Université de Metz

U.F.R. SciF.A.

Nom :_____ **Prénom :**_____

Groupe :_____

T.P. de Système d'Exploitation Unix

M.S.T. Télécom 1re Année

Année Universitaire 2003/2004

Y. Morère

Cette page est laissée blanche intentionnellement

Table des matières

Les buts fixés pour cette série de tps sont les suivants :

- ☑ Se familiariser avec un système d'exploitation professionnel : UNIX.
- ☑ Maîtriser les commandes de base de ce système afin de réaliser des tâches simples (gestion de fichiers, courrier...).

Évaluation et notation :

- ☑ Le ou les compte-rendus comporteront les commandes saisies, les résultats obtenus ainsi que les réponses aux questions.

Avant de commencer un peu de lecture

Procédure de Login

Suivant la salle où vous vous trouvez (ici c'est la salle Stroustrup, en hommage au créateur du langage *C++*), le type de distribution Linux installée sur la machine, de petites différences dans le mode de connexion peuvent survenir.
Pour se connecter à l'une des machines, il faut bien entendu, avoir un accès à celle-ci c-à-d avoir un compte.
La plupart du temps, pour se connecter, il suffit de donner son nom de login et son mot de passe au logiciel de connexion (la plupart du temps `xdm`, `gdm` ou encore `kdm`). Vous obtenez une nouvelle fenêtre ressemblant à la figure 1.

FIG. 1 – Connexion à une machine en mode graphique (ici Ulysse avec `gdm`)

Login : saisissez ici votre **identifiant** ou **login** et `enter`
Password : saisissez ici votre **mot de passe** (Attention les lettres frappées s'affichent sous la forme d'étoiles) et `enter`.
Vous vous trouvez alors dans l'environnement LINUX, et vous avez une fenêtre du type de la figure 2.

FIG. 2 – Connexion réussie vous pouvez travailler

Toute la puissance d'un OS multitâches multi-utilisateurs est à vous. Mais saurez-vous en tirer partie ?

Remarque sur le mot de passe

- Vous seul connaissez le mot de passe.

- Si vous l'oubliez, la seule solution est de contacter l'administrateur réseau pour l'effacement de l'ancien mot de passe.
- Le mot de passe est composé d'au moins 6 caractères et au maximum 8 dont 2 au moins ne sont pas des lettres.
- Conseil : mélanger les lettres, chiffres, majuscules et minuscules.

Pour changer de mot de passe

- *yppasswd* ou *passwd* et `enter`
- La machine vous demande l'**ancien** mot de passe et `enter`
- Elle vous demande le **nouveau** et `enter`
- Puis elle redemande le **nouveau** pour confirmation et `enter`

Si vous n'avez pas de message d'erreur, le mot de passe est alors changé.

Terminer votre session

Pour terminer votre session UNIX (se *deloger*), vous pouvez saisir le commande `logout` ou `exit` et `enter` ou encore `ctrl-d`.

Forme et validation d'une commande

Une ligne de commande possède la forme générale suivante :

nom-de-commande [options] arguments

où l'*option* a la forme *-lettre*.

Exemple : Dans la commande
`ls -s -i toto titils` est le nom de la commande, `-s -i` sont les options, `toto` et `titi` sont les arguments. Plusieurs options peuvent être regroupées derrière le signe `-`.
`ls -si toto titi`
Une ligne de commande n'est reçue et exécutée par le système qu'après *validation* par `enter`.

Commande importante : `man`

La syntaxe est la suivante :
`man` *commande*
ou
`man -k` *commande*
Cette commande affiche la page de manuel correspondante à la *commande*. On y trouve tous les détails concernant son utilité, sa syntaxe et ses options.

Exemple : réponse du système à la commande `man ls` sur la machine `OSF1 flore V4.0 1091 alpha`

```
ls(1)

NAME

  ls - Lists and generates statistics for files

SYNOPSIS

  ls [-aAbcCdfFgilLmnopqrRstux1] [file...|directory...]

STANDARDS

  Interfaces documented on this reference page conform to industry standards
  as follows:

  ls:  XPG4, XPG4-UNIX
```

```
Refer to the standards(5) reference page for more information about indus-
try standards and associated tags.
```

Pour toutes questions au sujet de la commande `man`, taper :

```
man man
```

Cette page est laissée blanche intentionnellement

Chapitre 1

Travaillons un peu

1.1 Introduction

1.1.1 Reconnaître votre `shell`

`echo $SHELL`
Réponse :

1.1.2 Reconnaître votre système d'exploitation

`uname -a`
Réponse :

1.2 Commandes Générales

1.2.1 Changer votre mot de passe

`passwd` ou `yppasswd`
Réponse :

Observations :

1.2.2 Afficher la ligne (terminal) sur laquelle vous êtes connectés

`tty`
Réponse :

1.2.3 Afficher les paramètres de votre terminal

`stty`
Réponse :

Observations :

1.2.4 Reconfigurer la touche `erase`

`stty erase ^H`
Réponse :

1.2.5 Qui êtes-vous ?

`whoami`
Réponse :

`who am i`
Réponse :

Observations :

1.2.6 Qui est connecté sur la station ?

`who`
Réponse :

`w`
Réponse :

`users`
Réponse :

Observations :

1.2.7 Afficher du texte et les valeurs des variables système

`echo "Bonjour"`
Réponse :

`echo $DISPLAY`
Réponse :

1.2.8 Afficher les variables de votre système

`env`
Réponse :

Observations :

1.2.9 Création d'un fichier simple

Saisissez les lignes suivantes
`cat > fic.txt`
`mon premier fichier <enter>`
`bbbbbbbbbbbbbbbbbb <enter>`
`cccccccccccccccccc <enter>`
`aaaaaaaaaaaaaaaaaa <enter>`
`ctrl-d`
Réponse :

1.2.10 Visualisation de votre fichier

`cat fic.txt`
Réponse :

`more fic.txt`
Réponse :

`less fic.txt`
Réponse :

Observations :

1.2.11 Compter le nombre de lignes, de mots, de caractères du fichier

`wc fic.txt`
Réponse :

1.2.12 Recherche d'une chaîne dans un fichier

`grep aa fic.txt`
Réponse :

Observations :

1.2.13 Trier le contenu du ou des fichiers passés en argument

`sort fic.txt`
Réponse :

Cette page est laissée blanche intentionnellement

Chapitre 2

Passons la vitesse supérieure

2.1 Système de fichiers

2.1.1 Les caractères spéciaux

Il existe plusieurs caractères qui sont traités d'une manière différente pas l'interpréteur de commande. On les appelle des **caractères spéciaux** ou **métacaractères**. Ils comprennent :
- des caractères d'abréviations ;
- des caractères spéciaux liés au lancement d'une commande.

2.1.1.1 Les caractères d'abréviations

`*` remplace toute chaîne de caractères permettant de compléter un nom de fichier sauf une chaîne commençant par le caractère "`.`" lequel doit être explicitement écrit.

`?` peut représenter un caractère quelconque (à l'exception de "`.`").

[***liste de caractères***] désigne un caractère quelconque écrit entre crochet. Par exemple `foo.[cp]` désigne la liste des 2 fichiers `foo.c` et `foo.p`.

2.1.1.2 Les caractères spéciaux

`;` sépare les instructions successives d'une ligne de commande.

`()` servent à regrouper les commandes.

`>` redirection de la sortie standard.

`>>` redirection de la sortie standard sans écrasement.

`2>` redirection de l'erreur standard.

`2>>` redirection de l'erreur standard sans écrasement.

`<` redirection de l'entrée standard.

`&` lancement d'un processus en arrière plan (tâche de fond).

`|` communication par tube entre deux processus.

Exemple :
- La commande `ls *.txt` permet de lister tous les fichiers possédant l'extension `txt`.
- La commande `ls t?t?.txt` permet de lister tous les fichiers `titi.txt`, `toto.txt`, `tctc.txt` ...
- La commande `ls t[ao]t[ao].txt` permet de lister tous les fichiers possédant l'extension `tata.txt`, `tota.txt`, `toto.txt`, `tato.txt`.

2.1.2 Quel est le répertoire courant

`pwd`
Réponse :

2.1.3 Lister les fichiers dans le répertoire courant

`ls`
Réponse :

`ls -l`
Réponse :

`ls -al`
Réponse :

Observations :

2.1.4 Copier un fichier

`cp fic.txt fic_copie.txt`
Réponse :

Vérification :

2.1.5 Renommer un fichier

`mv fic_copie.txt nouveau_nom.txt`
Réponse :

Vérification :

2.1.6 Effacer un fichier

`rm nouveau_nom.txt`
Réponse :

Vérification :
Recréer un fichier `nouveau_nom.txt` et effacer le par `rm -i nouveau_nom.txt`

Réponse :

Vérification :

Observations :

2.1.7 Créer un lien symbolique sur un fichier

`ln -s fic.txt fic_lien.txt`
Réponse :

Vérification :

2.1.8 Créer un lien sur un fichier

`ln fic.txt fic_lien.txt`
Réponse :

Vérification :

Observations par rapport au lien symbolique :

2.1.9 Visualiser le fichier `fic_lien.txt`

Réponse :

Effacer le fichier original `fic.txt`, Tester à nouveau les liens :
Observations entre le lien symbolique et le lien simple :

2.1.10 Créer un répertoire

`mkdir mon_repertoire`
Réponse :

Vérification :

2.1.11 Changer de répertoire

`cd mon_repertoire`
Réponse :

Vérification :

2.1.12 Revenir au répertoire précédent

`cd ..`
Réponse :

Vérification :

2.1.13 Effacer un répertoire

`rmdir mon_repertoire`
Réponse :

Vérification :

2.1.14 Effacer un répertoire : le retour

Créer un répertoire `parent`, se déplacer dans celui-ci, créer un répertoire `enfant`
Revenir dans votre répertoire `home` et lancer :
`rm -r parent`
Réponse :

Vérification :

Observations :

2.1.15 Récapitulatif

Dans votre répertoire courant, créez en une commande les fichiers suivants : `annee1 Annee2 annee4 annee45 annee41 annee510 annee_saucisse annee_banane`

Réponse :

Créer le répertoire `Year` dans votre répertoire courant, en une commande déplacez les fichiers précédemment créés dans le répertoire `Year`.

Réponse :

Lister tous les fichiers :
- se terminant par 5
- commençant par annee4
- commençant par annee4 et de 7 lettres maximum
- commençant par annee avec aucun chiffre numérique
- contenant la chaîne ana
- commençant par a ou A

Réponse :

Copier les fichiers dont l'avant dernier caractère est un 4 ou 1 dans le répertoire `/tmp` en une seule commande.

