev/null`

(c) Déduire le résultat de la commande `./sorties 2> /dev/null` **avant** de l'exécuter pour vérifier.

(d) Exécuter `sorties` en la rendant muette, c'est à dire en faisant disparaître tous ses messages

13. La commande suivante produit-elle une erreur, alors que `fichier-nouveau.txt` n'existe pas ? Pourquoi ?

```
ls -l fichier-nouveau.txt > fichier-nouveau.txt
```

14. Comparer le contenu de `fichier-nouveau.txt` (visualisable avec `cat`) et le résultat de la commande `ls -l fichier-nouveau.txt`. Comprenez-vous mieux pourquoi il n'y a pas eu d'erreur ?

Indication : il faut se demander quel programme a créé le fichier et quand, et quel programme y a écrit des données. La taille du fichier indiquée par `ls` et ce qu'il y a à la place dans le fichier `fichier-nouveau.txt` sont des éléments de réponse.

15. Créer le fichier `liste.txt` contenant la liste des fichiers de `tpunix` (avec informations détaillées).

16. En utilisant `cat`, créer le fichier `essai4.txt` qui contient la concaténation des fichiers `essai1.txt` et `essai2.txt`.

17. Créer avec `cat` un fichier `cat.txt` qui contient votre nom d'utilisateur (que vous taperez au clavier). La fin de fichier est produite au clavier avec la combinaison `CTRL-D`.

18. Taper la commande : `ls ~cpb/*`

Cela affiche des noms de fichiers, le contenu de certains répertoires et des messages d'erreur

19. Reprendre la commande précédente mais éliminer les messages d'erreur. Pour cela, se servir du terminal fictif `/dev/null`.
20. Reprendre la commande précédente pour placer les messages normaux de **ls** dans le fichier `lisibles` tout en éliminant les messages d'erreur. Vérifier en affichant `lisibles` avec **cat**.

Exercice 4

*Premiers pas avec les tubes (et les redirections) : combinaisons de **ls**, **less** et **cat**. Il s'agit d'étudier les entrées-sorties des commandes présentes dans un tube.*

1. Dessiner les entrées-sorties de la chaîne de traitement suivante :

```
./sorties | cat
```

en déduire ce qu'elle fait et l'exécuter plusieurs fois pour vérifier le résultat. Il peut arriver que les messages soient mélangés... Il est clair que l'utilisation de **cat** dans cette commande n'a un intérêt que pédagogique...

2. Dessiner les entrées-sorties de la chaîne de traitement suivante :

```
./sorties 2>&1 | cat
```

en déduire ce qu'elle fait et l'exécuter pour vérifier le résultat.

3. Faire afficher le contenu détaillé du répertoire `/home`. Le résultat est trop long pour tenir sur la fenêtre.
- 4.

i La commande **less** est une commande interactive contrôlée par le clavier. Elle est "intelligente" car elle doit se soucier à la fois de sa sortie (pour paginer correctement) mais aussi de son entrée pour pouvoir être contrôlée. Si elle est exécutée avec des arguments, elle ouvre en lecture les fichiers correspondants et utilise son entrée standard pour recevoir les ordres du clavier. Elle se termine aussitôt si elle est exécutée sans argument, sauf si son entrée standard n'est pas le terminal, auquel cas elle pagine ce qu'elle y lit. Mais, devant être contrôlée par les frappes au clavier, elle ouvre une entrée supplémentaire sur le terminal auquel elle est rattachée pour pouvoir les lire.

Dessiner les entrées-sorties de la chaîne de traitement suivante :

```
ls -l /home | less
```

Elle combine la commande précédente avec **less**, afin de paginer sa sortie. L'exécuter et naviguer dans la pagination de **less** avec `Espace` pour la page suivante, `b` pour la précédente, et `q` pour quitter. Au passage, observer les différents droits positionnés sur les répertoires d'accueil des utilisateurs.

5. Dessiner les entrées-sorties de la chaîne de traitement suivante :

```
./sorties | cat > fictube1
```

en déduire ce qu'elle fait et l'exécuter puis afficher `fictube1` pour vérifier le résultat.

6. Dessiner les entrées-sorties de la chaîne de traitement suivante :

```
./sorties | cat 2> fictube2
```

en déduire ce qu'elle fait et l'exécuter puis afficher `fictube2` pour vérifier le résultat.

