Stage d'initiation à UNIX

Catherine Cassagne

ENSIMAG

Bernard Cassagne

Laboratoire de Génie Informatique

IMAG

GRENOBLE

Version ENSIMAG

septembre 1993

Chapter 1

Introduction

1.1 Bienvenue

Tout d'abord:

Bonjour, et bienvenue à l'ENSIMAG !

1.2 Au sujet du stage

Le stage que vous suivez ici à pour but de vous familiariser avec l'environnement de travail à l'ENSIMAG.

Cette brochure offre un certain nombre d'explications et des suggestions de manipulations. Nous vous demandons de lire la brochure et de réaliser les manipulations. Si vous ne réussissez pas ou si vous ne comprenez pas quelque chose, ne passez pas outre, demandez de l'aide à la personne qui vous encadre. A la fin de ce stage, vous saurez vous connecter à un ordinateur via le réseau, créer et gérer des fichiers, créer et exécuter des programmes.

1.3 Notations

Quand on parle d'un langage, se pose le problème de déterminer ce qui est langage et ce qui est méta-langage. En d'autres termes, quand on dit: écrivez votre nom, faut-il écrire "votre nom" ou faut-il écrire "Pierre Durand"? Nous lèverons cette ambiguïté à l'aide des notations suivantes:

1. ce qui est écrit avec une fonte à largeur constante `comme ceci` fait partie du langage.

2. ce qui est écrit en italique *comme ceci* est du méta-langage.

Exemples

Quand on dit: taper *Return*, cela ne signifie pas taper R,e,t,u,r,n mais taper sur **la** touche marquée Return.

Quand on dit: taper `rm` *nom-de-fichier*, `rm` doit être tapé tel quel, mais *nom-de-fichier* doit être remplacé par un nom de votre choix.

Chapter 2

La connexion à un ordinateur

Le but de ce chapitre est de faire connaissance avec les terminaux et d'apprendre la procédure de connexion à un ordinateur. Cette procédure est dépendante de la manière dont les machines sont reliées entre elles ; celle qui vous est présentée ici est donc particulière à l'ENSIMAG.

2.1 Le terminal

L'appareil qui se trouve devant vous est un terminal, ce **n'est pas** un ordinateur. Un terminal est un instrument de communication permettant à un être humain de communiquer avec une machine. Terminal et machine sont reliés par une ligne de communication de type ligne téléphonique. Un terminal se compose d'un clavier et d'un écran ; tout ce que l'utilisateur tape sur le clavier est envoyé à la machine ; tout ce que la machine envoie sur la ligne, est affiché à l'écran.

Observons le clavier ; il comporte des touches du style de celles d'une machine à écrire : on y voit des lettres, des chiffres et un certain nombre de signes typographiques. En informatique, on désigne sous le terme générique de *caractère* l'ensemble des lettres, chiffres et signes divers. On parle donc du caractère `A`, du caractère `b`, du caractère `+` etc...

Dans un ordinateur, les caractères sont représentés sous forme de nombres. La fonction de correspondance entre un caractère et sa valeur associée, s'appelle un *codage*, et la valeur associée à un caractère s'appelle le *code* du caractère. Il existe deux normes de codage des caractères: le codage EBCDIC et le codage ASCII. A l'ENSIMAG, seul l'ASCII est utilisé. En codage ASCII, le code d'un caractère est un nombre entier compris entre 0 et 127 : les lettres de A à Z ont les codes de 65 à 90, les lettres a à z ont les codes de 97 à 122, les chiffres de 0 à 9 ont les codes de 48 à 57 etc...

2.1.1 Majuscules et minuscules

Comme pour une machine à écrire, les touches comportant une lettre, ont la lettre gravée sous sa forme majuscule, mais génèrent une **minuscule**.

Pour obtenir une lettre majuscule, il faut maintenir appuyée la touche *Shift* et ensuite appuyer sur la lettre désirée. Trouvez sur le clavier la position des touches *Shift* : il y en a deux, une de chaque côté du clavier.

2.1.2 Touches comportant deux gravures

Certaine touches comportent deux gravures. Par exemple, une touche comporte à la fois le caractère `"` et le caractère `'`. Lorsqu'on appuie sur une telle touche, elle génère le caractère correspondant à la gravure du bas (ici le caractère `'`). Pour obtenir le caractère correspondant à la gravure du haut (ici `"`), il faut maintenir appuyée la touche *Shift* et ensuite appuyer sur la touche à double gravure.

