Kit de survie sous Linux

philippe.preux@univ-lille3.fr

4 août 2005

Le but de ce document est d'expliquer en quelques pages l'essentiel de ce qui est à savoir pour pouvoir effectuer les manipulations de base sous Linux.

Comme avec d'autres systèmes commerciaux, de très nombreuses manipulations sont possibles simplement via la souris. Cependant, pour une utilisation plus exigente de Linux (en particulier, celle dont nous aurons besoin en TP), il est indispensable de savoir interagir avec son ordinateur via une fenêtre « terminal » et en y tapant des commandes. La connaissance de quelques principes de base (très peu nombreux) et de quelques commandes (elles-aussi, peu nombreuses, disons 20) permet d'effectuer très rapidement les manipulations de fichiers dont nous aurons besoin en TP. De plus, une fois pris au jeu, de très nombreuses commandes sont à découvrir pour nous simplifier encore plus la vie.

La lecture de cette note ne requiert aucune connaissance *a priori* particulière si ce n'est la manipulation d'un clavier et d'une souris.

Tout ce qui est dans cette note doit être maîtrisé pour aborder les TP l'esprit libre !

À faire

Connectez-vous à votre compte Linux. Ouvrez une fenêtre « terminal ». (Cela se fait généralement en cliquant sur une icone en forme d'écran dans la barre d'outils, en bas de l'écran. Une autre possibilité est d'ouvrir cette fenêtre depuis le menu, sous-menu « système ».)

1 Répertoires et fichiers

1.1 Notions essentielles

On commence par quelques règles toujours vérifiées sous Linux :

Règle

- toutes les données sont stockées dans des fichiers ;
- un fichier est une simple suite de caractères ;
- tout fichier est localisé dans un répertoire ;
- un répertoire contient des fichiers et des répertoires.

Du fait de ces deux derniers points, l'ensemble des répertoires[1] et fichiers[2] possède une structure arborescente. Cet arbre possède une racine, nommée `/`.

Chaque utilisateur possède un répertoire personnel[3]. C'est dans celui-ci qu'il place ses propres fichiers et répertoires.

[1] *directory* en anglais.
[2] *file* en anglais.
[3] *home directory* en anglais.

Règle

- À tout instant, un utilisateur se trouve dans un répertoire, dénommé son répertoire courant ou répertoire de travail[4] ;
- lors de sa connexion, un utilisateur est placé dans son répertoire personnel.

Pour connaître le nom du répertoire courant, on utilise la commande `pwd`.

Règle

Chaque fichier ou répertoire possède un nom. Ce nom est une suite quelconque de caractères.

À faire

Tapez la commande `pwd` et observez le résultat.

Une chaîne de caractères du genre `/home/preux` apparaît. Elle signifie que votre répertoire courant porte ce nom. Ce nom indique que vous êtes situé dans le répertoire `preux` (votre nom d'utilisateur), lui-même situé dans le répertoire `home` (remarquez le `/` entre les deux), lui-même situé dans le répertoire racine, ce qui est indiqué par le fait que la chaîne commence par `/`.

La commande `ls` permet de lister le contenu d'un répertoire.

À faire

Tapez la commande `ls` et observez le résultat.

Ce qui vient de s'afficher est le nom des fichiers et des répertoires situés dans votre répertoire courant.

On peut aussi lister le contenu de n'importe quel répertoire en donnant son nom à la suite de `ls`. Par exemple :

À faire

Tapez `ls /`

Il va alors s'afficher quelque chose qui ressemble à ce qui suit :

```
bin   etc     initrd.img      lost+found  opt   sbin  tmp  vmlinuz
boot  home    initrd.img.old  media       proc  srv   usr  vmlinuz.old
dev   initrd  lib             mnt         root  sys   var
```

C'est le contenu du répertoire racine. `bin`, `dev`, `etc`, `lib`, `lost+found`, `proc`, `tmp`, `usr` et `var` sont des répertoires que l'on trouve sur tous les systèmes Linux. Pour les autres, cela dépend du type de système Linux qui est installé sur votre machine. Donc, si vous obtenez un affichage un peu différent, ce n'est pas grave.

Naturellement, vous pouvez également lister ce qui se situe dans ces répertoires.

À faire

Listez le contenu du répertoire `/usr`

Listez le contenu de l'un des répertoires de `/usr`

Vous pouvez vous déplacer dans les répertoires à l'aide de la commande `cd`. Ainsi, si vous tapez la commande `cd /`, vous vous placez dans le répertoire racine, c'est-à-dire que votre répertoire courant devient le répertoire racine.

À faire

Tapez la commande `cd /`

[4] *working directory* en anglais.

Tapez la commande `pwd`
Tapez la commande `ls`

Que constatez-vous ?

Si vous tapez simplement la commande `cd` sans argument, vous vous replacez automatiquement dans votre répertoire utilisateur.

À faire
Tapez `cd`
Tapez la commande `pwd`

Que constatez-vous ?

À faire
Placez-vous dans le répertoire `/usr/bin`
Tapez la commande `cd ..`
Tapez la commande `pwd`
Tapez la commande `cd ..`
Tapez la commande `pwd`

Que constatez-vous ? Réflêchissez avant de lire la suite.

Oui, vous avez compris que la notation `..` signifie « le répertoire dans lequel se trouve le répertoire courant ».

À faire
Quel est le répertoire qui contient le répertoire racine ?

À faire
Placez-vous dans votre répertoire personnel.

Vous pouvez naturellement y créer des fichiers (et des répertoires, ce que l'on verra un eu plus loin). Il y a des tas de manières de créer un fichier : avec un éditeur de textes (`emacs` est notre préféré), avec Open Office, ...

À faire
Tapez la commande `emacs`

Une fenêtre s'ouvre dans laquelle vous pouvez taper votre texte.

À faire
Tapez une trentaine de lignes dans la fenêtre d'emacs. (Ce n'est pas la peine que ces lignes aient un sens ; ce qui compte, c'est qu'il y en ait une trentaine.)
Pour sauvegarder le fichier, tapez (dans la fenêtre emacs) la commande `^X^S`, le nom du fichier (appelez-le `mon-fichier`) puis retour-chariot.
Pour quitter emacs et fermer la fenêtre, tapez la commande `^X^C`

NB : la notation `^X` signifie ctrl-X. Toutes les commandes usuelles d'emacs sont accessibles soit *via* la touche ctrl, soit *via* la touche d'échappement.

