Les commandes Unix

(système de fichiers, processus, commandes)

– Travaux Dirigés (**1**) –

Résumé

Le but de ce TD est avant tout de vous familiariser avec la manipulation de machines Unix et de quelques outils. Ainsi, il s'agit surtout de vous montrer où et comment trouver de l'information, et de connaître les commandes de bases et leur utilisation.

Table des matières

Avant propos

Le *shell*, que nous allons explorer ici, est un environnement complet qui permet de composer des commandes ou des programmes entre eux et d'en recueillir le résultat sous la forme et à l'endroit voulu. Si cet environnement s'appelle le shell ('*coquille*' ou '*coquillage*' en anglais), c'est avant tout car il crée une 'coquille' autour du système d'exploitation en s'interfaçant avec l'utilisateur afin de nous permettre d'interagir plus facilement avec la machine.

Notons que certains shells sont graphiques et nécessitent de passer plus souvent par le biais de la souris. Pour ce qui nous concerne ici, nous nous focaliseront seulement sur les shells dit '*en ligne de commande*'. C'est à dire qu'ils nécessitent d'utiliser le clavier comme interface principale.

De plus, il est important de savoir que le système Unix a été pensé autour de deux abstractions majeures : les **fichiers**[1] et les **processus**[2]. Et, de fait, vous ne manipulerez à travers le shell que des fichiers (majoritairement) et des processus (plus rarement).

1. Certains préfèrent parler de '*stream of bytes*' (flux d'octets) à la place de 'fichiers' afin d'être plus génériques.
2. Les processus sont des programmes en cours d'exécution qui résident en mémoire.

Pour terminer, il est nécessaire de préciser qu'il existe un grand nombre de programmes de shells (`bash`, `csh`, `tcsh`, `kshell`, `zsh`, ...). Chacun introduit de petites variations dans sa manière de fonctionnner ou d'être configuré. Néanmoins, le `bash` (*Bourne Again Shell*) est le shell le plus fréquemment rencontré des nos jours et nous allons nous concentrer sur ce shell en particulier. Mais, rappelez-vous que certaines commandes que nous allons voir par la suite ne marchent qu'en `bash` et pas forcément avec les autres programmes de shell.

1 Trouver de l'aide sur Unix

Le système Unix contient ses propres manuels utilisateurs et deux commandes permettent principalement d'y accéder : `man` et `info`.

1.1 Le manuel utilisateur sous Unix

Chaque commande Unix a normalement une page de manuel associée, souvent traduite en français. On consulte les pages de manuel à travers la commande `man`, simplement en faisant : `man` *commande*. Consultez la table 1 pour en connaître les principales commandes.

Fonction	Commande
Consulter une page dans une section (1-9) précise	`man` *section command*
Recherche par mot clef	`man -k` *keyword*
Rechercher toutes les pages ayant un nom donné	`whatis` *command*
Faire défiler en ligne à ligne	<Down>/<Up>
Faire défiler en page à page	<PageDown>/<PageUp>
Aller au début ou à la fin	<Home>/<End>
Faire une recherche en avant	</> *word*
Faire une recherche en arrière	<?> *word*
Aller à l'occurrence suivante	<n>
Quitter	<q>

Table 1 – Commandes liées aux pages de `man`.

Questions

1. Ouvrez un terminal, et faites '`man man`'.
2. Faite défiler le contenu du manuel en ligne à ligne, en page par page, allez au début et à la fin et, finalement quittez.
3. Cherchez le mot '`man`' (en avant, en arrière) dans le manuel. Faites défiler quelques occurrences.
4. Cherchez toutes les pages de manuel dont le titre ou la description contient le mot `process`.
5. Cherchez toutes les pages de manuels ayant pour nom `open`. Et, consulter le manuel de la section des *programmeurs*.

2 Le système de fichiers

Le système de fichiers Unix est composé de différents types d'objets. Principalement, nous n'en verrons que trois [3] : les *répertoires*, les *fichiers* et les *liens symboliques*.

Un fichier (*file*) est un espace de stockage pour des données. Un répertoire (*directory*) est un élément de hiérarchisation et de classement qui permet de regrouper plusieurs fichiers ou répertoires. Enfin, un lien symbolique (*symbolic link*) est un point d'accès vers un fichier ou un répertoire qui se trouve ailleurs dans le système de fichier. La table 2 présente quelques commandes utiles pour explorer votre système de fichiers.

3. Les autres types d'objets étant : les '*named pipes*', les périphériques ('*block*' ou '*character*') et les '*sockets*'.

Fonction	Commande
Liste les fichiers dans le répertoire courant (*list*)	`ls`
Donne le type d'un fichier et son contenu	`file` *filename*
Affiche le contenu d'un fichier	`less`
Efface un fichier (*remove*)	`rm`
Efface un répertoire s'il est vide (*remove directory*)	`rmdir`

Table 2 – Commandes liées au système de fichiers.