2.1.3 Caractères "de contrôle"

Il existe d'autres caractères que ceux qui sont gravés sur le clavier. Le clavier comporte tous les caractères qui disposent d'une représentation graphique, mais il existe des caractères ne disposant pas de représentation graphique : on ne peut donc pas les afficher sur l'écran. Ces caractères sont dits "caractères de contrôles" et on les appelle *Null*, *Control A*, *Control B* etc.. *Control Z*, *Control* `[`, *Control* `\`, *Control* `]`, *Control* `^`, et *Control* `_`. Bien qu'invisibles, ces caractères n'en existent pas moins : ils ont chacun un code dont la valeur va de 0 à 31. Pour les générer, il faut maintenir appuyée la touche *Control* et ensuite appuyez sur une touche alphabétique. Par exemple, maintenir appuyée la touche *Control*, et appuyer sur `D` pour obtenir *Control-D*.

Trouvez sur le clavier la touche *Control* : elle porte la gravure `Ctrl`, et se trouve à gauche du clavier.

Parmi les caractères de contrôle, certains sont si souvent utilisés que les concepteurs de clavier leur allouent une touche particulière, de manière à pouvoir les générer plus commodément qu'en faisant *Control* quelque chose. Il s'agit des caractères suivants :

- *retour chariot* : permet d'aller à la ligne. La touche correspondante se trouve à droite du clavier et porte la gravure "Return". Le caractère *retour chariot* est en fait le caractère *Control M*.

- *Escape* : permet d'indiquer que ce qui est tapé ensuite est spécial. La touche correspondante porte la gravure `F11` (première rangée de touches). Le caractère *escape* est *Control* `[`.

- *Back Space* : permet de retourner en arrière d'un caractère (généralement pour corriger une erreur). La touche correspondante se trouve à droite de la touche `+` `=` au dessus de la touche *Return*. Le caractère *Back Space* est *Control H*.

2.1.4 Le "Set Up"

La touche portant la gravure `F3`, (première rangée de touches) permet de configurer le terminal. La configuration du terminal c'est l'ensemble des paramètres qui régissent son fonctionnement. Si vous modifiez la configuration du terminal vous avez une grande chance que plus rien ne fonctionne. Conclusion : **ne touchez pas à cette touche**.

Par inadvertance j'ai appuyé sur cette touche, que faire? Réappuyez dessus une deuxième fois, tout rentre dans l'ordre.

2.1.5 Mise en route du terminal

Il y a un bouton de mise sous tension à la base du socle, basculez-le pour mettre le terminal sous tension. Au bout de quelques secondes, un petit rectangle de lumière doit apparaître sur l'écran. Ce rectangle de lumière porte le terme technique de “curseur”, c'est ainsi que nous le désignerons par la suite.

Attention

Ça ne marche pas, que faire?

1. vérifier que le cordon d'alimentation du terminal est bien enfiché à la fois dans le terminal et dans la prise murale.

2. Un terminal comporte généralement un bouton permettant de régler la luminosité. Peut être la personne qui vous a précédé sur le terminal a réglé la luminosité à une intensité si faible que le curseur est invisible. Trouvez ce bouton et essayez de faire apparaître le curseur.

Si rien de tout ça ne marche, prévenez le personnel technique que le terminal a un problème.

2.2 Le serveur de terminaux

Quand on a mis le terminal sous tension, le curseur est apparut, puis s'est affiché le texte: `ensitty1`[1]. Si un tel message n'est pas apparu, faites quelques *Retour Chariot* il doit apparaître. Si tel n'est pas le cas, demandez de l'aide.

Il faut savoir que votre terminal n'est pas directement relié à un ordinateur. Il est relié à un serveur de terminaux. Un serveur de terminaux est une machine qui a pour but de créer des communications entre des terminaux et des ordinateurs connectés à un réseau.

Tous les ordinateurs de l'école sont relié par l'intermédiaire d'un réseau. Les serveurs de terminaux sont eux-aussi connectés au réseau. Grâce au réseau, un serveur de terminaux peut créer une communication entre un terminal et n'importe quel ordinateur.

[1] ou `ensitty2` ou `ensitty3`

L'organisation logique du matériel informatique est la suivante :

```
      -------------------------------------------------------------------
      |                     le reseau Ensimag                            |
      ------.----------------.-----------------------------.---------------
            |                |                             |
            |                |                             |
            |                |                             |
           +-+            +-----+                       +-----+
 serveur   | |            |     |                       |     |
   de      | |            | O1  |     etc ...           | On  |
terminaux  | |            |     |                       |     |
           +-+            +-----+                       +-----+
          / | \
         /  |  \
        /   |   \                     ordinateurs
       T    T    T

        terminaux
```

L'organisation physique est plus compliquée, mais n'a pas à être prise en compte par l'utilisateur.