À faire
Dans la fenêtre terminal, tapez la commande `ls`

Normalement, vous devriez voir le nom de votre nouveau fichier qui est listé.
Pour voir le contenu de ce fichier, vous avez plusieurs solutions :
– l'éditer avec emacs comme nous venons de le faire en tapant la commande `emacs mon-fichier` ;

– simplement lister son contenu à l'aide la commande `cat`, en tapant la commande `cat mon-fichier` ;
– en utilisant la commande `cat`, l'ensemble du fichier est listé d'un bloc ; si votre fichier est gros, par exemple plusieurs centaines de lignes, ces centaines de lignes vont défiler sans que vous puissiez les voir. Il y a une commande qui permet de visualiser le fichier page-écran par page-écran : tapez `less mon-fichier` et utilisez les touches de défilement de pages haut et bas, la barre d'espace, le retour-chariot et voyez ce qui se passe. Quand vous avez terminé de visualiser votre fichier, tapez `q` pour quitter `less` ;
– si seulement les 10 premières lignes de votre fichier vous intéresse, tapez la commande `head mon-fichier` ; si c'est seulement les 7 premières lignes, tapez `head -7 mon-fichier` ;
– si ce sont les 10 dernières lignes du fichier (resp. les 7 dernières), remplacez la commande `head` par la commande `tail`.

À faire

Manipulez les commandes `cat`, `less`, `head` et `tail` comme on vient d'en parler.

On peut créer un nouveau répertoire à l'aide de la commande `mkdir` suivie du nom du répertoire. On appelle « sous-répertoire » un répertoire situé dans un répertoire (donc, tous les répertoires sont des sous-répertoires, à l'exception du répertoire racine `/`).

À faire

Dans votre répertoire personnel, créez un répertoire dénommé `mon-rep`

Positionnez-vous dans ce répertoire.

Créez-y un fichier (nommons-le `fichier2`) avec `emacs` et tapez-y quelques lignes puis sauvez-le et quittez `emacs`.

Listez le contenu du répertoire courant.

Remontez dans le répertoire précédent (le répertoire qui contient celui que vous venez de créer, c'est-à-dire, votre répertoire personnel).

Listez le contenu du répertoire `mon-rep`.

La commande `cat` permet en fait de concaténer (mettre bout à bout) plusieurs fichiers. Par exemple, la commande `cat mon-fichier mon-rep/fichier2` concatène les deux fichiers que vous avez créés et les affiche à l'écran, en commençant par `mon-fichier` et en continuant par `mon-rep/fichier2`. On peut en fait mettre autant de fichiers en arguments de `cat` et les fichiers sont toujours traités dans l'ordre.

Le résultat d'une commande qui s'affiche à l'écran peut être aisément mis dans un fichier. Ainsi, le résultat de la concaténation précédente peut être mis dans un fichier que nous nommerons `bout-a-bout` par la commande suivante :

```
cat mon-fichier mon-rep/fichier2 > bout-a-bout
```

Le caractère `>` est un caractère spécial (dit « méta-caractère ») qui indique de placer ce qui aurait du s'afficher à l'écran dans le fichier qui est indiqué à sa suite, donc ici, le fichier `bout-a-bout`.

À faire

Faitez la manipulation précédente.

Vérifiez que le fichier `bout-a-bout` contient bien ce que l'on vient de dire.

Le méta-caractère `>` peut être utilisé avec toutes les commandes qui écrivent quelque chose à l'écran. Par exemple, si vous voulez que le résultat de la commande `ls` soit mis dans un fichier, vous pouvez l'utiliser. Cette opération se nomme une « redirection de la sortie standard » : la sortie standard est l'écran (son petit nom est *stdout*) et celle-ci est redirigée vers un fichier.

À faire

Mettez la liste du contenu du répertoire courant dans un fichier dénommé `contenu-du-rep-courant`.

Regardez le contenu de ce fichier ; comparez avec le résultat de la commande `ls` ; que constatez-vous ?

Ce que vous constatez est un cas plutôt exceptionnel[5] : le résultat de la commande `ls` dont la sortie a été redirigée est différent de celui obtenu sur la sortie standard (l'écran). C'est un cas vraiment exceptionnel et c'est fait exprès car en général, quand on redirige la sortie standard de `ls`, on veut généralement avoir un nom de fichier/répertoire par ligne.

On peut obtenir la même sortie à l'écran en mettant l'argument `-1` à la commande `ls` : ainsi `ls -1` donnera le même résultat à l'écran que quand vous redirigez sa sortie.

Notons enfin que quand on redirige la sortie d'une commande par `> toto`, le contenu du fichier `toto` est écrasé. Si l'on veut que le résultat de la commande se place au bout du fichier, on peut mettre `>> toto`. Dans ce cas, le contenu du fichier n'est pas écrasé et le résultat de l'exécution de l'instruction est placé au bout du fichier.

La commande `wc` compte le nombre de caractères, mots et lignes contenus dans un fichier.

À faire

Tapez la commande `wc mon-fichier`

Interprétez le résultat.

Trouvez un moyen qui vous donne le nombre de fichiers et répertoires présents dans un répertoire donné.

On peut encore utiliser `wc` pour compter le nombre de caractères, mots et lignes que vous tapez au clavier. Pour cela, tapez simplement la commande `wc` sans argument. Tout ce que vous allez maintenant taper est considéré comme le flux de données sur lequel `wc` agît, c'est-à-dire, effectue ce décompte. Pour arrêter ce flux, on tape un caractère spécial qui indique précisèment fin-de-flux (ou fin-de-fichier, encore dénommé *End of File* soit EOF) ; ce caractère est `^D` et il doit se trouver en début de ligne (c'est-à-dire que vous avez tapé retour-chariot juste avant de taper `^D`).

À faire

Tapez la commande `wc`

Tapez quelques lignes terminées par un caractère EOF.

Interprétez le résultat.