Réponse :

2.1.16 Comprendre les droits d'accès

La commande `ls -al` affiche le résultat suivant :

```
drwxr-xr-x 5 ymorere prof 1024 Sep 21 14 :57 public_html
```

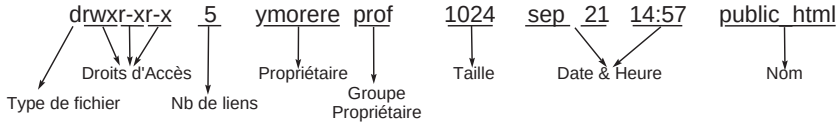

FIG. 2.1 – Signification des champs du résultat de la commande `ls -al`

Les droits d'accès sont les ensembles d'autorisations, qui déterminent *qui* peut avoir accès aux fichiers *en vue de quelle utilisation*.

droits d'accès	**utilisateurs**	**commandes**
lecture = **r**	propriétaire = **u**	ajouter = **+**
écriture = **w**	groupe = **g**	enlever = **-**
exécutable = **x**	autre = **o**	initialiser = **=**
pas d'accès = **-**	tous = **a**	

TAB. 2.1 – Tableau récapitulatif des droits d'accès

2.1.17 Modification des droits d'accès

Commande `chmod` *utilisateurs+/-droit fichier/répertoire*
Changer les droits d'accès de votre fichier `fic.txt` : vous devez être le seul à pouvoir lire et modifier votre fichier.
Réponse :

Vérification :

Observations :

Créer un répertoire et visualiser ses droits d'accès. Modifier ses droits d'accès afin que tout le monde puisse accéder à celui-ci mais ne puisse pas y faire de modification.
Réponse :

Vérification :

Observations :

2.1.18 Donner le type du fichier

Commande `file` *`nom_du_fichier`*
Exécuter la commande avec un fichier
Réponse :

Exécuter la commande avec un répertoire
Réponse :

Observations :

2.1.19 Trouver un fichier dans l'arborescence

Commande `find / -name "talk" -print`
recherche le fichier `talk` à partir du répertoire racine `/`
Rechercher les fichiers commençant par `x` dans le répertoire `/usr/bin`
Réponse :

2.1.20 Afficher les caractéristiques des disques

Commande `df`
Réponse :

Observations :

2.1.21 Récapitulatif

Dans votre répertoire courant, créez un répertoire courant `essai_droit`, par défaut ce répertoire est à `755 (rwxr-xr-x)`, quelles sont les commandes (en notation symbolique et en base 8) pour lui donner les droits suivant (on suppose qu'après chaque commande on remet le répertoire à `755` :

	propriétaire			**utilisateur**			**autre**		
	read	write	execute	read	write	execute	read	write	execute
commande 1	oui	oui	oui	oui	non	oui	non	non	oui
commande 2	oui	non	oui	non	oui	non	non	non	oui
commande 3	non	oui	non	non	non	oui	oui	non	non
commande 4	non	non	oui	oui	non	oui	non	non	non

Réponse :

Créez un fichier `droit` dans le répertoire `essai_droit`, par défaut ce fichier est à `644 (rw-r-r-)`.En partant du répertoire courant, pour chaque commande de l'exercice précédent, essayez d'accèder au répertoire `essai_droit` (commande cd), de faire un ls dans `essai_droit` et de modifier le fichier avec un éditeur quelconque (vi par exemple).
Réponse :

Tapez la commande `umask`, de manière à ce que les fichiers lors de leur création aient par défaut les droits `640 (rw-r---)`, et les répertoires `750 (rwxr-x--)`.

Réponse :

Chapitre 3

On a toujours besoin d'un éditeur de texte

3.1 Les éditeurs de texte

Un éditeur de texte est un outil pour écrire un fichier texte pur c'est à dire sans mise en forme (il ne contient que des caractères de la table ASCII) (`edit` sous DOS, `turbo` pour la programmation TurboPascal), alors qu'un traitement de texte est un outil pour écrire et mettre en forme du texte (*WordPerfect*, *MS Word*).

3.1.1 vi

`vi` est un éditeur vidéo, c'est à dire pleine page et interactif. Il est possible d'insérer du texte à tout endroit dans le fichier en édition. (Cf. figure 3.1)
L'appel de `vi` s'effectue de la manière suivante :

- `vi`
- `vi` *fichier*
- `vi` *+n fichier* pour se placer directement à la *n*-ième ligne du *fichier*
- `vi` *+/motif fichier* pour se placer directement à la première occurrence du *motif* dans le *fichier*

FIG. 3.1 – Fenêtre d'édition pleine page vi

Il dispose de 3 modes de travail :

Mode commande permet les déplacements, les recherches, les destructions, etc... C'est le mode par défaut.

Mode exécution permet l'utilisation de toutes les commandes de `ed` (éditeur en ligne UNIX). Mode utilisé pour les commandes globales et les commandes gérant les fichiers.

Mode saisie permet la saisie de texte.

3.1.1.1 Mode saisie

Pour passer en mode saisie (depuis le mode commande) il faut taper une des commandes d'édition :

- `a` pour "append" (ajout) ajoute du texte après le curseur.
- `A` pour "append" (ajout) ajout du texte en fin de ligne.
- `i` pour "insert" (insertion) insère du texte devant le curseur.

- `I` pour "insert" (insertion) insère du texte en début de ligne.
- `o` pour "open" (ouvrir) ouvre une ligne après le curseur.
- `O` pour "open" (ouvrir) ouvre une ligne en fin de ligne.

3.1.1.2 Mode commande

Pour repasser en mode commande il faut appuyer sur `ESC`.

3.1.1.3 Déplacement du curseur

Déplacement du curseur en mode commande :

- `x`,`j`,`k` et `l` déplacent le curseur dans les 4 directions (O, S, N et E). Ces touches peuvent être remplacées par les flèches du clavier.
- `w` place le curseur au début du mot suivant
- `b` place le curseur au début du mot précédent
- `e` place le curseur à la fin du mot courant
- `0` place le curseur en début de ligne
- `$` place le curseur en fin de ligne
- `enter` place le curseur au début de la ligne suivante
- `^F` avance le curseur d'une page
- `^B` recule le curseur d'une page
- `G` place le curseur en fin de fichier
- `n`*`G`* place le curseur à la *n*-ième ligne du fichier
- `/`*motif*`/` place le curseur à la prochaine occurrence du *motif*
- `m`*c* définit une *marque* : elle associe la position du curseur au caractère *c*. On peut alors retourner d'un autre endroit du fichier à cette position par la commande `'c` ou au début de la ligne par `'c`

3.1.1.4 Commandes d'effacement et remise de texte

- `x` efface le caractère sous le curseur
- `dw` efface un mot
- `db` efface le mot précédent
- `D` efface la fin de la ligne
- `dd` efface la ligne du curseur
- `r`*c* remplace le caractère courant par *c*
- `~` remplace une minuscule par une majuscule et vice-versa
- `p` réinsère le texte dernièrement effacé

3.1.1.5 Insertion d'un fichier extérieur

- `:r` *fichier* insère après la ligne courante un fichier extérieur

3.1.1.6 Annulation de la dernière commande

- `u` annule la dernière commande effectuée

3.1.1.7 Recherche et remplacement

- `:/`*motif* place le curseur sur la prochaine occurrence de *motif*
- `:s/`*motif*`/`*chaîne*`/` remplace dans la ligne courante la première occurrence de *motif* par la *chaîne*
- `:s/`*motif*`/`*chaîne*`/g` remplace dans la ligne courante toute occurrence de *motif* par la *chaîne*
- `:1,10s/`*motif*`/`*chaîne*`/g` remplace toute occurrence de *motif* par la *chaîne* de la première à la dixième ligne
- `:.,$%s/`*motif*`/`*chaîne*`/g` remplace toute occurrence de *motif* par la *chaîne* depuis la ligne courante (désignée par `"."`) jusqu'à la dernière ligne du fichier (désignée par `"$"`)
- `:%s/`*motif*`/`*chaîne*`/g` remplace dans tout le fichier toute occurrence de *motif* par la *chaîne*

3.1.1.8 Déplacement de texte

- `m`*c* définit le début de la section à déplacer
- se déplacer à la fin de la section à déplacer et frapper `d'`*c*, la fraction de texte est alors supprimée et placée dans le tampon
- se déplacer à l'endroit voulu et frapper `p` qui insère le texte contenu dans le tampon.

3.1.1.9 Mode exécution

Le mode exécution est activé depuis le mode commande par la caractère ' :' suivi de la commande :
- `:w` *nom_fichier* sauvegarde le fichier *nom_fichier*
- `:w` sauvegarde du fichier
- `:wq` sauvegarde et quitte `vi`
- `:q !` quitte `vi` sans sauvegarde
- `:x` équivalent à `:wq`

3.1.1.10 Initiation à Vim : `vimtutor`

Copier dans un de vos répertoire le fichier `tutor.fr` qui se trouve dans le répertoire `/serveur/home/profs/morere/tutor/`.