Les serveurs de terminaux de l'ENSIMAG sont tous des *annex* de Encore[2]. Par souci de brièveté, on emploie le terme d' "annex" pour désigner les serveurs de terminaux, et nous utiliserons cette convention dans la suite de cette brochure.

L'école possède 5 annex dont les noms de machines sur le réseau sont respectivement `ensitty1`, `ensitty2` `ensitty3` etc... Le message `ensitty1:` qui s'est affiché sur votre écran est une invitation à taper, ce que l'on appelle un *prompt* dans le jargon informatique. C'est le prompt de l'annex à laquelle votre terminal est connecté.

Tapez :

```
help
```

l'annex doit répondre :

```
                       Annex Command Summary

Name      Syntax                             Description
==========================================================================
?         ? [<command>]                      :Display help on Annex commands
bg        bg [%][%,+,-,<n>,<host>]           :Put a job in the background
fg        fg [%][%,+,-,<n>,<host>]           :Return to an established job
...
rlogin    rlogin <host> [-l <user>]          :Connect to host using RLOGIN protocol
...
who       who [[-l] [<user>]@<host>...]      :Show annex users
```

[2] Constructeur américain d'ordinateurs

L'*annex* a répondu à la commande help en donnant l'ensemble des commandes qu'elle accepte. Parmi toutes ces commandes, nous voyons la commande `rlogin`, c'est celle-là que nous allons utiliser pour obtenir une communication avec un ordinateur. La commande `rlogin` admet en paramètre le nom de la machine à laquelle on désire se connecter. Ce stage se fait sur la machine qui se nomme *ensisun*. Tapez donc :

```
rlogin ensisun
```

si tout se passe bien, vous devez obtenir le message:

```
login:
```

Ce message signifie que la requête de connexion que vous avez demandée à l'annex a été honorée. L'annex vous a mis en communication avec la machine *ensisun*. Tout ce que vous tapez est maintenant envoyé à *ensisun*, tout ce qui s'affiche sur votre écran vient de *ensisun*.

Ça ne marche pas

Vous n'avez pas obtenu le message `login:`. Vous avez pu commettre deux erreurs:

1. vous avez pu faire une faute de frappe sur la commande `rlogin`. Vous avez obtenu ceci:

   ```
   ensitty1: rkogin ensisun
   CLI: Command not found.
   ```

   Le `CLI` est le "command language interpreter". C'est le programme qui reconnaît les commandes données à l'annex. Il vous dit qu'il ne comprend pas `rkogin` (et pour cause).

2. Vous avez pu faire une faute sur le nom de la machine. Vous avez obtenu ceci:

   ```
   ensitty1: rlogin ensisoleil
   CLI: Named host in unknown to remote name server.
   If given host name was abbreviated, try again using host's full name.
   ```

   Là, le `CLI` vous dit qu'il ne connaît pas le "host" (l'ordinateur) que vous désirez.

Dans un cas comme dans l'autre, retapez la commande `rlogin`, cette fois ci sans erreur!

2.3 La procédure de login

La machine *ensisun* vient d'afficher le message `login:`. Ce message est un prompt : il vous invite à taper votre nom d'utilisateur. Pour que la machine puisse vous accepter, il faut lui donner un nom qui a été préalablement enregistré. En cours d'année, chaque élève dispose d'un nom d'utilisateur (en général son nom de personne physique). Pendant le stage, personne n'est encore enregistré, et le stage se fait en utilisant des noms d'utilisateurs banalisés: `user1`, `user2`, etc...

Regardez en haut à gauche de votre écran, vous devez trouver une étiquette avec un nom `USER`*n*. Tapez ce nom d'utilisateur **en minuscule**. Voici ce qui doit se passer (ici on suppose qu'il s'agit de `user3`) :

```
login: user3
Password:
```

La machine demande quel est votre mot de passe. En mode normal de fonctionnement chaque utilisateur doit avoir un mot de passe qu'il doit garder secret. Pour les besoins du stage, tous les utilisateurs ont été créés avec comme mot de passe `ensimag93`. Tapez donc ce mot de passe. Vous remarquerez que pour éviter que quelqu'un puisse lire le mot de passe sur l'écran, la machine n'affiche pas les caractères que vous tapez.