Donc, en résumé pour la commande `wc`, on peut soit spécifier le fichier à traiter en argument, soit lire des données au clavier. Le clavier a un caractère dual à l'écran : de la même manière que ce qu'une commande envoie à l'écran peut être redirigé vers un fichier, ce qu'une commande lit au clavier peut être redirigé vers un fichier. Le flux sur lequel une commande lit est dénommé le fichier standard d'entrée (*stdin*). Ce fichier standard d'entrée peut lui-aussi être redirigé à l'aide du méta-caractère `<`. Ainsi, la commande `wc < mon-fichier` fait comme si le contenu du fichier `mon-fichier` était tapé sur le clavier alors que la commande `wc` (sans argument) a été invoquée. Dans le cas de la commande `wc`, faire `wc mon-fichier` ou `wc < mon-fichier` est totalement équivalent.

Si vous avez bien compris ce qui précède, vous devez en avoir déduit aussi que les commandes `cat mon-fichier` et `cat < mon-fichier` auront exactement le même comportement.

Naturellement, on ne peut rediriger l'entrée standard (resp. la sortie standard) que de commandes qui y lisent (resp. écrivent) des données. Ainsi, rediriger l'entrée de `ls` n'a pas de sens puisque `ls` ne lit jamais de données sur son entrée standard.

[5] la règle sous Linux est que le comportement d'une commande soit totalement inchangé quand sa sortie est redirigée, mais aussi son entrée puisque nous verrons un peu plus loin que l'entrée d'une commande peut elle-aussi être redirigée.

On a assez souvent envie que le résultat de l'exécution d'une commande soit utilisé comme l'entrée d'une autre commande ; par exemple, pour compter le nombre de fichiers et de répertoires d'un répertoire, on aimerait que la sortie de `ls` soit envoyée automatiquement vers `wc`. En utilisant un fichier intermédiaire, on peut facilement faire cela (en principe, vous l'avez fait plus haut). Mais, on n'est même pas obligé de passer par un fichier temporaire : on peut taper la commande `ls | wc` qui signifie : rediriger la sortie standard de `ls` vers l'entrée standard de `wc`.

À faire

Tapez la commande `ls | wc`

Interprétez.

Cette opération (connecter la sortie d'une commande à l'entrée d'une autre) se nomme un pipeline. On peut ainsi pipeliner autant de commandes que l'on veut.

À faire

Vous voulez copier un fichier dans un autre (autrement dit, il sera dupliqué). Proposez une manière de le faire à l'aide de la commande `cat`.

1.2 Chemins

Pour l'instant, on a toujours spécifier des répertoires et des fichiers se trouvant dans le répertoire courant (à l'exception de la racine `/`). Sans changer de répertoire, on pourrait vouloir spécifier le fichier `fichier2` situé dans le répertoire `mon-rep` dans votre répertoire personnel. C'est possible en utilisant la notation suivante : `mon-rep/fichier2`. Cette notation spécifie le fichier `fichier2` situé dans le répertoire `mon-rep`, lui-même situé dans le répertoire courant.

Puisque le répertoire courant est votre répertoire personnel (supposons qu'il s'agisse de `/home/preux`), ce fichier est également spécifié par `/home/preux/mon-rep/fichier2`. Cette notation indique la localisation du fichier depuis le répertoire racine. De ce fait, on nomme cela son « chemin absolu » ; un chemin absolu commence toujours par `/`. Un chemin qui n'est pas absolu, est dit « relatif ». Donc, `mon-rep/fichier2` est un chemin relatif.

Pour un fichier ou un répertoire donné, il n'existe qu'un seul chemin absolu, mais il existe des tas de chemins relatifs. Par exemple, rien ne vous empêche de spécifier ce fichier en passant par le répertoire contenant votre répertoire courant, soit `../preux/mon-rep/fichier2`, ou encore par le répertoire le contenant, soit `../../home/preux/mon-rep/fichier2`.

Sachez aussi que le répertoire courant peut toujours être spécifié à l'aide de la notation `.` (le caractère point). Donc, `fichier2` peut être également spécifié par `./mon-rep/fichier2`.

Sachez enfin que le méta-caractère `~` représente votre répertoire personnel. Donc, le chemin `~/mon-rep/fichier2` permet aussi d'accéder à ce fichier.

Ces différentes notations semblent compliquer les choses, mais elles sont très utiles et très faciles à utiliser avec un minimum d'expérience.

Dans toutes les commandes, on peut utiliser des chemins absolus ou relatifs, cela n'a aucune conséquence sur le comportement.

1.3 Quelques commandes presqu'essentielles, en tout cas, très utiles

Quelques commandes très pratiques :

- `cp c1 c2` copie (= duplique) le fichier spécifié par le chemin `c1` dans le fichier spécifié par le chemin `c2`. Si `c2` est un répertoire, la copie de `c1` est placée dans ce répertoire ; sinon, `c1` est copié dans un fichier dont le nom est `c2` ;

- `cp -r c1 c2` copie l'objet spécifié par le chemin `c1` dans l'objet spécifié par le chemin `c2`. Si `c1` est un répertoire, cette commande copie l'ensemble des fichiers et répertoires qui s'y trouvent (récursivement). Sinon, cette commande est équivalente à `cp c1 c2` ;
- `mv c1 c2` déplace le fichier spécifié par le chemin `c1` dans le fichier spécifié par le chemin `c2`. Si `c2` est un répertoire, `c1` est déplacé dans ce répertoire ; sinon, `c1` est déplacé dans un fichier portant le nom `c2` ;
- `rm c` détruit (= efface[6]) le fichier spécifié par le chemin `c`. Ce chemin doit spécifier un fichier, pas un répertoire ;
- `rm -r c` détruit tous les fichiers et répertoires dont la racine est `c`. Si `c` est un fichier, cette commande est équivalente à `rm c` ;
- `rmdir c` détruit le répertoire spécifié par le chemin `c`. Ce chemin doit spécifier un répertoire vide ;
- `ln -s c1 c2` crée un lien pointant sur l'objet (fichier ou répertoire) spécifié par le chemin `c1` qui porte le nom indiqué par le chemin `c2`. Avant l'exécution de cette commande, l'objet `c2` ne doit pas exister. Après son exécution, l'objet `c1` peut être accédé *via* le chemin `c2`. Par lien, nous entendons que `c1` et `c2` sont en fait le même objet et que cet objet peut maintenant être accédé selon deux chemins différents (on peut en ajouter autant que l'on veut) ; si l'on modifie `c1`, on modifie automatiquement `c2` ; `c2` n'est pas vraiment un fichier ou un répertoire, c'est un lien ; si l'on détruit `c1`, `c2` ne pointe plus vers rien ; par contre, si l'on détruit `c2`, `c1` existe encore.