Éditez le à l'aide de vim, et réalisez les chapitres d'initiation. Ce qui suit consitue le premier écran du tutoriel de Vim.

```
===============================================================================
= B i e n v e n u e  dans le  T u t o r i e l  de  V I M  -  Version 1.5.fr.1 =
===============================================================================

     Vim est un éditeur très puissant qui a trop de commandes pour pouvoir
     toutes les expliquer dans un cours comme celui-ci, qui est conçu pour en
     décrire suffisamment afin de vous permettre d'utiliser simplement Vim.

     Le temps requis pour suivre ce cours est d'environ 25 à 30 minutes, selon
     le temps que vous passerez à expérimenter.  Les commandes utilisées dans
     les leçons modifieront le texte. Faites une copie de ce fichier afin de
     vous entraîner dessus (si vous avez lancé "vimtutor" ceci est déjà une
     copie).

     Il est important de garder en tête que ce cours est conçu pour apprendre
     par la pratique.  Cela signifie que vous devez exécuter les commandes
     pour les apprendre correctement.  Si vous vous contentez de lire le
     texte, vous oublierez les commandes !

     Maintenant, vérifiez que votre clavier n'est PAS verouillé en majuscules,
     et appuyez la touche  j  le nombre de fois suffisant pour que la leçon
     1.1 remplisse complètement l'écran.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

3.1.2 emacs

Les principaux avantages d'`emacs` sont l'extensibilité, la personnalisation et l'auto-documentation. Il possède de nombreuses fonctionnalités autres que celles de l'édition.
On peut compiler un programme, lire du courrier électronique, lire les forums, récupérer un fichier par ftp... `emacs` signifie editor macros.

L'appel d'`emacs` s'effectue de la manière suivante :
- `emacs`
- `emacs` *fichier*
- `emacs` *+n fichier* pour se placer directement à la *n*-ième ligne du *fichier*

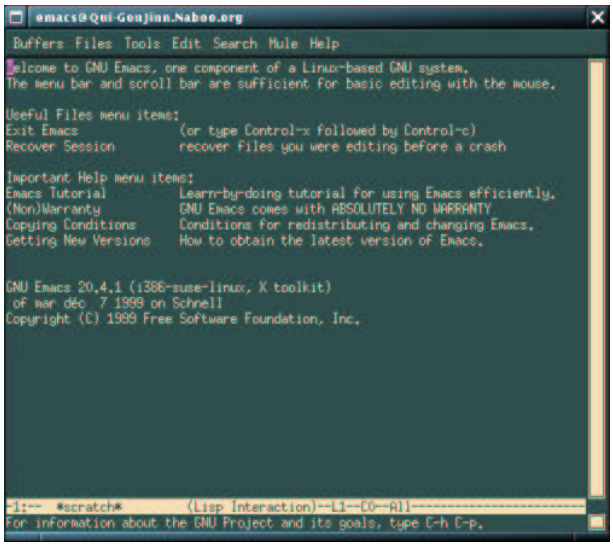

FIG. 3.2 – Fenêtre d'édition `emacs`

La plupart des commandes `emacs` font intervenir :
- la touche `<CTRL>` frappée en même temps qu'un autre caractère. Elle est habituellement notée `C-`. Par exemple `C-p` désigne la touche `<CTRL>` frappée en même temps que `p`.
- la touche `META`, existante sur la clavier, soit la touche `<ALT>` ou encore la touche `<ESC>` (dans ce cas elle doit être relâchée avant la touche qui suit). Elle est notée `M-`.

3.1.2.1 Les modes d'édition d'`emacs`

Selon que l'on travaille sur un fichier texte, un programme C, un fichier de données, les besoins d'édition sont différents. `emacs` propose donc plusieurs modes d'édition qui définissent un environnement de travail adapté au type de fichier.
Modes principaux :
- le mode **Fundamental**
- le mode **Text**
- le mode **Lisp**
- le mode **C**

Il existe des modes secondaires utilisés en conjonction avec un mode principal :
- le mode **Fill** : les lignes sont automatiquement coupées quand elles dépassent la marge droite.
- le mode **Abbrev** : l'expansion des abréviations est impossible.
- le mode **Ovwrt** : mode recouvrement.
- le mode **Narrow** : rend accessible qu'une partie du buffer.

3.1.2.2 Notion de buffer

C'est une partie de la mémoire d'`emacs`. A chaque ouverture de fichier un buffer portant le même nom est créé. Les modifications n'affectent pas tout de suite le fichier sur disque mais le buffer. Il faut sauver le fichier pour que les modifications soient effectuées sur le fichier d'origine.

3.1.2.3 Commande de déplacement du curseur

- `C-p` place le curseur sur la ligne précédente
- `C-n` place le curseur sur la ligne suivante
- `C-f` avance le curseur d'un caractère
- `M-f` avance le curseur à la fin du mot courant ou suivant
- `C-b` recule le curseur d'un caractère
- `M-b` recule le curseur au début du mot courant ou du précédent
- `C-a` place le curseur en début de ligne
- `C-e` place le curseur en fin de ligne
- `M-<` place le curseur en début de fichier
- `M->` place le curseur en fin de fichier
- `C-v` avance la fenêtre d'un écran
- `M-v` recule la fenêtre d'un écran

Toutes ces commandes peuvent être envoyées avec un argument numérique *n*. L'argument numérique est introduit par `C-u`.
La commande `C-u 8 C-n` avance le curseur de 8 lignes.

3.1.2.4 Insertion et suppression de texte

- `C-d` efface le caractère sur lequel se trouve le curseur
- `C-k` efface la fin de la ligne
- `C-y` réinsère le texte effacé
- `C-x u` annule l'effet de la dernière modification

3.1.2.5 Édition simultanée de plusieurs fichiers

- `C-x C-f` *fichier* crée un nouveau tampon et y place le *fichier*
- `C-x C-b` ouvre un nouveau tampon et une nouvelle fenêtre sur l'écran et y affiche la liste de tous les tampons ouverts (cette fenêtre peut être supprimée avec `C-x 1`)
- `C-x b` *tampon* place la curseur dans le *tampon*
- `C-x C-v` *fichier* place le *fichier* dans le tampon courant, son contenu actuel est éliminé.
- `C-x k` *tampon* supprime le *tampon* (par défaut c'est le tampon courant qui est supprimé)

3.1.2.6 Recherche et remplacement

- `C-s` *mot* recherche la première occurrence du *mot* dans la suite du texte. A chaque fois que l'utilisateur frappe une nouvelle lettre du *mot*, le curseur se place sur la prochaine occurrence de ce qui a été frappé. La répétition de `C-s` recherche l'occurrence du *mot* suivant.
- `C-r` *mot* a le même effet que `C-s`, mais la recherche se fait sur le texte qui précède.
- `M-%` *chaîne nouvelle_chaîne* remplace une *chaîne* de caractères par une *nouvelle_chaîne*. L'utilisateur doit valider chaque remplacement. S'il frappe `!` les validations sont omises

3.1.2.7 Insertion d'un fichier

- `C-x i` *fichier* insère le *fichier* à l'endroit du curseur

Pour n'insérer qu'une partie du fichier effectuer les commandes suivantes :
- `C-x C-f` *fichier* insère le *fichier* dans un nouveau tampon
- Placer le curseur au début de la région à insérer
- `C-@` ou `C-ESPACE` marque le début de la région
- Placer le curseur à la fin de la région à insérer
- `C-w` efface la région délimitée
- `C-x b` revient au tampon initial
- Se placer à l'endroit voulu de l'insertion
- `C-y` insère le contenu du tampon auxiliaire

3.1.2.8 Suppression de fenêtres

- `C-x 1` supprime toutes les fenêtres qui ont été ouvertes à l'exception de celle ou se trouve le curseur

3.1.2.9 Sauvegarder et quitter `emacs`

- `C-x C-s` sauvegarde les modifications effectuées si `emacs` connaît le nom de fichier. Sinon l'utilisateur est invité à entrer un nom de fichier dans le mini tampon.
- `C-z` quitte `emacs` provisoirement (le processus est suspendu). Le retour à `emacs` se fait en frappant `fg` ou encore `%emacs`.
- `C-x C-c` quitte `emacs` définitivement.

3.1.3 nedit, gedit, gnotepad & kedit

`nedit`, `gedit`, `gnotepad` & `kedit` sont des éditeurs graphiques qui ressemblent beaucoup aux éditeurs rencontrés sous les environnements PC, Mac, Amiga (Linux, Windows 3.xx et 9x, DOS, Mac OS, Amiga OS). `nedit`, `gedit`, `gnotepad` & `kedit` peuvent être complètement gérés à la souris et possèdent des menus conviviaux pour l'édition, la recherche de texte, le copier coller, la sélection de texte.
`nedit` utilise la bibliothèque graphique Motif, alors que `gedit, gnotepad` utilisent les bibliothèques de l'environnement gnome (GLIB, GTK+) et `kedit`, lui est basé sur KDE et les bibliothèques QT.

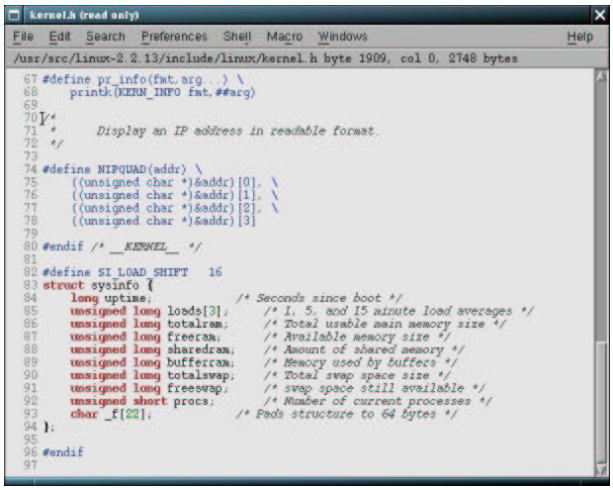

FIG. 3.3 – Fenêtre de l'éditeur graphique `nedit`

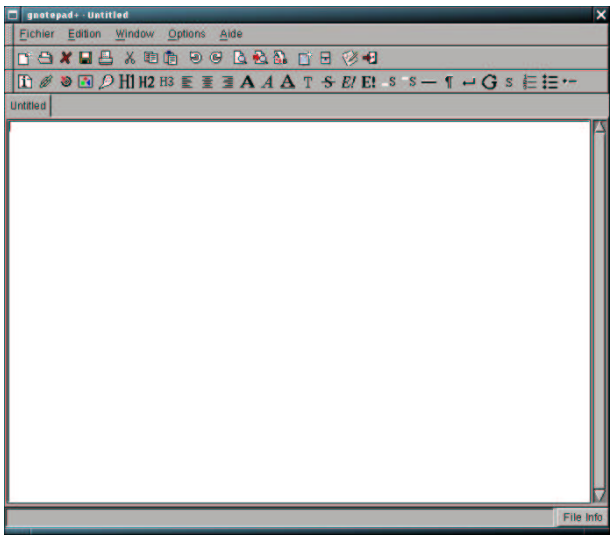

FIG. 3.4 – Fenêtre de l'éditeur graphique `gnp`

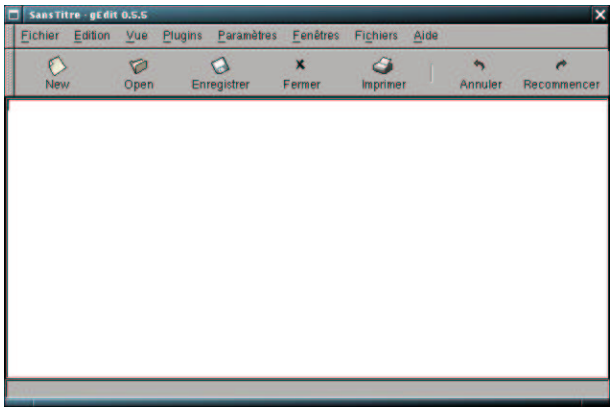

FIG. 3.5 – Fenêtre de l'éditeur graphique `gedit`

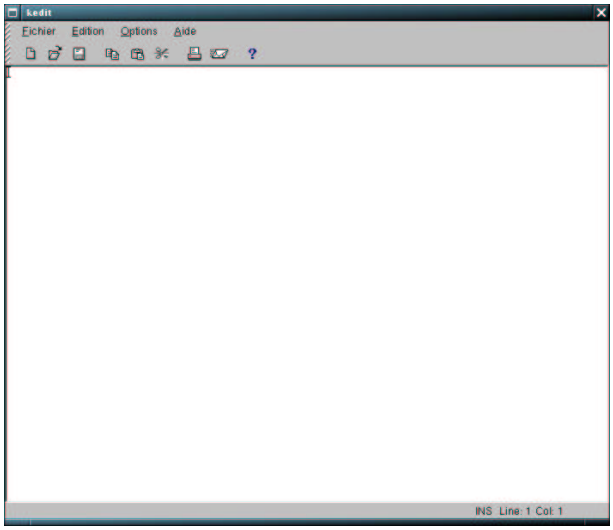

FIG. 3.6 – Fenêtre de l'éditeur graphique `kedit`

Il possède aussi des prédispositions pour la programmation. Il gère l'auto-indentation des lignes de programmes, la gestion du nombre des parenthèses, l'affichage des mots-clés du langage.

Cette page est laissée blanche intentionnellement

Chapitre 4

Unix : Un vrai OS multi-tâches

4.1 Gestion des processus

4.1.1 Introduction

Un processus est un programme binaire en cours d'exécution. Il possède les caractéristiques suivantes :
- identificateur : `PID`
- identificateur du père : `PPID`
- identificateur de l'utilisateur : `UID`
- le répertoire courant
- les fichiers ouverts
- le masque de création : `umask`
- la taille maximum des fichiers créés par le processus : `ulimit`
- la priorité
- les temps d'exécution
- le terminal de contrôle

4.1.2 Obtenir l'`UID` et le `GID`

`id`
Réponse :

Observations :

4.1.3 Visualiser les processus

`ps`
Réponse :

`ps -u`*votre_login*
Réponse :

`ps -aj`
Réponse :

Observations :

4.1.4 Lancer un processus en tâche de fond

nom_du_processus`&`
Lancer `kedit` en tâche de fond
Réponse :

Lancer `kedit`
Réponse :

Observations :

4.1.5 Tuer un processus

`kill -`*nom_signal numéro_de_processus*
Tuer le processus `kedit`

Réponse :

Observations :

4.1.6 Bloquer un processus

Si vous avez lancer un processus long en oubliant de le placer en tâche de fond, il est intéressant de pouvoir le bloquer afin de "reprendre la main" et de le relancer en tâche de fond.
Frapper `<CTRL-Z>`, stoppe le processus en cours et vous "rend la main", il vous est alors possible de la placer en tâche de fond.

4.1.7 Passer un processus en arrière plan

`bg` *numéro_de_job* ou `bg`
Lancer `nedit`, bloquer le, et passer le en tâche de fond
Réponse :

Observations :

4.1.8 Lancer une commande et afficher les temps consommés

`time` *commande*
Lancer une recherche sur les fichiers commençant par `X` dans toute l'arborescence et afficher les temps consommés.
Réponse :

Observations :

Cette page est laissée blanche intentionnellement

Chapitre 5

Vous serez bientôt des gourous UNIX

5.1 Redirections des entrées/sorties, tubes et filtres

5.1.1 Introduction

Tout processus communique avec l'extérieur par l'intermédiaire de trois fichiers appelés *fichiers standards* :
- Le fichier *entrée standard* sur lequel le processus lit ses données
- Le fichier *sortie standard* sur lequel le processus écrit ses résultats
- le fichier *sortie erreur standard* sur lequel le processus écrit ses messages d'erreurs

Par défaut ces fichiers sont associés au terminal :
- l'*entrée standard* est le clavier
- la *sortie standard* et *sortie erreur* sont l'écran

Il est possible de *rediriger* les entrées/sorties standards d'un processus. On peut leur associer un fichier autre que le terminal.
- *commande* `<` *référence* : redirige l'entrée standard de la *commande* sur le fichier dont on donne la *référence*
- *commande* `>` *référence* : redirige la sortie standard *avec écrasement* du fichier nommé
- *commande* `>>` *référence* : redirige la sortie standard *sans écrasement* du fichier nommé
- *commande* `2>` *référence* : redirige la sortie d'erreur *avec écrasement* du fichier nommé
- *commande* `2>>` *référence* : redirige la sortie d'erreur *sans écrasement* du fichier nommé

5.1.2 Créer un fichier contenant la liste des utilisateurs connectés

Réponse :

Vérification :

5.1.3 Enchaînement de processus

Il est possible, dans une même ligne de commande, de lancer plusieurs commandes qui vont s'exécuter soit *séquentiellement*, soit *concurremment* avec communication entre elle par l'intermédiaire d'une zone mémoire appelée *tube (pipe)*.

5.1.4 Lancement en séquence de plusieurs commandes

commande1 `;` *commande2* ou `(`*commande1* `;` *commande2*`)`
Mettre la date et la liste des utilisateurs connectés dans un fichier `essai`
Réponse :

Vérification :

5.1.5 Communication entre processus

commande1 | *commande2* envoie directement la sortie de la *commande1* vers l'entrée de la *commande2*
Lister les fichiers de votre répertoire et afficher ceux qui contiennent `ess` dans leur nom
Réponse :

Vérification :

Chapitre 6

$SHELL : csh répondit l'echo

6.1 Le Shell

6.1.1 Introduction

Sous Unix, il existe plusieurs *langages de commande*, les plus connus sont les suivants : le Bourne-Shell (`sh`), le C-Shell (`csh`), le Bash (Bourne Again Shell de Linux) (`bash`) et le Korn-Shell (`ksh`).
Dans la suite, on passera sous le shell `bash` par la commande `/bin/bash`.

6.1.2 Les variables du Shell

Le Shell donne à l'utilisateur la possibilité de définir des **variables** qui peuvent être utilisées dans la construction de commandes complexes.
Un certain nombre de variables sont prédéfinies dès le moment où l'utilisateur se loge.

Une variable possède un *nom* et une *valeur*. Son *nom* est une chaîne de caractères commençant par une lettre et composée de lettres, de chiffres et du caractère `_`. Sa *valeur* est une chaîne de caractère quelconque.

6.1.3 Définition de nouvelles variables

Le mécanisme d'*affectation* avec la syntaxe *nom*=*valeur* permet de définir une nouvelle variable et de lui affecter une valeur. La valeur de la variable *nom* est donnée par la chaîne `$nom` ou `${nom}` s'il faut isoler la variables des caractères qui suivent.

Exemple :