Questions

1. Affichez la liste des fichiers présent à votre racine (faites '`cd`' pour vous rendre à votre racine). Puis, affichez les *fichiers cachés*[4]. Et, enfin, affichez le descriptif *long* (voir ci-dessous) des tous les fichiers (y compris les fichiers cachés). Par exemple :

```
-rw------- 1 user user 8.0K Aug 28 23:59 .bash_history
-rw-r--r-- 1 user user  220 Oct 16  2011 .bash_logout
-rw-r--r-- 1 user user 2.0K Aug 21 12:24 .bashrc
drwxr-xr-x 7 user user 4.0K Aug  5 22:37 Desktop/
-rwx------ 1 user user 6.0K Aug 28 19:09 myprogram*
-rw------- 1 user user    3 Aug 28 21:32 mylink -> myprogram
```

   Grâce à l'option '`-F`' de la commande '`ls`', les répertoires sont suivis de '`/`', les fichiers exécutables sont suivis de '`*`' et les liens de '`@`' (ou indiquent vers quoi ils pointent dans la version longue).
2. Faites : '`file *`' puis '`file .*`'. Voyez à quoi sert chacun des fichiers que vous avez à votre racine. Éventuellement, regardez-en le contenu avec la commande '`less`'.
3. Faites un peu de ménage avec la commande '`rm`' si nécessaire.

2.1 Se déplacer dans le système de fichiers

Un *chemin* (*path*) est la succession de répertoires qu'il faut traverser jusqu'à l'objet que l'on veut atteindre. Ce chemin peut être *absolu* s'il part de la *racine* du système de fichier, ou encore *relatif* s'il part de l'endroit d'où l'on effectue la commande.

Fonction	Commande
Donne le chemin du répertoire courant (*print working directory*)	`pwd`
Change de répertoire (*change directory*)	`cd` *path*
Revient dans le répertoire où l'on était précédemment	`cd -`
Répertoire courant	'`.`'
Répertoire parent	'`..`'
Nom de la racine du système (*root*)	'`/`'
Raccourci vers le répertoire `HOME` de l'utilisateur	'`~`'

Table 3 – Commandes liées aux déplacements dans le système de fichiers.

Par exemple, considérons le système de fichiers (fictif) représenté à droite. Sachant que les nœuds qui se terminent par un '`/`' sont des répertoires et tous les autres sont des fichiers. Sachant aussi que le système comporte un seul utilisateur (`user`) et son `HOME` est '`/home/user`'. Et que seule la hiérarchie de fichiers à l'intérieur de son répertoire `HOME` nous intéresse (nous avons ignoré le reste). Répondez aux questions suivantes.

```
/
\-- home/
    \-- user/
        |-- foo
        |-- bar/
        |   \-- quux
        \-- baz/
```

Questions

1. Donnez le chemin absolu vers le fichier `quux`.

4. Sous Unix, les fichiers cachés sont toujours préfixés par un point ('`.`'). Ils n'apparaissent pas par défaut.

2. Donnez le chemin relatif du fichier `foo` depuis le répertoire `/home/user/baz/`.
3. Comment se rendre à son `HOME` ?
4. En supposant que l'on parte de `/home/user/`. Donnez les commandes qui permettent d'aller à la racine du système, puis de revenir.
5. Où êtes vous si vous faites : '`cd ~/..`' ?
6. Sur votre machine essayez au moins une fois la commande '`pwd`'. Cette commande est vraiment très pratique pour savoir où l'on est et se retrouver dans le système de fichiers.

Attention: Lorsque vous êtes en train de taper un chemin, appuyer deux fois successivement sur la touche <`Tab`> activera la complétion automatique. Cette astuce, si elle est bien utilisée, peut vous permettre de multiplier facilement votre vitesse de frappe par deux ou trois. N'hésitez donc pas à utiliser ce genre de fonctionnalité.

2.2 Les '*wildcards*' du shell

Les *wildcards* sont un moyen pour référencer un ensemble de fichiers ou de répertoires en donnant un *motif* (*pattern*) ponctué par des caractères spéciaux, les *wildcards*, qui groupent un ensemble de caractères à eux seuls. La table 4 liste les *wildcards* qui existent en shell. Pour la série de questions qui suit, vous pouvez créer des fichiers (avec la commande `touch`) ayant les mêmes noms que les mots à reconnaître et faire : `ls` <*pattern*>. Cela vous permettra de tester vos intuitions et valider votre réponse.

Attention: Les motifs du shell contenant des *wildcards* peuvent parfois être confondus avec les expressions régulières POSIX (*regex*). Ces dernières sont utilisées à travers des commandes bien spécifiques (`grep`, `sed`, `awk`, ...) ou bien à l'intérieur de certains langages de programmation (Perl, Python, Java, ...). Quoiqu'il en soit, même si la syntaxe du shell peut y ressembler, ne confondez pas les deux.