2 D'autres notions et commandes très importantes

2.1 L'aide en ligne

Toutes les commandes Linux sont documentées et cette documentation est accessible en ligne. Un exemple vaut mieux qu'un long discours :

À faire

Tapez la commande `man ls`

Vous vous retrouvez alors dans la page de manuel concernant la commande `ls`. Cette page est affichée en utilisant la commande `less` que nous avons rencontrée plus haut. Vous avez donc eu les informations permettant de naviguer dans cette aide.

2.2 Propriétaires et autorisations

Tout objet (fichier, répertoire, lien) possède un propriétaire et des autorisations. Ces autorisations indiquent si les différents types d'utilisateurs peuvent, par exemple, lire ou écrire cet objet.

Il existe trois types d'utilisateurs : le propriétaire de l'objet, le groupe (d'utilisateurs) auquel appartient l'objet et les autres utilisateurs. Comme son nom l'indique, le groupe est un groupe d'utilisateurs. Souvent (pas tout le temps), le groupe et l'utilisateur sont la même chose : il n'y a qu'un seul utilisateur par groupe et un groupe par utilisateur ; dans ce cas, la notion de groupe est plutôt historique.

À faire

Dans votre répertoire personnel, tapez la commande `ls -l`

L'option `-l` indique d'utiliser un format long. Ce format long indique pour chaque objet contenu dans votre répertoire personnel une ligne de la forme :

[6] attention, ici effacer signifie vraiment effacer, sans aucune possibilité de récupérer le fichier ; ce n'est pas une mise à la corbeille...

```
-rw-r--r--  1 preux preux   3048 2005-07-06 16:51 mon-fichier
```

Celle-ci indique les informations suivantes concernant le fichier dénommé `mon-fichier` :

- `-rw-r--r--` est constitué de 10 caractères :
  - le premier est soit `-` pour un fichier, soit `d` pour un répertoire, soit `l` pour un lien ;
  - ensuite, on a 3 groupes de 3 caractères, un groupe par type d'utilisateur : le premier pour le propriétaire, le deuxième pour le groupe, le troisième pour les autres. Pour chaque groupe, le premier caractère est `r` qui indique que le fichier peut être lu, ou `-` qui indique que le fichier ne peut pas être lu ; le deuxième caractère est `w` qui indique que le fichier peut être écrit, ou `-` qui indique que le fichier ne peut pas être écrit ; le troisième caractère est `x` qui indique que le fichier est un fichier exécutable (= un programme), `-` sinon.

    Dans le cas présent, cette chaîne de 9 caractères indique que le propriétaire peut lire ou écrire le fichier, que les utilisateurs appartenant au groupe possédant ce fichier peuvent le lire et que tous les autres utilisateurs peuvent le lire. Quand un fichier est créé, il possède ces autorisations-là.
- ensuite, on trouve le nom du propriétaire ;
- ensuite, on trouve le nom du groupe ;
- ensuite, on trouve la taille du fichier mesurée en caractères ;
- ensuite, on trouve la date et l'heure de dernière modification du fichier ;
- enfin, on trouve le nom du fichier.

Si c'est un répertoire, on trouve une ligne du genre :

```
drwxr-xr-x  2 preux preux   4096 2005-07-06 17:04 mon-rep
```

Les informations sont à peu près les mêmes : on note :

- le `d` en début de ligne ce qui indique qu'il s'agît d'un répertoire ;
- les `r` indiquent que telle catégorie d'utilisateurs (le propriétaire, le groupe et les autres) peut lister le contenu du répertoire ;
- les `w` indiquent que telle catégorie d'utilisateurs peut modifier le contenu du répertoire, c'est-à-dire, y ajouter ou retirer un objet, voire détruire le répertoire lui-même ;
- les `x` dans les autorisations indiquent ici que telle catégorie d'utilisateurs a le droit de visiter ce répertoire (s'y positionner).

Quand un répertoire est créé, il possède ces autorisations-là.

On peut connaître son nom d'utilisateur par la commande `id -nu` et son nom de groupe par la commande `id -ng`.

Pour modifier les autorisations associées à un objet, on utilise la commande `chmod`. Consultez le manuel pour en savoir plus.

2.3 Spécification de plusieurs chemins : * et ?

Dans un chemin, on peut utiliser les méta-caractères `*` et `?`.

`?` indique n'importe quel caractère. Par exemple, la commande `ls ?` ne liste que les objets dont le nom ne comporte qu'un seul caractère ; `ls ?.c` ne liste que les objets dont le nom comporte un caractère suivi des deux caractères `.c` ; `ls a??b` ne liste que les objets dont le nom est composé de 4 caractères, le premier étant `a` et le quatrième étant `b`.

`*` indique n'importe quelle suite d'autant de caractères que l'on veut (y compris 0 caractère). Par exemple, `ls *.c` liste tous les objets qui se terminent par les deux caractères `.c` ; `ls *a*b*a*?` liste tous

les objets dont le nom contient un `a` suivi d'un `b` suivi d'un `a` suivi d'au moins un caractère (ce « au moins un caractères » est spécifié par la notation `?`).

Naturellement, ces deux méta-caractères peuvent se rencontrer n'importe où dans un chemin. Par exemple, `/?/a*/*.txt` spécifient tous les objets dont le nom se termine par les 4 caractères `.txt` situés dans un répertoire dont le nom commence par un `a`, situé dans un répertoire dont le nom ne comporte qu'un seul caractère, situé dans le répertoire racine.

2.4 Des commandes de recherche

2.4.1 grep

Pour chercher tous les fichiers dans lesquels on trouve une certaine chaîne de caractères, on utilise la commande `grep chaine chemin`.

Par exemple, `grep abc *` affiche toutes les lignes des fichiers contenus dans le répertoire courant qui contiennent la chaîne de 3 caractères `abc`.

Quelques options utiles de `grep` :

- `-c` affiche non pas les lignes qui contiennent la chaîne de caractères, mais leur nombre ;
- `-i` ne tient pas compte de la casse (majuscule ou minuscule) des caractères de la chaîne. Ainsi, `grep -i Ab *` affiche les lignes contenant `ab`, `Ab`, `aB` ou `AB` ;
- `-l` affiche seulement le nom des fichiers contenant la chaîne.