```
$ x=gh
$ echo $x
gh
$echo $xijk

$echo ${x}ijk
ghijk
$
```

6.1.3.1 Créer une variable `var` contenant la valeur `etudiant_mst_telecom`

Réponse :

Vérification :

6.1.4 Mécanisme de l'accent grave

Le shell substitue un commande placée entre des accents graves ‘ par la chaîne de caractères qui serait envoyée sur la sortie standard.

Exemple :

```
$ a=‘pwd‘
$ echo $a
/usr/lib
$
```

6.1.4.1 Créer une variable `var2` qui renvoie la date du système lorsque l'on veut afficher son contenu

Réponse :

Vérification :

6.1.5 La commande interne `read`

On peut affecter à une ou plusieurs variables des valeurs lues sur l'entrée standard au moyen de la commande interne **`read`** avec la syntaxe suivante : **`read`** $var_1\ var_2 \ldots var_n$.
La commande **`read`** analyse la ligne lue sur l'entrée standard et affecte les chaînes successives aux différentes variables $var_1\ var_2 \ldots var_n$.

Exemple :

```
$ read nom adresse

lucifer 35, rue de la Gehenne    <-- entre au clavier
$ echo $nom
lucifer
$ echo $adresse
35, rue de la Gehenne
$
```

6.1.5.1 Utiliser la commande `read` afin de créer différentes variables contenant votre adresse

Réponse :

Vérification :

6.1.6 Variables prédéfinies

Les variables prédéfinies par défaut sont les suivantes.

`PS1` a pour valeur le premier caractère de l'invite (prompt).

`PS2` a pour valeur le second caractère de l'invite.

`HOME` a pour valeur la référence absolue du répertoire de l'utilisateur.

`LOGNAME` a pour valeur l'identification de l'utilisateur.

`PATH` est une variable très importante. Sa valeur est une chaîne de caractères indiquant une liste de références de tous les répertoires susceptibles de contenir des commandes utilisées par l'utilisateur.

`IFS` a pour valeur l'ensemble des caractères interprétés comme *séparateurs de chaîne* par le shell.

`TERM` est également une variable importante. Elle indique le type de terminal utilisé.

6.1.7 Les commandes `set` et `unset`

La commande `set` permet d'obtenir la liste des variables de l'environnement et de leurs valeurs. La commande `unset` permet de supprimer une variable.

6.1.7.1 Utiliser la commande `set` pour visualiser les variables prédéfinies

Réponse :

Observations :

6.1.8 Variables exportables

Afin d'ajouter une nouvelle variable à l'environnement shell on utilise la commande `export` ou `setenv`. On peut visualiser les variables à l'environnement shell par la commande `env`.

6.1.9 Exécution d'un fichier de commande

On peut rendre un fichier de commande exécutable de 4 manières :

1. en envoyant la commande `sh fichier`, il suffit dans ce cas que `fichier` soit accessible en lecture.
2. en lançant la commande `. fichier`, il suffit dans ce cas que `fichier` soit accessible en lecture.
3. en envoyant la commande `fichier`, mais là il faut que `fichier` soit accessible en lecture, en écriture et en **exécution**.
4. en envoyant la commande `exec fichier` ou `source fichier`, mais là il faut que `fichier` soit accessible en lecture, en écriture et en **exécution**.

6.1.10 Variables maintenues par le shell

Grâce à ces variables, il est possible, à l'intérieur d'un fichier de commande, de faire référence aux arguments de la ligne de commande.

6.1.10.1 Variables de contrôle

Elles donnent des informations sur les processus en cours.

`$` a pour valeur le numéro de processus shell en cours

`!` a pour valeur le numéro du dernier processus lancé en background

`?` a pour valeur le code de retour de la dernière commande exécutée. 0 si la dernière commande s'est exécutée convenablement, non nulle sinon.

6.1.10.2 Variables de position et paramètres d'un fichier de commande

Tout processus shell maintient une liste de chaînes de caractères que l'on peut connaître par la valeur des variables `*`, `#`,`1`, `2`, `3`, `...`, `9`.

`#` a pour valeur le nombre de chaînes présentes dans la liste

`*` a pour valeur la liste des chaînes de caractères

`i` pour `i=1,...,9` a pour valeur la `i`-ème chaîne de caractères

Remarque : Si le processus shell est crée pour exécuter une commande avec des arguments, alors la variable `0` prend pour valeur le nom de la commande et `*` prend pour valeur la liste des arguments.

6.1.10.3 Utiliser les paramètres positionnels

Saisir dans un fichier le script suivant :

```
echo procedure
echo il y a $# paramètres
echo qui sont [$*]
echo le troisième est $3
echo tous les paramètres sont contenus dans la liste [$@]
echo ce script a comme PID [$$]
```

Rendre le script exécutable et le lancer avec les paramètres `a b c d e f g h`
Réponse :

Observations :

6.1.10.4 La commande `expr`

La commande `expr` considère la suite de ses arguments comme une expression (numérique ou chaîne de caractères). Elle l'évalue et affiche le résultat sur la sortie standard.