Fonction	Motif
Représente le caractère '`a`'	`a`
Représente n'importe quel caractère	`?`
Représente n'importe quelle chaîne de caractères (y compris la chaîne vide)	`*`
Représente le caractère '`a`' ou le caractère '`b`'	`[ab]`
Un caractère parmi tous les caractères entre '`n`' et '`z`'	`[n-z]`
Un caractère parmi tous les caractères sauf ceux entre '`n`' et '`z`'	`[^n-z]`
Représente un caractère décimal	`[0-9]`
Représente tout sauf un caractère décimal	`[^0-9]`
Représente un caractère alphabétique (majuscule ou minuscule)	`[A-Za-z]`
Échappe le caractère '`a`' (utilisé pour les caractères spéciaux)	`\a`

Table 4 – Les différents wildcards du shell.

Questions

1. Quels sont les mots reconnus par '`ab[ab]*a`' ?
   ☐ `abababa` ☐ `aaba` ☐ `aabbaa` ☐ `aba` ☐ `aabababa`
2. Quels sont les mots reconnus par '`a?c`' ?
   ☐ `abc` ☐ `ac` ☐ `abbb` ☐ `bbc`
3. Quels sont les mots reconnus par '`a?[bc]*`' ?
   ☐ `abc` ☐ `abbbbbbbb` ☐ `azc` ☐ `abcbcbcbc` ☐ `ac` ☐ `asccbbbbcbcccc`

4. Quels sont les mots reconnus par '`[a-c]*`' ?
   ☐ `abc` ☐ `xyz` ☐ `azc` ☐ `abcxyz`
5. Quels sont les mots reconnus par '`[a-z]*[\.\?\!]`' ?
   ☐ `battle!` ☐ `Hot` ☐ `green` ☐ `_swamping.` ☐ `jump up.` ☐ `undulate?` ☐ `is.?`
6. Quels sont les mots reconnus par '`[a-zA-Z]*[^,h]=`' ?
   ☐ `Buffer=` ☐ `BotHEr,=` ☐ `Ample` ☐ `FIdDlE7h=` ☐ `Brittle =` ☐ `Other.=`
7. Écrire un motif qui reconnaisse : '`pit`', '`respite`', '`spate`', '`spot`'. Mais pas : '`pt`', '`Pot`', '`peat`', '`part`'.
8. Écrire un motif qui reconnaisse : '`tapeth`', '`apth`', '`wrap-try`', '`sap tray`', '`87ap9th`', '`apothec`'. Mais pas : '`aleht`', '`tarpth`', '`Apt`', '`peth`', '`tarreth`', '`ddapdg`', '`shape the`'.
9. Écrire un motif qui reconnaisse : '`affgfking`', '`baffgkit`', '`rafgkahe`', '`bafghk`', '`baffg kit`'. Mais pas : '`fgok`', '`a fgk`', '`affgm`', '`afffhk`', '`fgok`', '`afg.K`', '`aff gm`'.
10. Écrire un motif qui reconnaisse les mots bien parenthésés comme : '`(((())))`' ou encore '`(()(()))`'. Mais pas les mots mal parenthésés comme : '`(()))`' ou '`((()(()))`'.

2.3 Les expressions régulières POSIX

Comme nous l'avons déjà dit, il existe une deuxième catégorie d'expressions capables de coder des motifs de reconnaissance autre que les simples *wildcards* du shell. Il s'agit des *expression régulières*[5] qui sont plus simples à utiliser et sensiblement plus puissants que les *wildcards*. Nous parlerons ici des expressions régulières qui suivent la norme POSIX qui sont les plus répandus dans le monde Unix. Vous pouvez obtenir plus d'informations sur les expressions régulières en consultant cette page de manuel : '`man 7 regex`'. La table 5 résume aussi la syntaxe de ces expressions régulières. Enfin, la commande `grep -E` utilise les expressions régulières pour coder les motifs de recherche qu'elle prend en argument. Pour faire les exercices qui suivent, vous pouvez remplir un fichier avec les termes proposés et faire un `grep -E` dessus.

Fonction	Expression
Représente le caractère '`a`'	`a`
Représente n'importe quel caractère (*wildcard*)	`.`
Représente le début de la ligne	`^`
Représente la fin de la ligne	`$`
Échappe le caractère '`a`' (utilisé pour les caractères spéciaux)	`\a`
Répète l'expression précédente zéro ou une fois	`?`
Répète l'expression précédente zéro fois ou plus	`*`
Répète l'expression précédente une fois ou plus	`+`
Répète l'expression précédente exactement `n` fois	`{n}`
Répète l'expression précédente entre `n` et `m` fois	`{n,m}`
Répète l'expression précédente au moins `n` fois	`{n,}`
Répète l'expression précédente au plus `m` fois	`{,m}`
Représente le caractère '`a`' ou le caractère '`b`'	`[ab]`
Un caractère parmi tous les caractères entre '`n`' et '`z`'	`[n-z]`
Un caractère parmi tous les caractères sauf ceux entre '`n`' et '`z`'	`[^n-z]`
Représente un caractère décimal	`[[:digit:]]`
Représente un caractère alpha-décimal	`[[:alnum:]]`
Représente un caractère alphabétique (majuscule/minuscule)	`[[:alpha:]]`
Représente un caractère espace ou tabulation	`[[:blank:]]`
Groupe un ensemble d'expressions	`(expr1expr2)`
Ou logique sur les expressions `expr1` et `expr2`	`expr1|expr2`

Table 5 – Les expressions régulières POSIX.