Naturellement, ces options peuvent être combinées. D'autres options existent, voir pour cela la page du manuel. `grep` peut lire le fichier sur l'entrée standard ; donc la redirection de son entrée standard par les méta-caractères `<` et `|` est possible.

Plutôt que de chercher une chaîne de caractères, on peut rechercher un motif, par exemple, les chaînes qui commencent par `e` suivi de 3 caractères quelconques et des deux caractères `to` puis une fin de la ligne. Cela s'exprime par la commande suivante : `grep 'e...to$' *`

Ici, `'e...to$'` indique la chaîne à chercher dans les fichiers. Cette chaîne doit être mise entre `'` comme cela sera expliqué plus loin. Donc, le motif est spécifié par `e...to$` qui se comprend comme suit : une chaîne qui

- commence par `e`
- suivi de n'importe quel caractère (le premier `.`)
- suivi de n'importe quel caractère (le deuxième `.`)
- suivi de n'importe quel caractère (le troisième `.`) — donc, on en est à trois caractères quelconques derrière le `e` —
- suivi des deux caractères `to`
- suivi d'une fin de ligne (spécifiée par `$`).

Cette chaîne de caractères (le motif) est une « expression régulière ». Il faut noter qu'elle utilise des caractères comme `*` et `?` (méta-caractères du *shell*) dans un sens différent du *shell*.

La description complète de cette fonctionnalité nous emmènerait au-delà des objectifs de cette note ; retenez que `grep` est capable de faire cela et que pour en savoir plus, il suffit de lire le manuel.

2.4.2 find

Pour chercher tous les objets ayant un certain nom, on utilise la commande `find`.

Par exemple, vous savez que vous avez un fichier dénommé `fichier2` sur votre compte[7], mais vous

[7] le mot « compte » recouvre l'ensemble des répertoires et fichiers qui se trouvent dans votre répertoire personnel, ainsi que dans ses sous-répertoires.

ne savez plus où. Pour le retrouver automatiquement, la commande `find ~ -name fichier2 -print` va résoudre votre problème. La réponse sera par exemple :

```
/home/preux/mon-rep/fichier2
/home/preux/a/b/c/fichier2
```

(en supposant que vous avez par ailleurs créer un répertoire dénommé `c` dans un répertoire dénommé `b` dans un répertoire dénommé `a` situé dans votre répertoire personnel.)

Expliquons cette commande `find ~ -name toto -print` :

- le premier argument `~` indique le répertoire à partir duquel il faut commencer la recherche. Ce répertoire et tous ses sous-répertoires sont parcourus récursivement ;
- le deuxième argument `-name toto` spécifie le nom des objets que l'on recherche ;
- le troisième argument `-print` indique que l'on veut que `find` affiche tous les objets qui répondent aux directives précédentes, ici `-name toto`. (Remarque : cela paraît pour le moins incongru de devoir spécifier que l'on veut que la commande affiche le résultat de sa recherche ; oui, mais c'est comme cela ; si l'on ne met pas `-print` dans la commande précédente, rien n'est affiché, bien que la recherche soit faite ! C'est un archaïsme et l'une des quelques, et rares, bizarreries de Linux.)

Si vous voulez trouver un fichier dont le nom se termine par `.sxw`, vous tapez la commande `find ~ -name '*.sxw' -print`

Ici, le deuxième argument est donc `-name '*.sxw'` : tous les objets répondant au schéma `*.sxw` vont vérifier les directives, donc leur nom sera affiché. (Il est indispensable de mettre `*.sxw` entre apostrophes.)

D'autres directives peuvent être spécifiées, à la place ou en complément de `-name xxx`. Par exemple :

- `-type d` seuls les répertoires sont pris en compte ;
- `-type f` seuls les fichiers sont pris en compte ;
- `-type l` seuls les liens sont pris en compte ;

La commande `find` est très puissante et possède de nombreux arguments possibles. Consultez le manuel pour en savoir plus.

L'entrée standard de `find` ne peut pas être redirigée (cela n'a pas de sens) ; par contre, sa sortie peut l'être.

3 Utilisation améliorée du *shell*

Remarque : le *shell* est un programme qui saisit vos commandes, les exécute et affiche leur résultat. C'est le programme qui fonctionne dans une fenêtre « terminal » avec laquelle nous interagissons depuis le début de cette note. Actuellement, dans les versions standards de Linux, le *shell* se nomme `bash`.

3.1 Quelques touches utiles dans le *shell*

Quand vous tapez des commandes dans le *shell*, un certain nombre de touches vous simplifient la vie. On vous en présente ici les plus familières ; d'autres possibilités existent qui sont décrites dans le manuel en ligne de `bash`.

Le *shell* conserve un historique des dernières centaines de commandes que vous avez tapées. Ainsi, la touche ↑ remonte dans l'historique et affiche la commande précédent la commande actuellement affichée ; autrement dit, taper 3 fois la touche ↑, et la troisième dernière commande que vous aviez tapée est affichée. Quand la n^{e} commande est affichée, tapez ↓ pour voir la $n-1^{\text{e}}$, ... Si ces explications vous paraissent obscures, essayez, vous comprendrez tout de suite.

Quand une commande est affichée, vous pouvez la modifier en la parcourant avec les touches fléchées ← et →. Vous pouvez aussi revenir au tout début de la ligne de commande en tapant `^A`, ou aller à la fin de la ligne par `^E`. Ces commandes sont les mêmes que dans emacs.

Vous voulez éditer le fichier `bout-a-bout` ; vous allez donc taper la commande `emacs bout-a-bout`. Mais vous pouvez faire beaucoup plus rapide en tapant `emacs b` puis tabulation. S'il n'existe qu'un seul objet dans votre répertoire courant dont le nom commence par la lettre `b` (et c'est le cas si vous avez fait les manipulations précédentes et aucune autre), le fait de taper tabulation va automatiquement compléter ce `b` pour donner la seule possibilité, soit `bout-a-bout`.

Mainteanant, vous voulez éditer `mon-rep/fichier2`. Vous tapez `emacs mo` puis tabulation. Il y a deux objets dont le nom commencent par `mo` : le fichier `mon-fichier` et le répertoire `mon-rep`. Quand vous tapez tabulation, comme il y a plusieurs possibilités qui toutes deux commencent par `mon-`, le *shell* complète tant qu'il peut, soit `mo` est complété en `mon-` et le *shell* attend la suite. Il y a maintenant plusieurs possibilités et le *shell* ne peut pas deviner laquelle vous convient. Si vous tapez deux fois sur la touche tabulation, le *shell* les affiche :

```
mon-fichier  mon-rep/
```

et attend que vous tapiez la suite, ou au moins son début pour que l'appui sur tabulation puisse à nouveau compléter la commande de manière adéquate.