```
$ expr 4 + 7
11
$
```

Attention :

1. les différents termes intervenant doivent être séparés par des espaces.
2. si les opérateurs utilisés sont des caractères spéciaux, il doivent être déspécialisés (exemple : `>` doit être écrit `\>`).
3. on peut utiliser les parenthèses `(` et `)` pour regrouper des termes (il faut également les déspécialiser).

Les opérateurs utilisables sont les suivants :

- Le **OU** logique `|` : *expression*$_1$ `|` *expression*$_2$ a pour valeur celle de *expression*$_1$ si *expression*$_1$ n'est pas nulle (chaîne vide ou 0) et sinon pour valeur celle de *expression*$_2$ (ou 0 si *expression*$_2$ est vide).
- Le **ET** logique `&` : *expression*$_1$ `&` *expression*$_2$ a pour valeur celle de *expression*$_1$ si *expression*$_1$ et *expression*$_2$ sont toutes 2 non nulles et non vides ; vaut 0 dans le cas contraire.
- Les opérateurs de **comparaisons** : `<`,`>`,`=`,`>=`,`<=`, `!` (différent de). *expression*$_1$ *opérateur* *expression*$_2$ vaut 1 si le résultat de la comparaison est vrai, 0 sinon.
- Les opérateurs **additifs** : `+` et `-`
- Les opérateurs **multiplicatifs** : `*` multiplication, `/`division, `%` reste.

Excercice : Écrire le script qui permet de renvoyer la valeur en € d'une valeur entrée en francs.

Réponse :

Observations :

Excercice : Écrire le script `somme` qui permet de faire l'addition des nombres saisis au clavier (0 pour finir).

Réponse :

Chapitre 7

Filtrer et traîter les chaînes de caractères

7.1 Les filtres

On appelle un *filtre* toute commande qui lit sur son entrée standard, modifie (éventuellement) les données lues et écrit les résultats sur sa sortie standard. En redirigeant l'entrée standard et la sortie standard sur des fichiers, on peut alors prendre en entrée un fichier source et récupérer en sortie un autre fichier.
On peut combiner les filtres entre eux sur la même ligne de commande. Le plus souvent, ils sont utilisés avec des tubes pour former des commande complexes. Les filtres les plus utilisés sont `cat`, `wc`, `tail`, `tr`, `sort`, `sed` et `grep`.

7.1.1 Les expressions régulières

Certains utilitaires de filtrage comme les commandes `grep`, `ed`, `sed`, `more` utilisent des *expressions régulières* pour décrire des lignes de texte.

`.`	désigne n'importe quel caractère, excepté le passage à la ligne
`[...]`	désigne n'importe quel caractère contenu dans les crochets. On peut désigner plusieurs caractères qui se suivent par un tiret (`-`). Par exemple, `[a-z]` désigne une lettre en minuscule, `[0-9a-zA-Z]` un caractère alphanumérique. Si le caractère `^` se trouve en début des crochets, l'expression désigne n'importe quel caractère qui n'est pas entre crochets.
`^`	désigne un début de ligne lorsqu'il est placé en début d'expression.
`$`	désigne une fin de ligne lorsqu'il est placé en fin d'expression.
`\`	permet de déspécialiser le caractère qui suit.
`*`	désigne aucune ou au moins une occurrence du caractère précédent.

7.1.1.1 Le chapeau `^` et le dollar `$`

Les caractères `^` et `$` définissent une position dans la ligne de texte analysée. Le caractère `^` représente le début de ligne ou d'expression, et le caractère `$` représente la fin de ligne ou d'expression.

Exemple : ainsi pour rechercher dans un fichier, toutes les lignes commençant par la lettre B, on aura :

```
$ grep "^B" fichier
Bonbon
Bouton
```

Exemple : ainsi pour rechercher dans un fichier, toutes les lignes se terminant par le mot "Paris", on aura :

```
$ grep "Paris$" fichier
12h15 Paris
20h30 Paris
```

Exercice : Écrire la commande qui permet de rechercher les lignes blanches d'un fichier pour les compter.

Réponse :

Exercice : Écrire la commande qui permet rechercher dans un fichier, les lignes qui commencent par "Henri" et qui se termine par "Paris".

Réponse :

7.1.1.2 Le point .

Le point "." permet de représenter un caractère quelconque dans une expression.

Exemple : ainsi pour rechercher dans un fichier "`F1`", toutes les lignes contenant une chaîne de 8 caractères se terminant par "`al`" , on aura :

```
$ grep '......al' F1%
```

Exercice : Écrire la commande qui permet rechercher les programmes de `/usr/bin` dont le nom a une longueur de 4 caractères et se terminant par un "`r`". On commencera l'expression par un espace afin de limiter à 4 caractères.

Réponse :

7.1.2 Récapitulatif

Exercice : Créer un répertoire essai-grep dans votre home directory. Dans ce répertoire créer les fichiers suivants : `tomate poire pomme cerise Fraise fraise courgette POMME3 afraise`.
Editez les fichiers (sortie de la commande `ls` redirigée vers `grep`) avec les critères sur leur nom suivant :

Critère 1 : Le nom doit être Fraise ou fraise

Critère 2 : se est en fin de nom

Critère 3 : ai est présent dans le nom

Critère 4 : Nom contenant un chiffre numérique

Critère 5 : Nom contenant la chaîne mm ou MM

Réponse :

Exercice : Copiez le fichier /etc/passwd dans votre home directory. Editez la ligne commençant par votre nom de login. Réponse :

Exercice : Dans le fichier passwd qui est dans votre home directory, éditez les lignes commençant par des noms de login ne contenant pas de chiffre. Réponse :

Exercice : Editez les lignes du fichier passwd commençant par des noms de login de 3 ou 4 caractères. Réponse :

7.1.2.1 Les classes de caractères

Les crochets `[]` permettent de spécifier des classes de caractères recherchés dans un fichier. La recherche donne un résultat lorsqu'un des caractères, au moins, se trouvant entre crochets, est retrouvé dans une ligne de l'expression.

Exemple : ainsi pour rechercher dans un fichier "`fichier`", toutes les lignes contenant au moins un des caractères de l'ensemble `A`, `E`, `F` ou `K`, on aura :

```
$ grep [AEFK] fichier%
```

Il est possible de combiner les diverses possibilités des expressions régulières entre elles. Ainsi pour définir toutes les lignes commençant par un des caractères `A`, `E`, `F` ou `K`, on utilisera l'expression régulière suivante "`^[AEFK]`", respectivement "`[AEFK]$`" pour trouver toutes les lignes qui finissent par ces mêmes lettres.
Les ensembles de caractères peuvent être spécifiés sous la forme d'intervalle. Par exemple `[a-z]` signifie toutes les lettres de `a` à `z`.
De même les classes de caractères peuvent être combinées entre elles. Ainsi il est possible de rechercher toutes les suites de caractères dont le premier est pris dans l'ensemble `A` à `E`, le second égal à `o`, `i` ou `u`, suivi de trois

caractères quelconques, suivi d'un `b` ou d'un `c` ou d'une lettre comprise entre `i` et `m` par l'expression suivante : `[A-E][oiu]...[bci-m]`.

Exercice : Écrire la commande qui permet de recherche les lignes ne débutant pas par `S`, `H` ou `J` dans un fichier `fichier`.

Réponse :

Exercice : Écrire la commande qui permet de rechercher les lignes commençant par `P`, suivi de `a` ou `i`, puis deux lettres quelconques, et ayant pour cinquième lettre un `r` ou une `a` dans un fichier `fichier`.

Réponse :

7.1.2.2 Les accolades et les répétitions d'ensembles

Les accolades `{}` offrent la possibilité de spécifier des répétitions. Un nombre entre accolades suivant une expression régulière indique le nombre de répétitions que l'on souhaite appliquer à cette expression. Les accolades doivent être introduite par des antislash `\`. Par exemple pour représenter toutes les suites de caractères commençant par une lettre majuscule, suivie de 5 lettres minuscules et suivie de 2 chiifres entre 0 et 9 nous avons : `[A-Z][a-z]\{5\}[0-9]\{2\}`.
Par exemple, pour rechercher dans un fichier `F1` des suites de 8 lettres commençant par `T` ou `t`, on écrira : `grep '[Tt]\{7\}' F1`.
En règle générale, on précise le nombre minimum et maximum de répétitions `\{p,q\}`. Il existe de plus 3 caractères particuliers pour exprimer des répétitions :
– `*` est équivalent à `\{0,\}` : le minimum est zéro et le maximum est infini.
– `+` est équivalent à `\{1,\}` : le minimum est un et le maximum est infini.
– `?` est équivalent à `\{0,1\}` : l'expression est répétée une fois au plus.

Exercice : Écrire la commande qui permet de rechercher dans un fichier `fichier` des lignes commençant par `B` ou `C` et se terminant par les chiffres `2`, `4` ou `6`.

Réponse :

Exercice : Écrire la commande qui permet de rechercher dans un fichier `fichier` des lignes dont le second caractère est commençant par `r` ou `e` et l'avant dernier caractère est le chiffre `4`.

Réponse :

Exercice : Écrire la commande qui permet de rechercher dans un fichier `fichier` des mots d'au moins 6 lettres commençant par un `P`.

Réponse :

Exercice : Écrire la commande qui permet de rechercher dans un fichier `fichier` toutes les lignes pour lesquelles le mot comporte 5 lettres..

Réponse :

7.1.2.3 La spécification de mot

La spécification de mot dans une expression régulière est réalisée en englobant l'expression recherchée par `\<` et `\>`. Ceci permet de considéré l'expression comme un mot séparé du reste de la ligne par un séparateur.
Par exemple pour chercher le mot `Martin` dans le fichier `fichier` en début de ligne, on aura : `grep '^\<Martin\>'`.

7.1.3 Le filtre identité : `cat`

La syntaxe de `cat` est `cat` [*liste de fichiers*]
`cat` affiche sur la sortie standard ce qui est lu sur l'entrée standard, ou successivement les fichiers de la liste.
En redirigeant la sortie standard sur un fichier, il est possible de concaténer un ensemble des fichiers.
L'option `-v` permet de visualiser les caractères non imprimables.

Exemple :

```
$cat chrs
a b e
âzê û î
$cat -v chrs
a ^A b ^B e ^E
M-bzM-j M-{ M-n
```

7.1.4 Ligne, Mot et Caractère : `wc`

Le filtre `wc` compte le nombre de lignes, de mots et de caractères sur l'entrée standard ou dans une liste de fichiers donnés en argument et écrit ces nombres sur la sortie standard.

7.1.5 De la tête à la queue : `head` et `tail`

La syntaxe de `head` est `head` [*-nombre*] [*-liste-de-fichiers*]
`head` écrit sur la sortie standard les *nombre* premières lignes (par défaut les 10 premières) lues sur l'entrée standard ou dans chacun des fichiers donnés en argument.

Exemple :

```
$ head -2 prenoms
David
Francis
$ head -2 prenoms seneque
==> prenoms <==
David
Francis
==> seneque <==
Paucis natus est, qui populum aetatis suae cogitat.
Seneque.
$
```

La syntaxe de `tail` est `tail` [*-+nombre*] [*-liste-de-fichiers*]
`tail` écrit sur la sortie standard les lignes de chaque fichiers donnés en argument et situées à partir de *nombre* lignes comptées à partir du début (respectivement de la fin) si l'on choisi l'option `+` (respectivement l'option `-`).
Par défaut *nombre* vaut 10.

Exemple :

```
$cat villes
Paris
Londres
Rome
Jerusalem
$ tail -2 villes
Rome
Jerusalem
$ tail +3 villes
Rome
Jerusalem
$
```

7.1.6 Caractère pour caractère : `tr`

La syntaxe de `tr` est `tr` [*chaîne*$_1$] [*chaîne*$_2$]
Dans l'emploi courant de `tr`, les deux *chaînes* ont le même nombre de lettre et les lettres de *chaîne*$_1$ sont distinctes. L'entrée standard est recopiée sur la sortie standard, chaque lettre de *chaîne*$_1$ est remplacée par la lettre correspondante de *chaîne*$_2$.

Exemple :

```
$ tr "[a-z]" "[A-Z]" < prenoms
DAVID
FRANCIS
PHILIPPE
$
```

Notons que l'option `-d` utilisée dans `tr -d` *chaîne* a pour effet d'éliminer tous les caractères de *chaîne*.

Exemple :