5. Dans les années 1940, Warren McCulloch et Walter Pitts ont modélisés les neurones d'un système nerveux sous la forme d'automates finis. Mais c'est en 1956 que Steve Kleene introduisit le concept mathématiques d'*ensembles réguliers* qui décrivait parfaitement le modèle de McCulloch et Pitts.

Questions

1. Écrivez une expression régulière pour capturer :
   1. Tous les nombres binaires, mais en évitant la chaîne vide.
   2. Tous les nombres binaires qui commencent et finissent par un '1'.
   3. Tous les nombres binaires qui finissent par '00'.
   4. Tous les nombres binaires qui contiennent au moins trois '1'.
   5. Tous les nombres binaires qui ne contiennent pas '110'.
   6. Tous les nombres binaires qui sont des palindromes.
2. Écrivez une expression régulière pour capturer des adresses e-mails comme, par exemple : `ford_prefect3@theguide.com`, `arthur.dent@42.org`, `zaphod-beeblebrox@zaphod.me`, ...
3. Quel est le plus petit mot (non-vide) sur l'alphabet {a,b} qui n'est pas reconnu par '`a*(ab)*b*`'.
4. Toujours sur l'alphabet {a,b}, si l'on considère les expressions régulières suivantes :
   — `r1 = (a*|b*)`
   — `r2 = (ab*|ba*|b*a|(a*b)*)`

   Trouvez un mot de plus petite taille possible qui est reconnu par `r2` mais pas par `r1`. Puis, un mot qui est reconnu par les deux expressions.
5. Trouvez une expression régulière qui reconnaît l'ensemble des mots alphabétiques de longueur paire.
6. Trouvez une expression régulière qui accepte : '`ahkeiki5Ph`', '`Eiyaicoh7o`', '`Eisai7ohMo`', '`us5eeMo5oh`', '`shibeith7I`', '`OhK4quaesh`', '`Iena8uchoo`', '`PhiZie7chi`'.
   Et qui refuse : '`ahkaaseeY1`', '`yien8liZai`', '`Ahkahz4soa`', '`baFee4Joon`', '`OFu0boS0Oh`', '`Foob2oe5uo`', '`iepooV0ohL`', '`setheiieCe`'.
7. Trouvez une expression régulière qui accepte : '`Hp3UuCzFL`', '`;^R*B#0Rs~`', '`i6!Kcc[/<Q`', '`h$QLn2tEm^`', '`-]w4F13i1w`', '`7BEf>c3C#d`'.
   Et qui refuse : '`i65(YMmYhQ`', '`Y({j9m]q:W`', '`/((0ncW#;_`', '`7B:VVuH9|9`', '`n+F"Y^1p%G`'.
8. En utilisant `grep` sur le fichier `/etc/group`, trouvez toutes les lignes qui commencent par '`daemon`'. Puis, affichez celles qui ne commencent pas par '`a`', '`b`', '`c`' ou '`d`'.
9. Trouvez récursivement, sur votre compte utilisateur, toutes les occurences de votre nom de login (en *ignorant la casse*[6]) qui sont dans vos propres fichiers.
10. Regardez ce que fait l'option '`grep --color`' lors d'une recherche (essayez de rajouter cette option en re-résolvant les exercices précédents). Elle peut être utile pour débuguer vos expressions régulières.
11. Allez sur le site `https://regexcrossword.com/` et faites au moins le tutorial.

2.4 Explorer, chercher et modifier le système de fichiers

Savoir explorer, chercher et modifier votre système de fichiers requiert finalement assez peu de commandes (voir table 6). Mais certaines peuvent s'avérer assez compliquées à maîtriser, notamment des commandes comme `grep` et `find`. Nous allons essayer d'explorer leur utilisation.

Fonction	Commande
Crée un fichier vide s'il n'existe pas, et modifie le `atime` sinon	`touch`
Copie des fichiers ou des répertoires (*copy*)	`cp`
Déplace ou renomme un fichier ou un répertoire (*move*)	`mv`
Crée des liens vers des fichiers ou des répertoires (*link*)	`ln -s`
Crée une archive (*tape archive*)	`tar`
Recherche des fichiers ou des répertoires	`find`
Recherche des motifs dans un ou un ensemble de fichiers	`grep`

Table 6 – Commandes de recherche et de modification sur les fichiers et les répertoires.

6. Le terme '*ignorer la casse*' signifie que l'on veut capturer le mot même s'il a des des majuscules ou des minuscules indiféremment.