À faire

Manipulez ces touches qui permettent de remonter dans l'historique et la touche tabulation.

3.2 L'expansion des méta-caractères par le *shell*

Quand un `*` ou un `?` apparaît dans un chemin, le *shell* l'expanse, c'est-à-dire, le remplace par tout ce qui correspond à ce caractère, cela de manière totalement transparente pour vous, utilisateur du *shell*.

Par exemple, tapez la commande `ls *`.

Le *shell* remplace l'`*` par la liste de tous les objets qui se trouvent dans le répertoire courant. Ainsi, ici, après les manipulations que l'on a faites jusqu'alors, le *shell* va transformer ce `ls *` en `ls bout-a-bout mon-fichier mon-rep`. Ensuite, le *shell* va essayer de lancer la commande `ls` avec ces 3 paramètres.

De même, `ls mo*` va être expansé en `ls mon-fichier mon-rep`.

Le principe est exactement le même pour le méta-caractère `?`.

Ce qui vient d'être dit nous permet d'expliquer la mise entre `'` dans la commande `find ~ -name '*.sxw' -print` rencontrée plus haut.

Supposons que l'on tape la commande `find ~ -name *.sxw -print` (sans les `'` donc). Le *shell* ne se soucie pas de la commande qui a été tapée : il expanse les méta-caractères. Donc, il va expanser ce `*.sxw` qui, ici, ne donnera rien puisqu'aucun fichier ne répond à ce schéma dans le répertoire courant. Donc, le *shell* transforme cette commande en `find ~ -name -print` qui ne signifie pas du tout ce que l'on veut. Pour bien faire, il faut dire au *shell* : ici, n'expanse pas l'`*`.

Les `'` ont précisément cette fonction : ce qui se situe entre les deux `'` n'est pas expansé mais transmis tel quel à la commande. Donc, quand vous tapez la commande `find ~ -name '*.sxw' -print` la commande qui est invoquée par le *shell* est `find ~ -name *.sxw -print`.

Remarque : on utilise souvent des `'` dans les commandes `find` et `grep` quand on spécifie un schéma utilisant un méta-caractère du *shell*.

De la même manière, le méta-caractère `~` est expansé par votre répertoire personnel. Ainsi, quand on tape la commande `ls ~`, le *shell* l'expanse en `ls /home/preux`.

Il n'est pas question ici de détailler tous les méta-caractères du *shell*. Mais il est bon d'en connaître la liste car si vous utilisez l'un de ces caractères dans une commande sans y prendre garde, votre commande sera probablement comprise par le *shell* différemment de ce que vous croyez.

Les méta-caractères du `bash` sont : `* ? ' " ‘ ( ) [ ] { } < > | ; ! ~ &`
espace, tabulation, retour-chariot.

3.3 Variables d'environnement ; variable `PATH`

Quand vous tapez une commande, on vient de voir qu'il y a expansion des méta-caractères par le *shell*. L'étape suivante consiste à invoquer la commande. Cette commande est spécifiée par sa première chaîne de caractères (par habitude, dans la très grande généralité des cas, c'est une suite de lettres minuscules, mais cela pourrait être une suite de caractères quelconques). Cette commande spécifie en fait le nom d'un fichier que vous avez le droit d'exécuter[8]. Aussi, pour que l'invocation d'une commande se passe bien, il faut remplir deux conditions :

1. que le fichier correspondant à la commande soit trouvé ;
2. que ce fichier soit exécutable par vous.

Le second point ne dépend pas de vous : en tant qu'utilisateur normal, certaines commandes sensibles vous sont interdites, toutes les autres vous sont autorisées.

Par contre, concernant le premier point, le *shell* cherche le fichier exécutable portant le nom de la commande que vous avez tapée dans un répertoire qui est listé dans ce que l'on nomme une « variable », dénommée `PATH`.

À faire
Tapez la commande `echo $PATH` et observez le résultat.

Sur ma machine, cette commande affiche la chaîne de caractères `/home/preux/usr/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11` :c'est une suite de répertoires séparés par `:`.

Quand je tape une commande, le *shell* cherche un fichier exécutable de nom approprié tout d'abord dans le répertoire `/home/preux/usr/bin` ; si le fichier y est trouvé et s'il est exécutable, il est appelé avec les paramètres que vous avez tapés ; si le fichier est trouvé et non exécutable, un message d'erreur est affiché du genre `Permission non accordée`. Si le fichier n'est pas trouvé, la même procédure est effectuée avec le répertoire suivant dans la liste, ici `/usr/local/bin`, ...

Remarquons aussi que le nom de la commande peut également spécifier son chemin. Ainsi, on pourrait taper une commande du genre `~/mon-rep/ma-cmd a b c` en supposant que `ma-cmd` est un fichier exécutable. Dans ce cas, la variable `PATH` n'est pas utilisée puisque l'on a indiqué où se trouve le fichier correspondant à la commande.

Détail : la commande `echo` affiche une chaîne de caractères. Par exemple, `echo bonjour` affiche `bonjour`. Si le paramètre commence par un caractère `$`, c'est une variable et `echo` affiche sa valeur.

Il existe d'autres variables (dites variables d'environnement). Vous pouvez les visualiser en tapant la commande `printenv`. Vous pourrez ainsi repérer les variables suivantes ;
– `USER` qui est votre nom d'utilisateur ;

[8] on laisse ici de côté les commandes internes du *shell* (par exemple `cd`). Dans leur cas, il n'y a pas de fichier exécutable associé ; elles sont à l'intérieur du *shell*.

– `HOME` qui est votre répertoire personnel ;
– `PWD` qui est votre répertoire courant (la commande `pwd` n'est en fait qu'un `echo $PWD`).

On peut aisément créer de nouvelles variables si cela est nécessaire. Par exemple, en tapant `MAVARIABLE=toto`, je crée une variable qui se nomme `MAVARIABLE` et dont la valeur est la chaîne de caractères `toto`.

Je peux visualiser sa valeur par la commande `echo`.