Si le mot de passe a été correctement tapé, on doit voir ceci sur l'écran:

```
Sun Microsystems Inc.   SunOS 5.2       Generic March 1993
Terminal (vt100):
```

ce message signifie que la machine a accepté l'utilisateur. Elle l'invite maintenant à indiquer le type de terminal qu'il utilise. L'administrateur de la machine *ensisun* a fait en sorte que si on ne répond rien à cette question, le type de terminal qui sera pris par défaut est `vt100`. Ça tombe bien, vous êtes justement sur un terminal compatible *vt100* ! Tapez simplement *Return*, vous devez obtenir:

```
ensisun{1}
```

si tel est le cas, vous avez réussi votre procédure de login. Bravo!

Ça ne marche pas

Si à la place de l'invitation à donner le type de terminal, vous avez le message suivant:

```
Login incorrect
login:
```

c'est que vous vous êtes trompé soit dans le nom d'utilisateur, soit dans le mot de passe, soit dans les deux.

L'erreur classique est de laisser un blanc entre `user` et le numéro, ou bien de taper le nom d'utilisateur ou le mot de passe en majuscule.

Cependant ce n'est pas grave, le message `login:` vous invite à recommencer la procédure de login, tapez à nouveau le nom d'utilisateur et le mot de passe sans erreur cette fois-ci.

2.4 Réflexions sur la procédure de connexion

Prenez le temps de réfléchir à tout ce que vous venez de faire, car la procédure suivie est complexe. Vous avez compris que vous travaillez sur un terminal, ce n'est pas un ordinateur, c'est un simple instrument de communication. Ce terminal n'est pas connecté à un ordinateur, il est connecté à un "serveur de terminaux". Lui-même est connecté à un réseau, sur lequel sont connectés des ordinateurs.

Dans un premier temps, vous communiquez avec l'*annex*, vous lui demandez une connexion à une machine. Une fois cette connexion établie, vous demandez à la machine de vous accepter en temps qu'utilisateur (procédure de login).

Il est nécessaire de bien savoir à tout moment avec quelle entité vous êtes en communication. Heureusement, pour vous aider, chaque entité affiche une petite indication visuelle pour vous rappeler qui est votre interlocuteur. Une telle indication s'appelle un prompt. Quand vous êtes en communication avec l'annex, le prompt est `ensitty1:`. Quand vous êtes en communication avec *ensisun*, en phase de login le prompt est `login:` et quand le login est terminé, le prompt est `ensisun`{*numéro*}

2.5 Retour au serveur de terminaux

2.5.1 Le Break

Je suppose que votre login sur *ensisun* a réussi, vous avez donc `ensisun{1}` affiche sur l'écran.

Repérez sur le clavier du terminal la touche `F5`. Elle porte comme surinscription *Break*. Cette touche **ne génère pas** un caractère. Elle crée un *Break* (une interruption temporaire du signal) sur la ligne de communication du terminal. La machine qui est raccordée au terminal est capable de détecter cette condition inhabituelle et de réagir en conséquence.

Faites un *Break* et regardez ce qui se passe. Vous devez voir s'afficher le prompt de l'*annex*.

Quand l'*annex* nous a donné une communication avec un ordinateur, (en l'occurence *ensisun*), l'*annex* est devenue "transparente". Tout ce qu'on tape au clavier est envoyé a l'ordinateur, on ne dialogue plus avec l'annex. Il faut cependant pouvoir reprendre le dialogue avec l'annex, le *Break* est ce moyen.

2.5.2 Notion de job

Maintenant tapez `help`, vous remarquez la commande `jobs`, tapez ensuite cette commande, vous devez obtenir ceci:

```
+1      rlogin ensisun
```

Cela signifie que votre connexion avec *ensisun* n'est pas perdue. La connexion est seulement suspendue, on pourra la reprendre plus tard.

L'annex permet de gérer plusieurs connexions avec différents ordinateurs. En ce moment vous pourriez demander une connexion avec un autre ordinateur que *ensisun*. Comme pour le stage on ne dispose que de *ensisun*, on va demander une autre connexion avec *ensisun*. Tapez donc `rlogin ensisun` et poursuivez la phase de login jusqu'a sa terminaison correcte, c'est à dire jusqu'a obtenir le prompt `ensisun{1}`.

Maintenant, faites *Break*, et tapez `jobs`, vous devez obtenir :

```
-1      rlogin ensisun
+2      rlogin ensisun
```

voila vos 2 communications en cours qui sont suspendues par l'annex, mais que l'on pourra reprendre quand on le désirera.