7. Dessiner les entrées-sorties de la chaîne de traitement suivante :

```
./sorties 2> /dev/null | cat > fictube3
```

en déduire ce qu'elle fait et l'exécuter puis afficher `fictube3` pour vérifier le résultat.

8. La commande `ls -l /home/*/p*` tente d'afficher les détails sur les fichiers (ou sur le contenu des répertoires) commençant par `p` et présents dans les répertoires d'accueil des utilisateurs d'allegro (hors `root`). Cela comprend les éventuels répertoires `public` et `prive` des utilisateurs. S'assurer de bien avoir compris pourquoi, puis l'exécuter. Elle affiche effectivement des informations détaillées et aussi des messages d'erreur. Là encore, s'assurer de bien avoir compris pourquoi.
9. Dessiner les entrées-sorties de la chaîne de traitement suivante :

```
ls -l /home/*/p* | less
```

puis l'exécuter. Naviguer dans la pagination de `less`. Constater que le résultat est assez chaotique, notamment deux exécutions successives ne s'affichent généralement pas de la même façon. Ceci parce que les messages d'erreur se mélangent à la pagination de `less` (elle utilise des séquences de caractères spéciales permettant d'effacer l'écran et positionner le curseur). D'ailleurs, en naviguant d'une page à l'autre en avant et en arrière, les messages d'erreur finissent par disparaître complètement.

10. Reprendre la commande précédente afin de paginer aussi les messages d'erreur.
11. Reprendre la commande précédente mais cette fois se débarrasser des messages d'erreur (ils ne doivent plus apparaître nulle part), et ne paginer que les messages normaux.
12. Reprendre la commande précédente en ajoutant la commande `cat` dans la chaîne de traitement, tout en obtenant exactement le même résultat. Oui, on vous demande d'utiliser un `cat` ne servant à rien...

Exercice 5

Utiliser `wc` pour réaliser des comptages divers, éventuellement combinée avec d'autres commandes

1. Dans `tpunix`, utiliser `wc` pour faire afficher le nombre de lignes, mots et octets de votre fichier `cigale.txt`
2. Faire afficher uniquement la taille de la plus grande ligne de `cigale.txt`
3. Faire afficher le nombre de lignes de `cigale.txt`, sans son nom. C'est à dire que seul `21` doit être affiché.
Aide : `wc` affiche le nom des fichiers qu'elle reçoit en arguments et qu'elle ouvre en lecture mais si elle n'a pas d'argument, elle traite l'entrée standard et n'affiche que les comptages.
4. Faire afficher le nombre de lignes de chaque fichier d'extension `.txt`, ainsi qu'une totalisation
5. En combinant avec `cat`, faire uniquement afficher le nombre total de lignes des fichiers d'extension `.txt`.
Aide : `cat` doit fournir à `wc` le contenu de l'ensemble des fichiers d'extension `.txt`, afin qu'elle en compte le nombre de lignes.
6. Faire afficher le nombre de terminaux créés à la demande qui sont actuellement ouverts sur allegro.
Aide : on peut légitimement supposer que ces terminaux correspondent aux entrées du répertoire `/dev/pts` dont le nom commence par un chiffre. Aussi, `ls` (sans option `-l`) présente les fichiers en colonne si sa sortie est un terminal, mais les présente ligne par ligne si ce n'est pas le cas.

Exercice 6

Utilisation de `head` et de `tail`, éventuellement combinées

❶ À vous de choisir s'il faut utiliser `head` et/ou `tail`, selon ce qui est demandé.

1. Si ce n'est pas déjà fait, se placer dans votre répertoire `tpunix` et utiliser **cat** pour afficher le fichier `des_lignes.txt`. Remarquer que les lignes sont numérotées.
2. Faire afficher la première ligne de `des_lignes.txt`
3. Utiliser un nombre **négatif** en argument de l'option **-n** pour faire afficher `des_lignes.txt` mais pas ses deux dernières lignes
4. Faire afficher les deux dernières lignes de `des_lignes.txt`
5. Utiliser un nombre **positif** en argument de l'option **-n** pour faire afficher `des_lignes.txt` mais pas ses cinq premières lignes
6. Faire afficher les deux premières lignes de tous vos fichiers d'extension `.txt` (de `tpunix`), sans en-tête
7. Créer un fichier `extremes` (attention, **sans** extension `.txt`), qui contient la première ligne de tous les fichiers d'extension `.txt` et, à la suite, leur dernière ligne, le tout sans en-tête. Pour cela, il faut utiliser 2 commandes.
8. Faire afficher uniquement la ligne **total** produite par **ls -l**

i Cette ligne **total** est affichée si l'option **-l** est utilisée et lorsque **ls** affiche le contenu d'un répertoire en argument (ici, le répertoire de travail est affiché par défaut).