```
$ tr -d "[A-Z]" < prenoms
avid
rancis
hilippe
$
```

7.1.7 Un peu d'ordre : `sort`

La commande `sort` est une commande standard d'Unix qui permet d'ordonner des informations.
Cette commande permet de trier un fichier suivant différents critères, mais elle permet aussi de réaliser la fusion, le tri de plusieurs fichiers en un seul. La commande `sort` lit son entrée sur l'entrée standard et écrit sa sortie sur la sortie standard. Il sera donc nécessaire d'utiliser les redirections.
La syntaxe de la commande `sort` est la suivante :
`sort [options]` *fichiers*
Les options permettent de modifier le comportement ou les clés de tri utilisées par la commande. Les options les plus courantes sont les suivantes :
– `-b` saute les espaces en tête de ligne.
– `-td` utilise la lettre `d` comme séparateur de champs.
– `-d` effectue le tri dans l'ordre lexicographique sans tenir compte des caractères spéciaux et de ponctuation.
– `-f` ignore les différences entre majuscules et minuscules.
– `-M` effectue une comparaison chronologique.
– `-r` tri dans l'ordre inverse.
Ces options constituent des règles de tri qui sont appliquées globalement pour la ou les clés de tri utilisées.
La notion de clé de tri implique la notion de champ. Un champ est une suite de caractères délimitée par un séparateur de champs ou un saut le ligne. Le séparateur peut être défini par l'option `-t` comme vu plus haut.
La définition d'une clé de tri se fait par la notation *+p1 -p2*. La valeur *p1* spécifie le début de la clé de tri et la valuer *p2* spécifie la fin de la clé. Si la valeur *p2* est absente, la clé de tri est définie de *p1* jusqu'à la fin de la ligne. Les expressions de *p1* et *p2* sont des quantités de la forme : *m[.n]* ou *m* représente la position du champ et *n* représente le caractère du début du champ. Il est important de remarquer que les champs et les caractères d'un champ sont numérotés à partir de 0. Par exemple la spécification *2.1* correspond au second caractère du troisième champ.

Exemple : Soit le fichier *fitest* suivant :

```
$ cat fitest
Orange      18      124
Banane      23      98
Citron      12      112
Pamplemouse 17      65
Mangue      24      33
Goyave      28      12
Pomme       12      148
Poire       15      122
....
```

Le tri de ce fichier sans aucune option donne le résultat suivant :

```
$ sort fitest
Abricot     23      45
Ananas      18      24
Banane      23      98
Brugnon     22      32
Carotte     33      55
Cerise      23      46
Citron      12      112
Fraise      21      43
Goyave      28      12
Mandarine   15      156
Mangue      24      33
....
```

Si l'on désire trier le fichier sur tous les caractères de la ligne à partir du second champ on aura :

```
$ sort +1 fitest
Citron      12      112
Pomme       12      148
Mandarine   15      156
Poire       15      122
Raisin      17      87
Orange      18      124
Ananas      18      24
Peches      18      33
Fraise      21      43
Brugnon     22      32
Banane      23      98
Carotte     33      55
Abricot     23      45
Cerise      23      46
....
```

Il est possible de faire le tri sur le troisième champ. Et on remarque que les données ne sont pas triées suivant un critère numérique, mais suivant un critère de type de caractère.

```
$ sort +2 fitest
Citron      12      112
Goyave      28      12
Poire       15      122
Orange      18      124
Pomme       12      148
Mandarine   15      156
Mandarine   15      156
Raisin      17      87
Ananas      18      24
Brugnon     22      32
....
```

Afin de retrouver un ordre de tri numérique, il faut préciser l'option `-n` dans la commande de tri.

```
$ sort -n +2 fitest
Goyave      28      12
Ananas      18      24
Brugnon     22      32
Mangue      24      33
Fraise      21      43
Abricot     23      45
Carotte     33      55
....
```

La commande `sort` permet aussi de trier et de fusionner plusieurs fichiers en un.

```
$head fictest > f1
$tail fictest >f2
$ sort f1 f2
Abricot     23      45
Ananas      18      24
Banane      23      98
Brugnon     22      32
Carotte     33      55
Fraise      21      43
Mangue      24      33
....
```

Exercice : Écrire la commande qui permet de trier le fichier `fichier` où les champs sont séparés pas des " :", sur les champs 3 et 4 en ordre inverse.

Réponse :

7.2 Le filtre-éditeur : `sed`

La commande `sed` est une commande puissante qui peut être considérée comme un filtre et comme un éditeur. Son utilisation la plus courante est de recevoir en entrée successivement chaque ligne d'un fichier, lui faire subir éventuellement des modifications et l'envoyer sur la sortie standard.

7.2.1 Utilisation courante de `sed`

Les commandes les plus utilisées sont les commandes de substitution (`s`) et de suppression (`d`).
La commande la plus simple de modification est la substitution `s` sous la forme :
`sed s/`*motif*`/`*chaine*`/` *fichier*
où *motif* est une expression régulière dont la syntaxe sera précisée. Dans notre cas le *motif* est une simple chaîne de caractères.

Exemple : soit le fichier `toufo` qui contient des erreurs.

```
$cat toufo
pierre qui roule n'a masse pas mousse
dabo dabon dabonnet
2 peste soit des avatars et des avares au cieux

Sacha chassa son chat, Sancho secha ses choux;
```

La commande suivante permet de corrigé la chaîne `n'a masse` en la chaîne `n'amasse`.

```
$sed s/"n'a masse"/"n'amasse"/ toufo > toufo.1
$cat toufo.1
pierre qui roule n'amasse pas mousse
dabo dabon dabonnet
2 peste soit des avatars et des avares au cieux

Sacha chassa son chat, Sancho secha ses choux;
```

Essayons de corriger la seconde ligne.

```
$sed s/dabo/dubo/ toufo.1
$cat toufo.1
pierre qui roule n'amasse pas mousse
dubo dabon dabonnet
2 peste soit des avatars et des avares au cieux

Sacha chassa son chat, Sancho secha ses choux;
```

On remarque que seule la première occurence de `dabo` a été corrigée. Il faut demander explicitement une substitution "globale" en ajoutant le *drapeau* `g`.

```
$sed s/dabo/dubo/g toufo.1 > toufo.2
$cat toufo.2
pierre qui roule n'amasse pas mousse
dubo dubon dubonnet
2 peste soit des avatars et des avares au cieux

Sacha chassa son chat, Sancho secha ses choux;
```

Supprimons maintenant le `2` suivi de l'espace. La commande suivante supprime tout chiffre en début de ligne suivi de zéro ou plusieurs espaces :

```
$sed 's/^[0-9]*[ ]*//' toufo.2 > toutfo.3
$cat toufo.3
pierre qui roule n'amasse pas mousse
dubo dubon dubonnet
peste soit des avatars et des avares au cieux

Sacha chassa son chat, Sancho secha ses choux;
```

Ajoutons maintenant un point-virgule à chaque fin de ligne qui se termine par une lettre.

```
$sed 's/[a-zA-Z]$/&;/' toufo.3 > toutfo.4
$cat toufo.4
pierre qui roule n'amasse pas mousse;
dubo dubon dubonnet;
peste soit des avatars et des avares au cieux;

Sacha chassa son chat, Sancho secha ses choux;
```

Le caractère `&` permet de désigner dans la partie remplacement le motif capté dans la partie initiale.
La commande de suppression `d` s'utilise de la manière suivante : `sed /`*motif*`/d` *fichier*
où *motif* est une expression régulière dont la syntaxe sera précisée. Dans notre cas le *motif* est une simple chaîne de caractères.

Exercice : Écrire la commande qui permet de remplacer le point-virgule de la dernière ligne par un point et d'écrire le résultat dans le fichier `toufo.5`.

Réponse :

Exercice : Écrire la commande qui permet de supprimer toutes les lignes vides du fichier `toufo.5`.

Réponse :

Exercice : Écrire la commande qui permet d'insérer 2 espaces en début de chaque ligne. On utilisera le fait que, dans un motif, le point "`.`" est un métacaractère désignant n'importe quel caractère autre que *newline*.

Réponse :

Lorsque de nombreuses commandes `sed` sont nécessaires, il devient intéressant de les réunir dans un fichier et de les faire exécuter grâce à la commande :
`sed -f` *fichier_de_commande fichier*

Exercice : Réunir dans un fichier `script.sed` les commandes déjà utilisée et ajouter celle qui est nécessaire pour compléter la correction du fichier `toutfo`.

Réponse :

7.2.2 Les commandes de `sed`

Outre les commandes de substitution `s` et de suppression `d`, il existe d'autres commandes disponibles.

Ajouter : *[adresse]*`a\` suivi d'un texte sur la ligne suivante. Ajoute le texte après l'*adresse* indiquée.

Exemple :

```
$cat seneque
Paucis natus est, qui populum aetatis suae cogitat.
Seneque.
$cat script1
a\
Il est ne pour peu d'hommes, celui qui n'a en tete\
que les gens de son siècle.
$sed -f script1 seneque
$cat seneque
Paucis natus est, qui populum aetatis suae cogitat.
Seneque.
Il est ne pour peu d'hommes, celui qui n'a en tete\
que les gens de son siècle.
$
```

Insérer : *[adresse]*`i\` suivi d'un texte sur la ligne suivante. Insére le texte avant l'*adresse* indiquée.

Exemple :

```
$cat prenoms
David
Francis
Philippe
$cat script2
/Phi/i\
Marcel\
Paul
$sed -f script2 prenoms
$cat prenoms
David
Francis
Marcel
Paul
Philippe
$
```

Changer : *[adresse]*`c\` suivi d'un texte sur la ligne suivante. Remplace la ligne qui se trouve à l'*adresse* par le texte.

Exemple :

```
$cat script3
/Francis/c\
Francois\
Maurice
$sed -f script3 prenoms
$cat prenoms
David
Francois
Marcel
Paul
Philippe
$
```

Afficher l'espace de travail : `l`
La commande `l` affiche l'*espace de travail* de `sed`, c'est à dire, à chaque étape, le contenu de la ligne sur laquelle vont être effectués les modifications éventuelles. Cette commande permet notamment de visualiser les caractères non imprimables affichés par leurs codes ASCII.

Coder des caractères : `y`
La commande `y` permet de remplacer un caractère par un autre :
`y/abc/uvw/`
remplace `a` par `u`, `b` par `v` et `c` par `w`.

```
$cat Saintex
Car j'ai vu trop souvent le pitie s'egarer.
saint-exupery.
$sed 2y/aeinprstuxy/AEINPRSTUXY/ Saintex
Car j'ai vu trop souvent le pitie s'egarer.
SAINT-EXUPERY.
$
```

Afficher les numéros de lignes : *[adresse]*=
La commande = affiche les numéros de lignes qui se trouvent à l'*adresse*.

```
$sed  /p/= toutbon
1
  pierre qui roule n'amasse pas mousse;
  dubo dubon dubonnet;
3
  peste soit des avatars et des avaricieux;
  Sacha chassa son chat, Sancho secha ses choux.
$
```

Suppression de l'envoi sur la sortie standard : `-n`

Sauvegarder l'espace de travail : *[adresse]*`w` *fichier*
La commande `w` sauvegarde dans le *fichier* les lignes qui se trouvent à l'*adresse*.

```
$sed -n '/p/w toutou' toutbon
  pierre qui roule n'amasse pas mousse;
  peste soit des avatars et des avaricieux;
$
```

Quitter : *[adresse]*`q`
Stoppe le traitement d'un fichier dès que l'adresse a été atteinte.