On pourrait comme cela avoir 3 connexions différentes avec des ordinateurs. Si on essayait d'obtenir plus de 3 connexions, l'annex refuserait avec le messages:

```
CLI: Job limit exceeded.
```

Vous voyez que l'annex donne le nom de *job* à une connexion.

2.5.3 Action sur les connexions

On peut agir de deux manières différentes sur une connexion : on peut la détruire ou on peut la reprendre. Vous remarquez que lorsque vous tapez `jobs`, l'annex affiche les connexions en leur affectant un numéro d'ordre. Ce numéro permet de désigner une connexion.

1. Pour détruire une connexion, on dispose de l'ordre `kill`. L'ordre kill a la syntaxe suivante: `kill %`*numéro*. Tapez `kill %1` (par exemple) et refaites jobs, pour vérifier que cette connexion a bien été détruite.

2. Pour reprendre une connexion suspendue, on dispose de l'ordre `fg`, qui a la même syntaxe que `kill` : `fg %`*numéro*. Utilisez fg pour reprendre la connexion qui vous reste, et faites quelques *Return* pour vérifier que le prompt que vous obtenez est bien celui de *ensisun*.

2.6 Procédure de logout

La procédure de login est la procédure permettant à un utilisateur d'avoir accès à un ordinateur. La procédure de logout est la procédure par laquelle l'utilisateur préviens l'ordinateur qu'il cessé son travail. L'ordinateur va donc interrompre la communication avec le terminal.

La procédure de logout se résume à un seul ordre: l'ordre `logout`. Je suppose que vous avez le prompt `ensisun{1}`, tapez `logout`, vous devez obtenir ceci:

```
au revoir
CLI: Connection closed
```

La communication avec *ensisun* est rompue, l'annex s'en est bien aperçue, vous êtes donc maintenant en communication avec l'annex.

Si vous avez complètement terminé votre travail et que vous désirez partir, il ne faut pas oublier de tuer les autres connexions que vous auriez pu créer au cours de votre séance de travail. Pour cela deux possibilités:

1. vous faites `jobs` et vous détruisez par `kill` toutes vos connexions.

2. vous tapez l'ordre `hangup`, l'annex détruira automatiquement toutes vos connexions.

vous pouvez alors partir, vous laissez une situation propre.

Attention

Que peut il se passer si vous quittez votre place en laissant une connexion active? Quelqu'un arrivant après vous, a juste à utiliser `fg` pour se trouver connecté sur votre compte, avec vos fichiers. Il peut faire toute modification ou destruction qu'il lui plaît. Prenez donc l'habitude de ne partir qu'après avoir terminé proprement votre session.

Reconnectez vous à *ensisun* pour continuer la manip.

Chapter 3

Le système d'exploitation

3.1 Notion de système d'exploitation

Dans un ordinateur, peuvent s'exécuter deux types de logiciels:

- les logiciels qui fournissent un service particulier à l'utilisateur final, comme par exemple un traitement de texte ou un programme d'analyse statistique. De tels logiciels sont appellés des applications, et ce sont eux qui justifient l'usage d'un ordinateur.
- les logiciels dont le but est de fournir un service à la machine. Dans cette catégorie, on trouve deux types de logiciels:
    1. les logiciels dont le but est d'offrir un service aux applications. Pour que l'expression "offrir un service aux applications" ne reste pas quelque chose de mystérieux, nous allons prendre un exemple. Une application a généralement besoin de dialoguer avec l'utilisateur par l'intermédiaire du clavier et de l'écran. Comme une personne tapant sur un clavier fait de temps en temps des fautes de frappe, il est nécessaire de prévoir une convention permettant de corriger ce type de faute. Le caractère *Back Space* est généralement utilisé à cet effet. Si chaque application devait elle-même gérer les fautes de frappe cela correspondrait à une duplication inutile d'effort. Toute machine comporte donc un logiciel gérant le clavier et l'écran et offrant aux applications le service "lecture au clavier débarrassée des fautes de frappes corrigées".
    2. les logiciels offrant un service à la personne qui gère la machine. La aussi, nous allons prendre un exemple pour donner une idée du type de service dont on a besoin. Une machine comporte une ou plusieurs unités de disques magnétiques permettant de stocker de l'information. L'espace disque n'étant pas infini, il est nécessaire de le gérer, c'est à dire en particulier de pouvoir connaître son taux de remplissage, et pouvoir effacer les informations inutiles. Toute machine est donc livrée avec des logiciels permettant de réaliser ce type d'opération.