Questions

1. Commencez par créer la hiérarchie de fichiers suivante :

```
foo/
 |-- bar
 |-- quux/
 |   \-- baz
 \-- rny/
     |-- heak
     \-- gpq
```

2. Avec la commande `cp`, faites une copie récursive du répertoire `foo/rny/` vers `foo/ohka/`.
3. Avec la commande `mv`, renommez et déplacez le fichier `foo/rny/heak` en `foo/ohka/zab`.
4. Avec la commande `ln`, créez un lien symbolique `foo/quux/zanb` qui pointe vers `foo/rny/gpq`.
5. Avec la commande `tar`, créez une archive `foo.tar.bz2` (compressée en bzip2) de tout le répertoire `foo/`, puis listez en le contenu et enfin, extrayez la dans `/tmp`.
6. Avec la commande `rm`, effacez le répertoire `quux/` et tout ce qu'il contient récursivement. Puis restaurez le en décompressant l'archive que vous avez faites à la question précédente.
7. Avec la commande `find`, trouvez tous les liens symboliques de votre répertoire utilisateur.
8. Avec la commande `find`, trouvez tous les fichiers dans `foo/` qui ont un '`a`' dans leur nom.
9. Créez au moins un répertoire vide, puis avec la commande `find`, dénombrez tous les répertoires vides qui sont sur votre compte utilisateur.
10. Créez un répertoire `tmp/`. Puis, avec la commande `find`, copiez tous les fichiers de `foo/` qui commencent par '`b`' dans `tmp/`.

2.5 Les droits sur les fichiers

À chacun de ces trois types d'objets sont associés des *droits* qui régissent l'utilisation que l'on peut faire du fichier, du répertoire ou du lien symbolique. Principalement, il existe trois types de droits : la lecture ('`r`'*ead*), l'écriture ('`w`'*rite*) et l'exécution (*e*'`x`'*ecute*). Notez que le droit en exécution pour un répertoire signifie en fait que l'on a le droit d'entrer dedans (ce qui correspond à l'exécution du répertoire).

Ces droits s'appliquent à trois types d'acteurs : le *propriétaire* de l'objet en question, les *groupes* du propriétaire de l'objet en question et, enfin, l'ensemble de *tous les utilisateurs*.

Ainsi, on symbolise l'ensemble des droits d'un objet par une suite de 9 lettres, par exemple '`rw-r--r--`' signifie que le propriétaire de l'objet en question a le droit de lire et écrire sur l'objet ('`rw-`'), que les groupes dont fait partie le propriétaire ont seulement le droit de lire l'objet ('`r--`') et il en va de même pour l'ensemble des utilisateurs.

Autre exemple, '`rwxr-xr-`' signifie que le propriétaire de l'objet a le droit de lecture/écrire/exécuter, que les groupes auquel il appartient ont le droit de lire et d'exécuter et, enfin, tous les autres utilisateurs ont seulement le droit de le lire.

Il peut aussi exister des droits '*spéciaux*' comme le bit `SETUID` qui permet à un utilisateur d'utiliser un programme qui s'exécutera momentanément avec les droits du propriétaire de l'exécutable. Ceci est notamment utile lorsqu'on ne veut pas donner trop de droits aux utilisateurs, tout en leur laissant accès à certaines fonctionnalités bien précises (le `shutdown` par exemple). Pour finir, les trois commandes utilisées pour changer les droits, le propriétaire ou le groupe d'un objet sont listées dans la table 7.

Fonction	Commande
Donne toutes vos informations utilisateur	`id`
Donne les groupes auxquels vous appartenez	`groups`
Change les droits d'un objet (*change mode*)	`chmod`
Change le propriétaire d'un object (*change owner*)	`chown`
Change le groupe d'un objet (*change group*)	`chgrp`

Table 7 – Commandes liées aux droits sur le système de fichiers.

Questions

1. Créez un fichier `file.txt` et donnez lui successivement les permissions suivantes avec la commande '`chmod`' (vérifiez avec '`ls -l`') :

   a. `rwxrwxrwx`
   b. `rwxrwxr-x`
   c. `rwxr-xr-x`
   d. `r-x------`
   e. `r--r-----`
   f. `rw-r--r--`
   g. `r--r--r--`
   h. `rw-rw-rw-`
   i. `rwx------`

2. Faites '`echo 'hello!' > file.txt`'[7]. Pour tester si vous pouvez lire le fichier, faites simplement '`cat file.txt`'. Interdisez le fichier en écriture et vérifiez ce qu'il se passe lors d'une écriture, puis interdisez le en lecture et vérifiez ce qu'il se passe lors d'une lecture du fichier. Enfin, restaurez vos droits en lecture et écriture.
3. Créez une profondeur de trois répertoires '`level0/level1/level2/`'. Protégez le répertoire `level0` en écriture, et le répertoire `level1` en écriture et en lecture.

   Vérifiez que l'on ne peut plus créer de fichiers dans `level0`, ni lire ce que contient `level0/level1`. Par contre, vous pouvez remarquer que l'on peut toujours accéder à `level2` en faisant directement : `ls -l level0/level1/level2`.