```
$sed /F/q prenoms
David
Francis
$
```

Exercice : Le petit script que vous allez devoir écrire sera fort utile à ceux qui utilisent conjointement Linux et Windows, et qui doivent échanger des fichiers texte entre ces systèmes. En effet Linux indique les fins de ligne avec un seul caractère (`\n`), alors que windows en utilise 2 (`\r\n`). Les caractères (`\r`) sont souvent affichés sous Linux comme des (`^M`). Il vous faut donc écrire le petit script qui va permettre d'enlever les `^M` à chaque fin de ligne dans une série de fichiers.
Vous constaterez que certains caractères de contrôle sont utilisés dont le caractère de contrôle `^[`, qui représente le caractère d'échappement. Ces caractères sont représentés avec la notation classique `^C`, où C est la lettre à utiliser avec la touche CTRL pour obtenir ce caractère. Ces notations ne font que représenter les caractères de contrôle, et doivent être remplacées par les caractères qu'elles représentent dans le fichier de configuration. Pour saisir ces caractères spéciaux, vous devrez passer en mode insertion dans vi, puis utiliser la combinaison de touches CTRL+V. Ce raccourci permet d'indiquer à vi qu'il doit insérer les codes d'échappement directement issus du clavier, sans les interpréter. Vous pourrez alors taper la séquence de touches générant le caractère de contrôle ou la séquence d'échappement désirée. Par exemple, vous pourrez obtenir le caractère `^H` en tapant la combinaison de touches CTRL+H, et le caractère `^?` en appuyant sur la touche Backspace (retour arrière). Les caractères d'échappement peuvent être générés par la touche Echap ou directement par les touches du curseur.

Note : En fait, vous pouvez également utiliser les chaînes de caractères \e et <Esc> pour représenter le caractère d'échappement. Mais certaines options ne fonctionnent pas avec ces notations, et je vous les déconseille.

Réponse :

Cette page est laissée blanche intentionnellement

Chapitre 8

Unix Avancé : `find` et `cron`

8.1 Trouver ses petits : `find`

`find` parcoure récursivement l'arborescence en sélectionnant des fichiers selon des critères de recherche, et exécute des actions sur chaque fichier sélectionné.

8.1.1 Syntaxe générale

`find` *repertoire_de_depart critere_de_recherche action_a_executer*

Exemple : `$ find $HOME -print`
Cette commande va parcourir toute l'arborescence à partir du répertoire de login `$HOME`, va sélectionner tous les fichiers puisqu'il n'y a aucun critère de recherche et va afficher le nom de chaque fichier trouvé (les répertoires aussi).

8.1.2 Les critères de recherche

`-name` ***modele*** sélectionne uniquement les fichiers dont le nom correspond au modèle.
Attention ! Le modèle doit être interprété par la commande `find` et non par le shell, s'il contient des caractères spéciaux pour le shell (par exemple `*`), ceux-ci doivent être protégés.

Exemple : `$ find /usr/utilisateur -name '*.c' -print`
L'option `-name` n'accepte qu'un argument. S'il y en a plusieurs il faut rajouter `-name` pour chacun.
De même il ne faut pas oublier de mettre une action (par exemple `-print`), sinon la recherche s'effectuera mais il ne se passera rien.

`-perm` ***nombre_octal*** sélectionne les fichiers dont les droits d'accès sont ceux indiqués par le nombre octal.

Exemple : `$ find /usr/utilisateur -perm 0777 -print`
affiche tous les fichiers qui sont autorisés en lecture, écriture et exécution pour l'utilisateur propriétaire, les personnes du groupe et tous les autres.

`-type` ***caractere*** sélectionne les fichiers dont le type est celui indiqué par le caractère. C'est à dire :
- `c` pour un fichier spécial en mode caractère,
- `b` pour un fichier spécial en mode bloc,
- `d` pour un répertoire,
- `f` pour un fichier normal,
- `l` pour un lien symbolique.

Exemple : `$ find /usr/utilisateurutilisateur -type d -print`
affiche tous les répertoires et sous-répertoires de `/usr/utilisateur`.

`-links` ***nombre_décimal*** sélectionne les fichiers dont le nombre de liens est donné par le nombre décimal. Si le nombre est précédé d'un `+` (d'un `-`) cela signifie supérieur (inférieur) à ce nombre.

Exemple : `$ find /usr/utilisateur -links +2 -print`
affiche tous les fichiers qui ont plus de 2 liens.

`-user` ***n[ou]m_utilisateur*** sélectionne les fichiers dont l'utilisateur propriétaire est le *nom_utilisateur* ou dont la numéro d'utilisateur (UID) est *num_utilisateur*.

Exemple : `$ find /dev -user utilisateur -print`
affiche tous les fichiers spéciaux qui appartiennent à ***utilisateur***.

`-size` ***nombre_décimal[c]*** sélectionne les fichiers dont la taille est de *nombre_decimal* blocs. Si on post-fixe le *nombre_decimal* par le caractère `c`, alors la taille sera donnée en nombre de caractère.

`-inum` ***nombre_décimal*** sélectionne les fichiers dont le numéro d'I-node est *nombre_decimal*.

`-atime` ***nombre_décimal*** sélectionne les fichiers qui ont été accèdés dans les *nombre_decimal* derniers jours.

Exemple : `$ find /usr/utilisateur -atime -2 -print`
affiche tous les fichiers qui ont été accèdés dans les 2 derniers jours.

`-mtime` ***nombre_décimal*** sélectionne les fichiers qui ont été modifiés dans les *nombre_decimal* derniers jours.

`-newer` ***fichier*** sélectionne les fichiers qui sont plus récents que celui passé en argument.

8.1.3 Combinaison de critères

Plusieurs critères peuvent être groupés par les opérateurs `(` et `)`. Comme ces deux opérateurs sont des caractères spéciaux, ils doivent être despécialisés.
Si plusieurs critères sont mis à la suite, `find` sélectionne les fichiers qui répondent à tous les critères. Le "ET logique" est donc implicite.

Exemple : `$ find /usr/utilisateur \( -name '*.c' -mtime -3 \) -print`
affiche les fichiers se terminant par "`*.c`" et modifiées dans les 3 derniers jours.
Le "OU logique" est représenté par l'opérateur `-o`.

Exemple : `$ find /usr/utilisateur \( -name '*.txt' -o -name '*.doc' \) -print`
affiche tous les fichiers se terminant par "`*.txt`" ou "`*.doc`".
Le "NON logique" est représenté par l'opérateur `!`.

Exemple : `$ find /usr/utilisateur ! -user utilisateur -print`
affiche tous les fichiers n'appartenant pas à ***utilisateur***.

8.1.4 Les actions possibles sur les noms de fichiers

`-print` affiche le nom des fichiers sélectionnés sur la sortie standard.

`-exec` ***commande*****`\;`** exécute *commande* sur tous les fichiers sélectionnés. Dans la commande Shell, `{}` sera remplacé par les noms des fichiers sélectionnés.

Exemple : `$ find /usr/utilisateur -name '*.txt' -exec lp {} \;`
envoie à l'impression tous les fichiers se terminant par `.txt` dans l'arborescence `/usr/utilisateur`.

`-ok` ***commande*****`\;`** même chose que `-exec`, mais demande confirmation avant chaque exécution.

Exemple : `$ find /usr/c1 -name '*.sav' -ok lp {} \;`

```
lp /usr/utilisateur/jour.sav ? y
lp /usr/utilisateur/nuit.sav ? y
lp /usr/utilisateur/essai.sav ? n
lp /usr/utilisateur/toto.sav ? n
```

imprime (ou non) chaque fichier `.sav` dans l'arborescence `/usr/utilisateur` après avoir demandé confirmation.

Exercice : Dans les deux commandes suivantes, la première fonctionne, mais pas la suivante. Pourquoi ?

```
find /usr/utilisateur \( -name '*.c' -mtime -3 \) -print
find '/usr/utilisateur \( -name *.c -mtime -3 \) -print'
```

Réponse :

Exercice : Dans l'arborescence complète, afficher toutes les informations de tous les fichiers dont vous êtes propriétaire.
Réponse :

Exercice : Cherchez dans toute l'arborescence les fichiers dont le nom se termine par `.c`, redirigez les erreurs vers le fichier poubelle `/dev/null`, Commençant par X ou x, dont les noms ne contiennent pas de chiffre.
Réponse :

Exercice : Chercher dans `/usr` les fichiers dont la taille dépasse 1Mo (2000 blocs de 500Ko) et dont les droits sont fixés à `755 (-rwxr-xr-x)`.
Réponse :

Exercice : Combien il y a de fichiers dans toute l'arborescence vous appartenant et ayant les droits fixés à `666 (-rw-rw-rw-)`.
Réponse :

Exercice : Trouver tous les fichiers `core` dans l'arborescence et supprimez les après confirmation.
Réponse :

8.2 Commandes retardées : `at` et `crontab`

8.2.1 La commande `at`

La commande `at` permet de lancer une commande un jour donné, à une heure donnée. Une fois que cette commande a été exécutée, elle n'existe plus. Pour l'utiliser il suffit de taper `at` puis l'heure :

```
at 12:30
```

déclenchera la commande à 12h30.
La syntaxe des entrées est la suivante :

– `at 12 :30 11/30/01` déclenchera la commande le 30 novembre 2001 à 12h30.
– `at now +1hour` déclenchera la commande dans 1 heure à partir de maintenant.
– `at 00 :00 +2days` pour exécuter la commandes dans 2 jours à minuit.
Mais jusque la, nous n'avons entré aucune commande. Mais lorsque vous tapez `at 12 :30`, vous obtenez l'invite de la commande `at` :

```
$at 12:30
at>ping -c 5 192.168.0.1
at>^D
$
```

Vous entrez donc la commande que vous désirez effectuer et vous obtenez la réponse suivante :

```
job 1 at 2001-11-10 12:30
```

La commande va envoyer un `ping` sur la machine `192.168.0.1` à 12h30. Il est bien sur possible de faire exécuter un script. Ensuite `at` envoie par mail le résultat de cette commande à l'auteur.
Si vous désirez savoir ce qu'il va se passer vous pouvez tester la commande `at -c 1`. Cette option permet de montrer la commande numéro 1.
La commande `atq` permet de lister toutes les commandes `at`, et la commande `atrm` permet de supprimer un des *job* de `at`.

8.2.2 La commande `crontab`

`crontab` est un utilitaire qui permet de programmer des actions régulières la machine.
Un démon nommé `cron` lit le fichier qui se trouve dans le répertoire `/var/spool/cron` (le plus souvent) et exécute les commandes qui s'y trouvent.
Afin de créer ce fichier, on peut utiliser le programme `crontab` avec l'option `-e` qui permet d'éditer le fichier crontab à l'intérieur de `vi`. Il ne reste plus qu'a entrer les commandes en respectant la syntaxe décrite ci-dessous. Après avoir enregistrer votre nouveau fichier, `crontab` vous affiche le message suivant `installing new crontab`. Il est possible de visualiser tous les `crontab` avec l'option `-l`.
La syntaxe `crontab` est la suivante :
`<minute> <heure> <jour_du_mois> <mois> <jour_de_semaine> <commande>`
– `minute` : de 0 à 59,
– `heure` : de 0 à 23,
– `jour_du_mois` : de 1 à 31,
– `mois` : de 1 à 12,
– `jour_de_semaine` : de 0 à 6, 0 étant le dimanche et ainsi de suite,
– `commande` : peut comporter plusieurs commandes.
Les mois et les jours peuvent aussi être donnés avec les abréviations anglaises : jan, feb, ... et mon, tue, ...
De plus, on sépare les jours, les mois par des virgules, donc par exemple, pour exécuter une commande tous les 15 et 30 du mois on aura `15,30`.
Si on sépare par un tiret `-`, il s'agit alors d'un intervalle. Ainsi `15-30` signifie du 15 au 30.
Le `/` permet de spécifier une répétition. Ainsi `*/3` indique toutes les 3 minutes. `*` peut aussi être utiliser pour signifier tous les jours de semaines, tous les mois, toutes les heures.

Exemples :
– `0 1 1 * * commande` signifie que la commande sera exécutée le premier jour du mois à 1 heure.
– `0 1 * * mon commande` signifie que la commande sera exécutée un fois par semaine le lundi à 1 heure.
– `0 1 1,15 * * commande` signifie que la commande sera exécutée tous les 1 et 15 du mois à 1 heure.
– `0 1 1-15 * * commande` signifie que la commande sera exécutée tous les 15 premiers jours du mois à 1 heure.
– `0 1 */5 * * commande` signifie que la commande sera exécutée tous les 5 jours à 1 heure.
– `*/3 * * * * commande` signifie que la commande sera exécutée toutes les 3 minutes.
La commande suivante efface tous les jours, les fichiers présents dans le répertoire `/var/log` vieux de plus de 7 jours.