  L'ensemble des logiciels offrant des services ainsi orientés vers la machine porte le nom de *système d'exploitation*.

Chaque constructeur d'ordinateur possède généralement son système d'exploitation "maison", mais dans le domaine de la micro-informatique, il y a une tendance à utiliser des systèmes standard, essentiellement MS/DOS pour les machines mono-utilisateur, et UNIX pour les machines multi-utilisateurs.

3.2 Notion de commande

Nous allons parler ici de la fraction du système d'exploitation que nous avons définie comme: "logiciels offrant un service à la personne qui gère la machine ". Dans UNIX, ces services sont réalisés par des programmes portant le nom de **commande**. Certaines commandes, particulièrement simples, ne nécessitent pour être activées, que la frappe de leur nom sur le clavier.

Manipulation

UNIX comporte une commande permettant d'imprimer la date et l'heure. Cette commande porte le nom de *date*. Activez-la en tapant sur votre clavier

`date` *Return*

bien qu'en anglais, la réponse s'interprète sans peine.

D'autres commandes, faisant un travail paramétrable par l'utilisateur, nécessitent qu'on leur fournissent des paramètres lors de l'activation. L'activation d'une telle commande se fait en tapant sur la même ligne le nom de la commande suivi de ses paramètres, comme ceci:

nom-de-la-commande $paramètre_1$ $paramètre_2$ ... $paramètre_n$ *Return*

Manipulation

UNIX comporte une commande, nommée *cal*, permettant d'imprimer le calendrier d'un mois quelconque d'une année comprise entre 0 et 9999. A l'activation, il faut lui fournir en paramètre le numéro du mois et de l'année choisis. Pour vous familiariser avec son emploi, tapez par exemple:

`cal 2 1988` *Return*

puis faites imprimer le mois de votre naissance.

3.3 Notion d'interpréteur de commandes

Le lecteur est en droit de se demander par quel miracle le seul fait de taper le clavier le nom d'une commande permet d'activer celle-ci. L'*interpréteur de commandes* est la réponse à cette question. L'interpréteur de commandes est un logiciel faisant partie du système d'exploitation UNIX dont le but est de faire l'interface entre la personne travaillant au terminal et les commandes. Plus précisément, l'interpréteur de commandes réalise le travail suivant:

1. écriture sur l'écran de l'invitation à taper. (il s'agit d'une courte chaîne de caractères généralement réduite à un seul caractère, le caractère `%` ou `$`).
2. lecture sur le clavier d'une ligne tapée par l'utilisateur et terminée par *Return*
3. interprétation du premier mot comme étant le nom d'une commande, et des autres mots comme étant les paramètres de la commande.
4. activation de la commande désirée.
5. retour à l'étape numéro 1.

Manipulation

Que se passe-t-il si l'utilisateur tape un nom de commande incorrect? tapez par exemple:

`xyz32` *Return*

et mémorisez bien le message d'erreur: c'est celui que vous donnera l'interpréteur de commandes à chaque fois que vous taperez quelque chose qui ne correspond à aucun des nom de commande qu'il connaît.

3.4 Notion de fichier

Les buts principaux d'un ordinateurs sont:

1. conserver l'information
2. traiter l'information
3. transmettre l'information

ce que nous avons à dire dans ce paragraphe concerne la fonction de conservation de l'information.

Etant donné qu'un utilisateur peut avoir une masse considérable d'information conservée sur une machine, il n'est pas possible de la conserver "en vrac", il faut la structurer.

La solution que l'on donne à ce problème dans le travail de secrétariat, consiste à classer des feuilles dans des chemises.

La solution informatique de ce problème a consisté à inventer le concept de fichier. Un fichier peut être vu comme un container à information.

A tout instant, un utilisateur d'un système UNIX, possède, mémorisé par la machine, un ensemble de fichiers.

Nous allons voir maintenant quels sont les services que peut nous rendre le système concernant la manipulation des fichiers

3.5 Commandes de manipulation des fichiers

3.5.1 Lister les noms des fichiers

Le premier besoin éprouvé est de pouvoir répondre à la question: Quels sont les fichiers que je possède?. Pour cela, on dispose d'une commande, la commande `ls` (abréviation de list), qui, lorsqu'on la tape au clavier, demande au système d'afficher la liste des noms des fichiers que l'on possède.

Tapez donc:

`ls` *Return*

le système doit vous répondre par une liste de noms qui sont les noms de vos fichiers. Ce sont des fichiers que nous avons avons installés sur la machine pour que vous puissiez travailler.