   Comment empêcher totalement l'accès à `level2/` ?
4. Rendez un fichier exécutable et positionnez son bit `SETUID` à *vrai*.
5. Utilisez `umask` pour positionner vos droits par défauts, successivement à : '`rwxrwxrwx`', '`rw-r-r-`' et enfin '`rwx------`'.

2.6 Écrire ou éditer un texte (Emacs)

Nous allons à présent voir un éditeur de texte typique du monde Unix : '`emacs`'. Emacs est un éditeur de texte un peu particulier car il est extrêmement personnalisable pour ceux qui savent programmer en *elisp* (Emacs Lisp). On peut virtuellement tout faire à l'intérieur d'Emacs et y inventer ses propres modes, ses propres fonctions ou configurer des modes existants pour les adapter à ce que l'on veut faire.

L'un des autres avantages d'Emacs est qu'il affiche tous les caractères sans les interpréter et que vous avez ainsi une vision complète du fichier que vous éditez. Il s'agit d'une grosse différence par rapport aux éditeur dits 'WYSIWYG' (*What You See Is What You Get*), car le but d'Emacs n'est pas la composition de texte mais bien la programmation.

Nom et fonction	Touche clavier
Touche *Meta* (<`M->`)	<`Alt`>
Touche *Control* (<`C->`)	<`Ctrl`>
Touche *Escape* (<`Esc->`)	<`Esc`>

Table 8 – Les touches spéciales d'Emacs

De ce fait, Emacs comporte, de base, un certain nombre de fonctionnalités visant à faciliter la vie du programmeur. Ainsi, il atteint le niveau d'un IDE[8] avec ses nombreux plugins'. Mais, si la plupart du temps, Emacs devinera ce que vous désirez faire et proposera le mode d'édition correct pour faire ce que vous voulez, certaines fois, il faudra juste lui donner un coup de pouce pour l'aider ou connaître la bonne fonction pour réaliser ce que vous voudrez. Et le but de cette section est justement de vous familiariser avec cet outil. Commencez par lire attentivement les tables 8 et 9, puis faites les exercices qui suivent.

Questions

1. Partionnez votre éditeur en trois parties, en faisant : '`C-x 3`', '`C-x o`', '`C-x 2`', puis finalement, faites '`C-x 1`'. Trouvez l'utilité de chacune de ces commandes.
2. Ouvrez un fichier `Emacs.txt` et copiez les premiers paragraphes de la page Wikipédia d'Emacs.
3. Reformatez les différents paragraphes avec la commande `M-q`[9].

7. Cette commande va inscrire '`hello!`' dans le fichier `file.txt` et écraser tout ce qu'il y avait précédemment.
8. *Integrated Development Environment* : Il s'agit de logiciels habituellement utilisés pour programmer.
9. La touche `M-` (Méta) correspond des nos jours à la touche <`Alt`>, mais il fut un temps où la touche Méta existait bel et bien à la place de Alt.

Fonction	Commande
Quitte Emacs	C-x C-c
Affiche la FAQ d'Emacs	C-h C-f
Chargement d'un fichier (load-file)	C-x C-f
Sauvegarde du fichier courant (save-file)	C-x C-s
Annuler le dernier changement (undo)	C-_
Rechercher-remplacer (query-replace)	M-%
Recherche en avant (search-forward)	C-s
Recherche en arrière (search-backward)	C-r
Aller au début du fichier	<Esc>-<
Aller à la fin du fichier	<Esc>->
Réindente le paragraphe	M-q
Rappel de la dernière commande	M-x M-p
Retour au mode normal (en cas de problème)	C-g
Donne le mode utilisé et toutes ses fonctions	C-h m
Sauve et efface la sélection	C-w
Marque le début d'un bloc	C-<space>
Efface la ligne	C-k
Copie le texte sauvé	C-y
Enlève toutes les partitions	C-x 1
Partionne l'éditeur de façon verticale/horizontale	C-x 2/C-x 3
Change de partition	C-x o

Table 9 – Raccourcis clavier de base pour Emacs

4. En allant dans le menu `Tools`→`Spell Checking`. Positionnez le dictionnaire pour être dans la langue de la page Wikipédia que vous avez choisie. Puis, activez l'option '*Flyspell*'. Faites traverser au curseur l'ensemble des lignes. Enfin, positionnez la souris sur un mot sous-ligné et appuyez sur le bouton du milieu.
5. Recherchez toutes les occurences du mot 'emacs' en avant (`C-s emacs`), puis en arrière (`C-r emacs`).
6. Positionnez vous au début du texte (`ESC-<`), puis remplacez toutes les occurences du mot '`emacs`' par '`vim`' (`M-%`). Notez que vous pouvez répondre oui à chaque coup en tapant '`!`' (mais seulement si vous êtes sûr).
7. Cela n'a pas loupé, vous avez remplacé trop de `emacs` par des `vim`, revenez en arrière sur les précédentes modifications (`C-_`).
8. Allez à la ligne 10 (`M-x goto-line`). Notez qu'en tapant le nom de la fonction vous pouvez utiliser la touche de complétion (<`TAB`>).
9. Rappelez la commande '`goto-line`' à travers un '`M-x M-p`' et allez en ligne 66.
10. Supprimez quelques lignes (`C-k`) puis restaurez les avec `undo`.
11. Explorez le menu '`Options`' et changez la taille des fontes.
12. Lisez la FAQ d'Emacs (`C-h C-f`) ainsi que la page suivante :
`http://guidespratiques.traduc.org/vf/Emacs-Beginner-HOWTO.html`