Je peux modifier sa valeur en tapant par exemple `MAVARIABLE=titi`. Je peux aussi écrire : `MAVARIABLE=toto$MAVARIABLE`.

À faire

Trouver un moyen d'ajouter des caractères au bout de `MAVARIABLE`. (Pensez à utiliser des méta-caractères.)

3.4 Scripts

On peut placer des commandes dans un fichier pour qu'elles soient ensuite exécutées. Cela nous permet de créer nos propres commandes. Ce type de fichier qui contient des commandes *shell* se nomme un script.

Supposons que l'on veuille disposer d'une commande qui liste le contenu de notre répertoire personnel ainsi que du répertoire racine (exemple qui n'a vraisemblablement aucun intérêt pratique, c'est pour faire simple). Créons un fichier (avec emacs) contenant les 2 lignes suivantes :

```
ls ~
ls /
```

et sauvons-le dans un fichier dénommé `mon-premier-script`.

On peut exécuter ce script en tapant la commande `bash mon-premier-script`.

À faire

Faites les manipulations qui viennent d'être décrites.

On peut également rendre ce fichier directement exécutable en en modifiant ses autorisations. La commande `chmod u+x mon-premier-script` fait cela : elle ajoute (le `+`) l'autorisation d'exécution (le `x`) à moi (le `u` pour utilisateur, sous-entendu, celui qui est propriétaire de ce fichier).

Ensuite, on peut taper la commande `mon-premier-script`... sauf que, si vous avez bien suivi le paragraphe précédent, si le répertoire courant n'est pas listé dans la variable `PATH`, vous obtenez un message d'erreur `command not found`. Donc, soit vous modifiez la valeur de la variale `PATH` pour que cela marche, soit vous tapez `./mon-premier-script`.

À faire

Faites les manipulations qui viennent d'être décrites.

Quand vous saurez que le *shell* cache un langage de programmation complet (documenté dans la documentation en-ligne, soit `man bash`), vous comprendrez que vous pouvez écrire TOUS les programmes que vous pouvez imaginer directement dans ce langage. Cette idée est loin de n'être qu'une idée théorique : Linux dispose déjà d'énormément de commandes de base ; il est très courant que la composition de quelques commandes de base (par pipeline ou dans un script) suffise à effectuer le traitement que vous voulez s'il s'agît de manipulations de fichiers et d'édition automatique de fichiers.

Enfin, notez l'existence de 2 scripts particulier qui sont situés dans votre répertoire personnel :
– `.login` automatiquement appelé à chaque fois que vous vous connecté ;

– `.bashrc` automatiquement appelé à chaque fois que vous lancez un *shell* `bash`.

Vous pouvez modifier ces deux scripts afin que certains traitements soient effectués automatiquement.

3.5 Processus

Un processus est un programme en cours d'exécution. Un programme est un fichier exécutable.

On peut obtenir la liste des processus que l'on a lancé par la commande `ps`.

Pour lancer l'exécution d'un programme depuis un terminal tout en pouvant continuer de taper des commandes dans ce terminal, on met le méta-caractère `&` à la fin de la commande. Par exemple, quand on lance `emacs`, on a souvent envie de pouvoir continuer à taper des commandes pendant que la fenêtre `emacs` est ouverte. Pour cela, on tapera une commande du genre `emacs fichier-édité &`.

3.6 Des commandes de manipulations des fichiers

3.6.1 Extraire une colonne d'un fichier de données

Au cours des TPs, on va manipuler des fichiers de données qui ont la forme suivante : chaque ligne contient une donnée et chaque donnée est composée de plusieurs valeurs disposées selon les colonnes. Par exemple, on aura un fichier dénommé `data` contenant 6 données, chacune décrite par 4 valeurs numériques et une chaîne de caractères :

```
abc     5.1     3.5     1.4     1.3
xyz     4.9     3.0     1.4     0.9
defghi  4.7     3.2     1.3     2.7
#45     4.6     3.1     1.5     1.2
abc     15.1    3.5     1.4     1.3
12      5.0     3.6     1.4     0.4
```

Dans ce type de fichiers, on sépare les colonnes par un caractère séparateur. Selon le cas (selon le logiciel), ce séparateur peut être un espace, une virgule, une tabulation (c'est le cas ici), ...

Une opération classique sur ce genre de fichiers est d'en extraire une colonne, par exemple, la deuxième. Cela se fait très bien par la commande : `cut -f 2 < data`

Cette commande entraîne l'affichage suivant :

```
5.1
4.9
4.7
4.6
15.1
5.0
```

L'argument `-f 2` indique que l'on veut extraire la deuxième colonne.

À faire

Faites les manipulations précédentes. Que se passe-t-il si vous indiquez une colonne qui n'existe pas (un nombre > 5 ici).

Par défaut, `cut` suppose que les données sont séparées par une tabulation. Si elles sont séparées par un autre caractère, une virgule par exemple, on l'indique à `cut` de la manière suivante :

`cut -d, -f 2 < data,` où `data,` contient les mêmes données que le fichier `data` séparées par une `,` (et rien d'autre !).

À faire

Créez ce fichier `data,` où les données sont séparées par une `,` et manipulez la commande `cut`. Que se passe-t-il si vous tapez la commande `cut -f 2 < data,` ?

`cut` permet de faire très facilement ces manipulations qui autrement ne sont pas si simples à faire avec un éditeur de textes.

`cut` dispose d'autres fonctionnalités que vous découvrirez en lisant la page du manuel. Par exemple, au lieu de considérer que le fichier est structuré en champs comme ici, on peut considérer que chaque ligne est une simple suite de caractères et vouloir en extraire la 5^{e} colonne (de caractères). Dans ce cas, on utilisera une option `-c 5` au lieu de `-f 5`.

3.6.2 Ajouter une colonne à un fichier de données

L'opération inverse de celle effectuée par `cut` consiste à ajouter une colonne à un fichier de données. Elle se fait avec la commande `paste`.

Supposons que nous ayons le fichier suivant que nous nommerons `mes-animaux` :

```
cobaye
chouette
bufle
girafe
bronto
diplo
```

On veut obtenir un fichier en « collant » (comme du papier peint[9]) cette colonne devant celles du fichier `data`. La commande `paste mes-animaux data` fait exactement cela, et l'envoi sur le fichier standard de sortie.