3.5.2 Voir le contenu d'un fichier

Pour voir le contenu d'un fichier, le plus efficace est d'utiliser la commande `more` qui admet en paramètre le nom du fichier concerné. En utilisant la commande `ls`, vous avez vu que vous disposez d'un fichier dont le nom est `chap1`, vous allez regarder son contenu. Pour cela, tapez:

`more chap1` *Return*

Sur votre écran doit s'afficher le début du fichier. Quand on est sous le contrôle de `more`, l'utilisateur peut émettre plusieurs ordres, dont les principaux sont:

- *Return*, pour faire apparaître une ligne supplémentaire du fichier.
- *Espace*, (c'est à dire appui sur la barre d'espacement), pour faire apparaître 24 lignes supplémentaires du fichier.
- `b` pour remonter dans le texte
- `q` pour arrêter l'exécution de `more`.

Utilisez ces trois ordres pour constater leur effet.

3.5.3 Détruire un fichier

Quand on ne désire plus conserver un fichier, on peut l'effacer de l'espace disque à l'aide de la commande `rm` (abréviation de remove). Utilisez `ls` pour vérifier que vous possédez un fichier qui porte le nom `brouillon`. Détruisez ce fichier en tapant:

`rm brouillon` *Return*

utilisez ensuite `ls` pour vérifier qu'il a bien disparu.

Attention

Avec le système UNIX, il n'y a aucun moyen de récupérer un fichier que l'on a détruit. Bien réfléchir avant d'utiliser `rm`.

3.5.4 Copier un fichier

Il arrive souvent que l'on veuille dupliquer un fichier par exemple pour faire une sauvegarde avant modification. On dispose d'une commande permettant de procéder à cette opération de duplication, il s'agit de la commande `cp` (abréviation de copy). Cette commande admet deux paramètres qui sont respectivement le nom du fichier original, et le nom du fichier que l'on désire créer. Faites une copie du fichier `chap1` en tapant:

`cp chap1` *nom-de-fichier* *Return*

en utilisant un nom de votre choix pour *nom-de-fichier*. Puis, vérifiez à l'aide de `ls` qu'un nouveau fichier vient d'apparaître, et vérifiez à l'aide de `more` que son contenu est bien le même que celui de `chap1`.

3.5.5 Changement du nom d'un fichier

La commande permettant de changer le nom d'un fichier est la commande `mv` (abréviation de move[1]) qui, comme la commande `cp` admet deux paramètres. Le premier paramètre est l'ancien nom du fichier, et le deuxième paramètre est le nouveau nom. Changez le nom du fichier que vous avez créé lors de l'exercice précédent en tapant:

`mv` *ancien-nom-de-fichier* *nouveau-nom-de-fichier* *Return*

Vérifiez à l'aide de `ls` et `more` que tout s'est bien passé.

[1]Le mot move peut paraître curieux pour une commande permettant de changer le nom d'un fichier, mais en réalité la commande move permet de faire des choses plus compliquées dont nous ne parlons pas maintenant.

Chapter 4

L'éditeur vi

4.1 Notion d'éditeur de textes

Nous venons de voir les commandes de base de manipulation de fichiers. Nous avons vu comment lister les noms des fichiers à l'aide de `ls`, comment voir le contenu d'un fichier à l'aide de `more`, comment effacer un fichier à l'aide de `rm`, comment copier un fichier à l'aide de `cp` et comment changer le nom d'un fichier à l'aide de `mv`. Mais toutes ces commandes agissent sur les fichiers considérés comme un tout. Il nous manque donc un outil permettant de faire des modifications à l'intérieur d'un fichier. Un tel outil porte le nom d'*éditeur de textes*, ou en abrégé, d'*éditeur*. Le système UNIX comporte un éditeur qui se nomme `vi`, nous allons en étudier les principales caractéristiques.

4.2 Appel de l'éditeur `vi`

A l'aide de la commande `ls` vérifiez que vous disposez d'un fichier dont le nom est `chap1`. Nous allons activer l'éditeur `vi` en le faisant travailler sur ce fichier. Pour cela émettez la commande:

`vi chap1` *Return*

votre écran doit alors se remplir avec un texte qui est le début du fichier `chap1`.

4.3 Interaction entre l'utilisateur et le terminal

Lorsque l'utilisateur travaille sous le contrôle du système UNIX, tout ce qu'il tape s'affiche sur l'écran, et un ordre donné au système n'est pris en compte que lorsque l'utilisateur a tapé sur la touche *Return*.