9. Faire afficher uniquement les lignes 5, 6 et 7 de `des_lignes.txt`

Exercice 7

Utiliser **cut** pour extraire des parties de lignes, éventuellement combinée à d'autres commandes

1. Extraction de champs des enregistrements d'un fichier :
 - (a) Afficher le fichier `/etc/passwd`. Chaque ligne est un **enregistrement** d'un utilisateur du système. Les enregistrements comportent sept **champs** séparés par un `:` (deux-points) et suivent le format suivant :


```
nom : motdepasse : uid : gid : infos : repertoire : shell
```

 où chaque champ comporte un nombre indéterminé de caractères (différents de `:`).
 - (b) Ne faire afficher de ce fichier que les champs *nom* et *uid*
 - (c) Modifier la commande précédente pour que le nom et le numéro soient séparés par un espace
2. Extraction de caractères dans les lignes fournies par une commande :

- (a) Afficher les informations détaillées du répertoire de travail (`tpunix`)
- (b) En combinant (au moins) **ls** et **cut**, faire uniquement afficher la taille et le nom des entrées (fichiers/répertoires) de `tpunix`, séparés par un espace, à raison d'une ligne par entrée. Par exemple, à partir de :

```
$ ls -l
total 60
-rw-r--r-- 1 toto toto 460 1 nov. 11:20 acrostiche.txt
-rw-r--r-- 1 toto toto 4840 1 nov. 11:20 a_decouper.txt
-rw-r--r-- 1 toto toto 1168 1 nov. 11:20 amphigouri.txt
-rw-r--r-- 1 toto toto 422 1 nov. 11:20 amphi.txt
-rw-r--r-- 1 toto toto 677 1 nov. 11:20 cigale.txt
-rw-r--r-- 1 toto toto 552 1 nov. 11:20 cig.txt
```


il faut obtenir :

```
$ .....???
 460 acrostiche.txt
4840 a_decouper.txt
1168 amphigouri.txt
 422 amphi.txt
 677 cigale.txt
 552 cig.txt
```

Aide : si la sortie de **ls** commence par une ligne `total` comme ici, il faut s'en débarrasser. L'option **-f** de **cut** ne paraît pas appropriée... Pour repérer le numéro des colonnes des caractères présents à l'écran, vous pouvez utiliser le petit programme `~cpb/public/bin/colonnes` qui affiche une règle de 80 colonnes (modifiable) : la première ligne affichée donne la dizaine ; la seconde affiche les unités.

- (c) L'option **-f** de **cut** est manifestement plus pratique que son option **-c**. On n'a pas pu utiliser **-f** ci-dessus à cause de la présence de blancs successifs et indéterminés dans les lignes de **ls**, empêchent de traiter ces lignes comme des enregistrements (ensemble de champs). Pour contourner ce problème, une solution souvent employée est d'insérer la commande **tr** entre **ls** et **cut** afin que **tr** réduise toutes les suites de blancs en un seul, ce qui permet ensuite à **cut** de traiter les lignes comme des ensembles de champs séparés par un blanc. Ainsi, la commande :

```
ls -ld * | tr -s ' ' | cut -d' ' -f 5,9
```

affiche un résultat similaire à la commande précédente. Nous étudierons une autre solution plus tard avec **sed**.

3. Utilisation conjointe de **head**, **tail** et **cut** :

- (a) Agrandir au maximum la fenêtre de votre terminal
- (b) Afficher le fichier `a_decouper.txt`. Les premières lignes indiquent les numéros de colonnes. Si on l'affiche en utilisant l'option **-n** de **cat**, chaque ligne débute par son numéro.
- (c) Ne faire afficher que l'intérieur du cadre.

4. Traitements divers :

- (a) Combiner les commandes nécessaires pour n'afficher que la taille en Kio de l'arborescence de votre répertoire d'accueil, et rien d'autre (juste ce nombre, sans même la référence du répertoire d'accueil). Attention, cette taille n'est pas indiquée par **ls**.
- (b) Le fichier `~cpb/public/unix/annuaire` contient des noms, prénoms et codes postaux :

```
aidubois;laure;13013
ametto;lucie;13008
atant;charles;13013
...
```

Ne faire afficher que les codes postaux.