2.7 Quitter `vim(m)`

Si jamais vous vous retrouvez piégé dans un éditeur de texte inconnu et que rien ne marche, c'est que vous avez ouvert '`vi(m)`'. Pour quitter proprement, il faut :

1. Entrer en mode '*commande*' : '`:`' (deux points) ;
2. Quitter (force) : `q!`.

3 Les processus

Les processus sont des programmes en cours d'exécution, il s'agit d'une des abstractions majeures d'Unix (avec les fichiers). En fait, très souvent l'utilisateur a un certain nombre de programmes qui s'exé-

cutent sur sa machine. Il y a d'ailleurs souvent plus de programmes en cours d'exécution que de cœurs au processeur de la machine. Le concept de processus a donc été introduit pour permettre de partager efficacement les ressources d'une machine entre un nombre important de programmes en cours d'exécution. Ce qui donne à l'ensemble une impression de multi-tâche (même si en fait, les programmes sont exécutés les uns après les autres suivant un ordre et une pondération fixée par le système d'exploitation).

L'utilisateur a donc la possibilité de démarrer, d'interrompre (`Ctrl-z`), de reprendre ou encore de tuer définitivement (`Ctrl-c`) ses processus [10]. Le shell lui-même (ou encore le pointeur de la souris) est un processus et vous n'interagissez avec la machine qu'à travers des processus (et aussi des périphériques matériels tels que le clavier ou la souris).

Sachant cela, l'utilisateur a la possibilité d'influer sur ces processus et de mieux tirer parti du multi-tâche. C'est que nous allons voir à présent.

3.1 Connais-toi toi-même...

Fonction	Commande
Donne le nom de login de l'utilisateur	`logname/whoami`
Liste les utilisateurs logés sur la machine	`users/who`
Montre les derniers utilisateurs logés sur la machine	`last`

Table 10 – Commandes liées à l'utilisateur.

Questions

1. Lisez la page de manuel de chacune des commandes de la table 10.
2. Exécutez chacune des commandes pour en voir le résultat.

3.2 Rechercher un processus

Sur une machine, il existe un nombre insoupçonné de processus. Savoir les lister tous, repérer les plus gourmands en mémoire, en temps CPU ou encore ceux qui tournent depuis très longtemps reste une chose importante. Les commandes de la table 11 permettent de lister et trier les processus plus simplement.

Fonction	Commande
Affiche tous les processus en cours	`ps`
Affiche tous les processus en cours classés par activité	`top`

Table 11 – Commandes liées aux processus.

Questions

1. Lisez la page de manuel de chacune des commandes de la table 11.
2. Exécutez chacune des commandes pour en voir le résultat.
3. Expliquez ce que montrent les commandes suivantes [11] :
   - `ps aux | less`
   - `ps aux | grep $USER`
   - `ps aux | grep -v $USER`
4. Exécutez la commande `pstree` et observez l'arbre des processus. Quelle en est la racine ? Et essayez de retrouver les processus qui vous appartiennent.

10. Seulement les siens, car la notion de droits s'applique aussi aux processus comme aux fichiers.
11. Nous verrons l'utilisation des tubes ('|') un peu plus tard

3.3 Gérer les processus

Enfin, pouvoir gérer l'état des processus (les mettre en veille, en tâche de fond, ou encore les mettre au premier plan) permet de tirer plus de l'environnement Unix. La table 12, liste les commandes qui agissent directement sur les processus et leur état.

Fonction	Commande
Exécute le programme en le détachant du shell	`program &`
Envoi un signal à un processus	`kill`
Lance une commande qui se détache du shell	`nohup`
Règle la priorité d'un processus	`nice`
Attend le nombre de secondes indiqué en argument	`sleep`
Attends la terminaison d'un autre processus pour finir	`wait`
Affiche tous les processus issus du shell courant	`jobs`
Met un processus en '*background*'	`bg`
Remet un processus en '*foreground*'	`fg`

Table 12 – Commandes liées aux processus.

Questions

1. Comparez la commande '`emacs`' avec la commande '`emacs&`'. Qu'ont comme effet ces deux commandes sur le shell initial ?
2. Ouvrez deux consoles, puis :
   a. Dans l'une exécutez : `sleep 10000`.
   b. Dans l'autre, essayez de trouver le `PID` du premier processus.
   c. Dans la fenêtre du '`sleep`', faites `Ctrl-z`. Que notez-vous sur l'état du processus lorsque vous le cherchez à nouveau ?
   d. Revenez dans la console du '`sleep`' faites un '`jobs`', puis restaurez le processus '`sleep`' en faisant un '`fg`'.
3. Utilisez les commandes `nice`, `nohup` et `kill` après avoir lu leurs pages de manuel.