À faire

Faites cette manipulation. Collez `mes-animaux` au début puis à la fin de `data`.

Collez également `mes-animaux` à la fin de `data,`. Que constatez-vous ? Êtes-vous content du résultat ?

De même que pour `cut`, on peut indiquer à `paste` le séparateur à utiliser à l'aide de l'option `-d,` dans le cas d'un séparateur `,` (c'est exactement comme pour `cut`).

À faire

Refaites la dernière manipulation de manière à avoir le résultat souhaité (tous les champs sont séparés par une `,`).

Exercice 1

Mettez les colonnes 2, 4 et 5 du fichier `data,` dans un fichier dénommé `manipulation-de-cut`. (Ce que l'on a dit suffit à effectuer cette manipulation. Une autre manière de faire est possible en utilisant uniquement `cut` : lire le manuel pour cela.)

[9] en anglais, *paste* signifie coller du papier peint.

3.6.3 Trier un fichier de données

Une dernière manipulation courante que nous allons détailler consiste à trier les lignes d'un fichier. Par exemple, partant du fichier `data` plus haut, on souhaite obtenir un tri des lignes. La commande `sort < data` envoie sur la sortie standard :

```
12      5.0     3.6     1.4     0.4
#45     4.6     3.1     1.5     1.2
abc     15.1    3.5     1.4     1.3
abc     5.1     3.5     1.4     1.3
defghi  4.7     3.2     1.3     2.7
xyz     4.9     3.0     1.4     0.9
```

qui est le résultat du tri des lignes en les parcourant de gauche à droite, caractère par caractère (et non pas champ par champ), selon l'ordre des caractères dit ASCII[10].

On voit que le résultat est cohérent avec ce que l'on attend quand le premier champ est une chaîne de lettres.

Le fichier étant structuré en colonnes (champs), on peut indiquer que le tri doit se faire sur la n^{e} colonne. Par exemple, un tri sur la deuxième colonne s'obtient par : `sort -k 2 < data`, ce qui donne :

```
abc     15.1    3.5     1.4     1.3
#45     4.6     3.1     1.5     1.2
defghi  4.7     3.2     1.3     2.7
xyz     4.9     3.0     1.4     0.9
12      5.0     3.6     1.4     0.4
abc     5.1     3.5     1.4     1.3
```

(Remarque : la commande précédente n'a pas tenu compte de la 1re colonne, c'est voulu et c'est ce qu'on lui a dit en mettant l'option `-k 2`. Si on veut trier en utilisant le 3^{e} champ, on mettra `-k 3`, ...)

Je sens le lecteur perplexe... 15.1 n'est pas inférieur à 4.6 et pourtant, on trouve 15.1 avant 4.6 (pour les lignes suivantes, ça a l'air d'aller mieux...). Il n'y a pourtant pas lieu d'être perplexe ; on l'a dit plus haut, `sort` fait un tri en considérant des caractères. Donc, quand `sort` trouve `15.1`, c'est pour lui une chaîne de caractères qui commence par `1` suivi de `5` suivi de `.` ... En aucun cas, il ne s'agît du nombre 15.1. Normalement, la perplexité du lecteur vient de disparaître (sinon, relire attentivement tout ce qui précède).

On a maintenant compris le résultat mais, on n'en est pas content pour autant : on voudrait que `sort` considère 15.1 comme un nombre, pas comme une chaîne de caractères. Naturellement, c'est possible et il suffit de l'indiquer à `sort` par l'option `-n`. Ainsi, `sort -n -k 2 < data` fournit le résultat :

```
#45     4.6     3.1     1.5     1.2
defghi  4.7     3.2     1.3     2.7
xyz     4.9     3.0     1.4     0.9
12      5.0     3.6     1.4     0.4
abc     5.1     3.5     1.4     1.3
abc     15.1    3.5     1.4     1.3
```

[10] *in fine*, tout ce qu'utilise un ordinateur est codé sous une forme de nombres en base 2 (binaire). Ainsi, tous les caractères sont codés sous forme d'un nombre (le nombre 48 pour le caractère 0 – le caractère que vous obtenez quand vous appuyez sur la touche 0 de votre clavier –, le nombre 49 pour le caractère 1, ..., 57 pour le 9, 65 pour le A, 66 pour le B, ..., 97 pour le a, 98 pour le b, ..., 32 pour la touche espace, 9 pour la tabulation, ...) Sachant cela, vous pouvez comprendre le résultat de l'exemple d'utilisation de la commande `sort` donné ici.

qui est bien le résultat attendu : le fichier a été trié selon la deuxième colonne.

À faire

Faites ces manipulations avec la commande `sort`.

Si les champs sont séparés par autre chose qu'une tabulation, une `,` par exemple, on utilisera l'option `-t ,`.

À faire

Faites les mêmes manipulations sur `data,`.

3.6.4 Les archives

Une archive est un fichier qui contient des fichiers, répertoires, ... objets en tous genres. Il est très courant d'en manipuler lorsque l'on veut installer un logiciel qui est fourni sous forme d'une archive. Par convention, une archive est stockée dans un fichier dont le nom se termine par `.tar` ou `.tar.gz` ou `.tgz`. Dans les deux derniers cas, l'archive est compressée afin d'en réduire la taille.

Les manipulations typiques sont : désarchiver le contenu d'une archive (sortir les objets de l'archive) et visualiser son contenu.

Pour désarchiver une archive non comprimée, on utilise la commande `tar xf nom-de-l-archive`. La commande `tar` avec laquelle sont effectuées toutes les manipulations sur les archives, comprimées ou pas, est assez vieillote et ses options sont spécifiées d'une manière assez différente des autres commandes Linux que nous avons rencontrées jusqu'à maintenant. Ici, l'option `xf` signifie : désarchiver (`x`) l'archive contenue dans le fichier (`f`) dont le nom suit.

Pour visualiser le contenu d'une archive non comprimée, on utilise la commande `tar tf nom-de-l-archive` le `t` indiquant ici que l'on veut simplement visualiser le contenu.

Pour une archive comprimée, ce sont les mêmes commandes sauf que l'on ajoute un `z` aux options : `tar zxf nom-de-l-archive-comprimée` et `tar ztf nom-de-l-archive-comprimée` respectivement.

L'opération inverse, création d'une archive, est réalisée par la commande : `tar cf mon-archive liste-des-chemins-des-objets-à-y-placer`.