- (c) Continuer la commande précédente en ajoutant `| sort | uniq` et observer le résultat : **sort** a trié les lignes (codes postaux) et **uniq** a supprimé les lignes successives identiques.
- (d) Faire afficher le nombre de codes postaux différents présents dans `annuaire`.

Exercice 8

Communication inter-terminaux avec les redirections/tubes. Dans cet exercice, on doit prêter une attention particulière aux entrées-sorties qui vont mélanger plusieurs terminaux.

1. Ouvrir une nouvelle fenêtre **gnome-terminal** à partir du PC. On l'appellera **term2**. La précédente fenêtre terminal (avec la connexion SSH sur allegro) s'appellera **term1**.
2. Depuis **term2**, se connecter à allegro par SSH.
3. Sur **term1** et **term2**, taper **tty** pour connaître le terminal auquel le shell est rattaché.
4. On a vu en début de TP qu'on a les droits de lecture et d'écriture sur nos propres terminaux. Depuis **term2**, utiliser **echo** pour écrire le texte `bonjour` sur **term1**.
Aide : il suffit de rediriger la sortie de **echo** dans le fichier terminal utilisé par **term1**.
5. Depuis **term2**, taper une commande qui fait afficher dans **term1** le contenu du fichier `/etc/passwd`.
6. Depuis **term2**, écrire une commande qui affiche et pagine dans **term1** les informations détaillées sur le contenu du répertoire `/home`. Depuis quel terminal **less** est-elle contrôlée (lit les frappes du clavier) ? Noter que c'est conforme à ce que nous avons déjà observé au sujet de **less** dans un tube et son terminal de rattachement.
7. Communiquer à votre voisin la référence du terminal utilisé dans **term1** et lui demander d'écrire un message (avec **echo**) sur ce terminal. Il ne devrait pas y parvenir car il n'a pas les droits suffisants.
8. Ajouter les droits d'écriture aux autres (uniquement) sur le terminal utilisé dans **term1**. Vérifier les droits. Demander à nouveau à votre voisin d'écrire un message sur ce terminal. Cette fois, il le peut et son texte s'affiche sur votre **term1**.
9. Enlever le droit d'écriture aux autres sur le terminal utilisé dans **term1**. Vérifier les droits.

i Un terminal peut être ouvert en lecture aux autres, leur laissant une possibilité de lire ce qui y est tapé. Mais ce n'est pas une bonne idée. . .

ASCII Art

i Cette partie est un entraînement ludique pour la gestion des entrées-sorties. Elle peut être considérée comme facultative.

L'ASCII-Art est l'art d'utiliser de simples caractères afin de reproduire des images ou différents effets. Le fichier `simpsons.txt` en est un exemple. Plusieurs exécutable génèrent de l'ASCII-Art, notamment **cowsay**, **cowthink**, **figlet**, **showfigfonts** et **boxes** :

- **cowsay** (**cowthink**) fait dire (penser) par une vache (ou d'autres créatures selon les options) le texte en arguments ou lu sur l'entrée standard. Si ce texte tient sur plusieurs lignes, il faut utiliser l'option **-n**.
Exemple : **cowsay 'hello kids'** provoque l'affichage suivant :

3. Faire afficher la liste des entrées de votre répertoire d'accueil, en utilisant la police **mini**, et en encadrant le tout par la boîte **peek** pour donner quelque chose comme :

```

/*      _\|/_
      (o o)
+-----oOO-{}-OOo--+
|
| . _ . _ o _
| | _ | | \ / (/_
| |
|
| . _ | _ | o _
| | _ | | | _ | | ( _
| |
|
| _ | _ . _
| | _ | _ )
| |
+-----*/

```

4. Utiliser **boxes** de façon à encadrer le code source de `~cpb/public/unix/sans-commentaire.cxx` par la boîte **c** (qui a pour effet de commenter le texte intérieur) et produire le fichier `tout-commentaire.cxx` dans votre répertoire `tpunix`.
5. Faire raconter la fable « *la cigale et la fourmi* » (contenu dans `cigale.txt`) par la vache tout en la plaçant dans un champ de printemps (**spring**).
6. Concaténer les textes `racine.txt` et `amphigouri.txt` pour les afficher dans un parchemin (**parchment** ou **scroll**).

❗ À vous d'inventer la suite...