```
0 1 * * * find /var/log -atime 7 -exec rm -f {} \;
```

8.2.3 Contrôle

Dans le cas de `at` ou `crontab`, il est possible de définir qui a le droit d'utiliser ces commandes. Pour cela il existe les fichiers `/etc/cron.allow` et `/etc/cron.deny` et `/etc/at.allow` et `/etc/at.deny`. Par exemple, pour interdire l'utilisation de la commande `cron` à certains utilisateurs, il suffit d'entrer leurs noms dans le fichier `cron.deny`.

Exercice : Créer un crontab qui sauvegarde toutes les nuits dans le répertoire `/var/sauv` sous la forme d'un fichier `tar` compressé, votre répertoire `home`.

Réponse :

Cette page est laissée blanche intentionnellement

Annexe A

Récapitulatif des commandes de Vim

A.1 Déplacement et insertion de text

1. Le curseur se déplace avec les touches fléchées ou les touches `hjkl`.
   `h (gauche)  j (bas)  k (haut)  l (droite)`
2. Pour entrer dans Vim (à l'invite %) tapez : `vim FICHIER <Entrée>`
3. Pour quitter Vim tapez : `<Échap> :q! <Entrée>` pour perdre tous les changements OU tapez : `<Échap> :wq <Entrée>` pour enregistrer les changements.
4. Pour effacer un caractère sous le curseur en mode Normal tapez : `x`
5. Pour insérer du texte au niveau du curseur en mode Normal tapez :
   `i tapez le texte <Échap>`

NOTE : Appuyer <Échap> vous place en mode Normal ou annule une commande partiellement tapée dont vous ne voudriez plus.

A.2 Effacement de mot

1. Pour effacer du curseur jusqu'à la fin d'un mot tapez : `dw`
2. Pour effacer du curseur jusqu'à la fin d'une ligne tapez : `d$`
3. Pour effacer toute une ligne tapez : `dd`
4. Le format d'une commande en mode Normal est :

   `[nombre] commande objet`

   où

   `commande [nombre] objet`

   où :
   (a) nombre - est combien de fois répéter la commande
   (b) commande - est ce qu'il faut faire, par exemple `d` pour effacer
   (c) objet - est ce sur quoi la commande devrait agir, par exemple `w` (mot), `$` (jusqu'à la fin de la ligne), etc.
5. Pour annuler des actions précédentes, tapez : `u` (u minuscule)
6. Pour annuler tous les changements sur une ligne tapez : `U` (U majuscule)
7. Pour annuler l'annulation tapez : `Ctrl-R`

A.3 Placer, remplacer du texte

1. Pour remettre du texte qui vient d'être effacé, tapez `p`. Cela Place le texte effacé APRÈS le curseur (si une ligne complète a été effacée, elle sera placée sous la ligne du curseur).
2. Pour remplacer le caractère sous le curseur, tapez `r` suivi du caractère qui remplacera l'original.
3. Le changement vous permet de changer l'objet spécifié, du curseur jusqu'à la fin de l'objet. Par exemple, tapez `cw` pour changer du curseur jusqu'à la fin du mot, `c$` pour changer jusqu'à la fin d'une ligne.
4. Le format pour le changement est :

   `[nombre] c objet`

   où

   `c [nombre] objet`

A.4 Copier et Coller du texte

A.4.1 Copier

1. `["x]y{deplacement}` Copie le texte défini par le deplacement (commande qui déplace le curseur `w` pour un mot, `4j` pour 4 lignes vers le bas, `/le` jusquà la prochaine occurence de ce mot) [dans la mémoire x].
2. `["x][nombre]yy` Copie le `nombre` de ligne `dans la mémoire x`
3. `["x]Y` idem `yy`

Par exemple, en mode commande, `y3w` copie dans le buffer par défaut les trois mots suivants le curseur, `5yy` ou `5Y` copie les 5 lignes suivantes dans le buffer à partir de la position du curseur, `yy` copie dans le buffer la ligne courante.
Si votre fichier `.vimrc` contient la ligne `set mouse=a`, vous avez alors la possiblilité d'utiliser la souris pour réaliser la copie et le collage de texte.

1. `{Visuel}["x]y` Copie le texte en surbrillance [dans le buffer x]

A.4.2 Coller

1. `["x][nombre]p` Colle le texte du buffer `x` un nombre de fois après le curseur
2. `["x][nombre]P` Colle le texte du buffer `x` un nombre de fois après le curseur
3. `["x][nombre]<MiddleMouse>` Colle le texte du buffer `x` un nombre de fois après le curseur. Ceci ne fonctionne que si les mode "mouse" n ou a sont activés.

A.5 Positionnement, recherche dans le fichier

1. `Ctrl-G` affiche votre position dans le fichier et l'état de celui-ci. `Maj-G` vous place à la fin du fichier. Un numéro de ligne suivi de `Maj-G` vous place à cette ligne.
2. Taper `/` suivi d'un texte recherche ce texte vers l'AVANT. Taper `?` suivi d'un texte recherche ce texte vers l'ARRIÈRE. Après une recherche tapez `n` pour trouver l'occurence suivante dans la même direction ou `Maj-N` pour rechercher dans la direction opposée.
3. Taper `%` lorsque le curseur est sur `(`, `)`, `[`, `]`, `{` ou `}` déplace celui-ci sur le caractère correspondant.
4. Pour remplacer le premier `aa` par `bb` sur une ligne tapez `:s/aa/bb`
5. Pour remplacer tous les `aa` par `bb` sur une ligne tapez `:s/aa/bb/g`
6. Pour remplacer du texte entre deux numéros de ligne tapez `:#,#s/aa/bb/g`
7. Pour remplacer toutes les occurences dans le fichier tapez `:%s/aa/bb/g`
8. Pour demander une confirmation à chaque fois ajoutez 'c' `:%s/aa/bb/gc`

A.6 Exécuter des commandes, enregistrer des fichiers

1. `:!commande` exécute une commande externe.
   Quelques exemples pratiques :
   (a) `:!dir` affiche le contenu du dossier courant.
   (b) `:!del FICHIER` efface FICHIER.
   Les mêmes exemples, pour les systèmes Unix :
   (a) `:!ls` affiche le contenu du dossier courant.
   (b) `:!rm FICHIER` efface FICHIER.
2. `:w FICHIER` enregistre le fichier Vim courant sur le disque avec pour nom `FICHIER`.
3. `:#,#w FICHIER` enregistre les lignes # à # dans le fichier FICHIER.
4. `:r FICHIER` récupère le fichier FICHIER et l'insère dans le fichier courant à partir de la position du curseur.

A.7 Les modes de Vim

1. Taper `o` ouvre une ligne SOUS le curseur et y place celui-ci en mode Insertion. Taper un `O` majuscule ouvre une ligne au DESSUS de la ligne où se trouve le curseur.
2. Tapez un `a` pour insérer du texte APRÈS le caractère où se trouve le curseur. Taper un `A` majuscule ajoute du texte automatiquement à la fin de la ligne.

3. Taper un `R` majuscule active le mode Remplacement jusqu'à ce que la touche `<Échap>` soit appuyée pour en sortir.
4. Taper `:set xxx` active l'option '`xxx`'.

A.8 Utiliser le système d'aide en ligne

Vim a un système complet d'aide en ligne. Pour y accéder, essayez l'une de ces trois méthodes :

1. appuyez la touche `<Help>` (si vous en avez une)
2. appuyez la touche `<F1>` (si vous en avez une)
3. tapez `:help <Entrée>`

Tapez `:q <Entrée>` pour fermer la fenêtre d'aide.
Vous pouvez accéder à l'aide sur à peu près n'importe quel sujet en donnant des arguments à la commande :help . Essayez par exemple (n'oubliez pas d'appuyer sur <Entrée>) :

1. `:help w`
2. `:help c_<T`
3. `:help insert-index`
4. `:help user-manual`

A.9 Activer les fonctionnalités de Vim

Vim a beaucoup plus de fonctionnalités que Vi, mais la plupart de celles-ci sont désactivées par défaut. Pour commencer à les utiliser, vous devez créer un fichier "vimrc".

1. Commencez à éditer le fichier "`vimrc`". Ceci dépend de votre système :

   ```
   :edit ~/.vimrc pour Unix
   :edit $VIM/_vimrc pour MS-Windows
   ```

2. Intégrez maintenant le texte du fichier "`vimrc`" d'exemple :

   ```
   :read $VIMRUNTIME/vimrc_example.vim
   ```

3. Enregistrez le fichier avec :

   ```
   :write
   ```

La prochaine fois que vous démarrerez Vim, le surlignage syntactique sera activé. Vous pouvez ajouter tous vos réglages préférés dans ce fichier.

Cette page est laissée blanche intentionnellement

Bibliographie

[Agix, 1995] Axis & Agix. *Unix Utilisation : Guide de Formation*. EDITIONS LASER, 1995.

[Blaess, 2002a] CH. Blaess. *Langages de scripts sous Linux*. Eyrolles, 2002.

[Blaess, 2002b] CH. Blaess. *Programmation Système en C sous Linux*. Eyrolles, 2002.

[Champarnaud et Hansel, 1995] J.M. Champarnaud et G. Hansel. *Passeport pour UNIX et C*. Passeport pour l'informatique. Internationnal Thomson Publishing, 1995.

[Charman, 1994] P. Charman. *Unix et X-Window : Guide Pratique*. CÉPADUÈS ÉDITIONS, 1994.

[Poulain, 1994] T. Poulain. Cours unix. 1994.

[Stoeckel, 1992] Richard Stoeckel. *Filtres et Utilitaires Unix*. ARMAND COLIN, 1992.