4 Les commandes shell

Ce que l'on appelle les commandes shell sont de petits utilitaires programmés dans l'esprit Unix. C'est à dire qu'ils sont fait pour appliquer des traitements simples à des flots de texte. Ils sont pensés pour pouvoir être combinés entre eux pour donner des résultats plus complexes.

4.1 Quelques utilitaires Unix

Il existe un très grand nombre de commandes Unix de base. Certaines sont standards et se retrouveront sur tous les systèmes, d'autres sont moins standard et doivent être installées spécifiquement. Nous parlerons ici des commandes standards les plus utiles (si l'utilité de certaines commandes de la table 13 ne vous apparaît pas tout de suite, vous verrez leur utilité surtout en programmation shell).

4.2 Combiner les commandes shell entre elles

L'intérêt des commandes shell est que l'on peut les combiner entre elles. La sortie d'une commande peut servir à nourrir une autre commande, ou bien elles peuvent s'éxecuter séquentiellement ou encore en parallèle l'une de l'autre. Les façons les plus classiques de les combiner entre elles sont explicitées dans la table 14.

Fonction	Commande
Affiche ou concatène des fichiers en sortie (*con**cat**enate*)	`cat`
Affiche la ligne de texte passée en paramètre	`echo`
Renvoi les n premières lignes d'un fichier	`head`
Renvoi les n dernières lignes d'un fichier	`tail`
Efface des sections d'une ligne	`cut`
Retire les répétitions dans une liste	`uniq`
Teste la véracité d'une expression	`test`
Trie les lignes d'un fichier ou de l'entrée standard	`sort`
Imprime une séquence de nombres	`seq`
Compte le nombre de mots, de lignes ou d'octets (***w**ord **c**ount*)	`wc`
Génère des permutations aléatoires	`shuf`
Copie `stdin` à la fois sur `stdout` et dans un fichier	`tee`
Découpe un fichier en morceaux suivant certaines conditions	`split`
Fusionne les lignes de deux fichiers	`join`
Positionne un timeout sur l'exécution d'une commande	`timeout`
Donne des statistiques sur le temps d'exécution d'une commande	`time`

Table 13 – Commandes shell utiles en Unix.

Fonction	Commande
Lance '`cmd`' en tâche de fond	`cmd &`
Exécute séquentiellement les deux commandes	`cmd ; cmd`
AND logique : Exécute la deuxième commande si et seulement si la première n'a pas renvoyé de code d'erreur	`cmd && cmd`
OU logique : Exécute la deuxième commande si et seulement si la première commande a renvoyé un code d'erreur	`cmd || cmd`
Écrit la sortie de '`cmd`' en écrasant le fichier '`file`'	`cmd > file`
Écrit la sortie de '`cmd`' en l'ajoutant à la suite du fichier '`file`'	`cmd >> file`
Écrit la sortie standard de '`cmd`' dans '`file`'	`cmd 1> file`
Écrit la sortie d'erreur de '`cmd`' dans '`file`'	`cmd 2> file`
Écrit la sortie standard et d'erreur de '`cmd`' dans '`file`'	`cmd &> file`
Envoie la sortie de la première commande sur l'entrée de la seconde	`cmd | cmd`
Grouper des commandes entre elles avec '{' et '}'	`cmd || { cmd ; cmd }`
Substitue le résultat de la commande '`cmd`' dans la ligne	`prefix $(cmd) suffix`

Table 14 – Combiner des commandes shell entre-elles.

Questions

1. Lisez les pages de manuels de chacune des commandes de la table 13.
2. Faites afficher à l'écran une séquence de nombres de 10 à 150 avec un incrément de 5. Puis, copiez cette liste dans un fichier '`test.txt`'.
3. Faites afficher le contenu du fichier '`test.txt`' en mélangeant aléatoirement les lignes.
4. Triez le contenu du fichier '`test.txt`' par ordre lexicographique et affichez le résultat. Puis, triez le par ordre numérique.
5. Écrivez une suite de commandes qui permettent de se déplacer dans un répertoire '`dir1`' et d'y créer un répertoire '`dir2`' si le déplacement a réussi.
6. Avec les commandes '`test`' et '`mkdir`', écrivez une ligne de commande qui teste si le répertoire '`toto`' existe, et, s'il n'existe pas, le crée.
7. Exécutez deux '`ls`' séquentiellement, puis parallèlement.
8. Avec les commandes '`cal`', '`tail`' et '`wc`', faites une commande qui compte combien de jours il y a dans le mois courant. Puis une commande qui dit combien de jours il y a eu dans le mois de février 1678.
9. Écrivez une ligne de commandes qui exécute '`ls -al`' et qui redirige la sortie standard vers le fichier '`stdout.txt`' et la sortie d'erreur vers le fichier '`error.txt`'. Refaites la même chose avec la commande '`ls -alz`'.