L'interaction avec un éditeur comme `vi` se passe de manière différente. L'écran est entièrement utilisé pour afficher une fraction du fichier sur lequel on travaille, et n'est dont plus disponible pour afficher les ordres tapés par l'utilisateur. L'utilisateur est donc obligé de taper des ordres sans avoir de contrôle visuel de ce qu'il a tapé. Ceci n'est pas gênant dans la mesure ou les ordres que l'on peut donner à l'éditeur se composent d'un très petit nombre de caractères. Autre différence, il n'est pas nécessaire de terminer un ordre par l'appui de la touche *Return*.

4.4 Concepts utilisés par l'éditeur

4.4.1 Concept de fenêtre courante

Un fichier pouvant contenir un nombre très important de lignes, et votre terminal ne pouvant en afficher que 23 ou 24, il est en général impossible de faire tenir le fichier en entier sur l'écran. A tout instant, `vi` affiche sur l'écran 24 lignes consécutives du fichier qui portent le nom de *fenêtre courante*. Cette expression vient du fait que l'écran peut être vu comme une fenêtre ouverte sur le fichier, et elle est dite courante, car on peut la déplacer sur le fichier.

4.4.2 Concept de curseur courant

Un caractère particulier dans la fenêtre est particularisé: c'est celui que l'on appelle le *curseur*. Pour que l'utilisateur humain puisse savoir sur quel caractère se trouve le curseur, `vi` affiche sur ce caractère un petit rectangle lumineux généralement clignotant. Tout comme la fenêtre, le curseur peut se déplacer dans le texte, et c'est la raison pour laquelle on l'appelle le *curseur courant*.

4.4.3 Concept de ligne courante

La *ligne courante* est la ligne sur laquelle se trouve le curseur.

Manipulation

Repérez l'endroit où se trouve le curseur dans la fenêtre. Comme vous venez d'invoquer `vi`, le curseur doit normalement se trouver sur le premier caractère de la première ligne.

4.5 Déplacements dans le texte

4.5.1 Défilement de la fenêtre

Il existe quatre ordres permettant de déplacer la fenêtre en avant et en arrière sur le texte. Il s'agit des ordres *Control F* et *Control B* d'une part, et *Control D* et *Control U* d'autre part. L'ordre *Control F* fait avancer la fenêtre (elle va vers la fin du texte), et l'ordre *Control B* fait reculer la fenêtre (elle va vers le début du texte). La progression se fait écran par écran, c'est à dire 24 lignes par 24 lignes.

Les ordres *Control D* et *Control U* ont respectivement le même effet que *Control F* et *Control B* sauf que la progression dans le texte se fait demi-écran par demi-écran.

Mnémotechnique: F pour forward, B pour backward, D pour down, U pour up.

Manipulation

1. Vous êtres normalement au début du texte, faites *Control F* et observez ce qui se passe. Faites le nombre de *Control F* nécessaire pour atteindre la fin du texte. Quand vous êtes à la fin du texte, l'éditeur peut afficher des lignes ne contenant qu'un seul caractère: le caractère ~ en début de ligne. Cela ne veut pas dire que ce caractère se trouve dans votre fichier, mais signale simplement que ces lignes de l'écran ne sont pas utilisées pour afficher une partie de votre texte.

2. Faites le nombre de *Control B* nécessaire pour revenir au début du texte.

3. Refaites un parcours complet du texte en avant puis en arrière avec *Control D* et *Control U*

4.5.2 Positionnement du curseur dans la fenêtre

Pour positionner très rapidement le curseur près de l'endroit où on désire aller, on dispose des trois ordres `H`, `M`, `L` qui sont des ordres de positionnement relatifs à la fenêtre courante. L'ordre `H` positionne le curseur sur la ligne du haut, `M` le positionne sur la ligne du milieu et `L` le positionne sur la dernière ligne. Mnémotechnique: H pour home, M pour middle, L pour last.

Manipulation

Emettez des ordres `H`, `M` et `L` en regardant leur effet sur l'écran. Attention à bien taper `H`, `M` et `L` en majuscule, il existe également des ordres `h`, `m` et `l` qui ont un effet différent.

4.5.3 Déplacement du curseur dans les 4 directions

Après avoir fait un positionnement grossier grâce aux ordres précédents, on peut faire un positionnement précis à l'aide d'ordres permettant de se déplacer caractère par caractère en horizontal, et ligne par ligne en vertical.

Il existe pour réaliser cela trois groupes d'ordres:

- Si votre terminal dispose de 4 touches comportant les flèches dans les 4 directions, vous pouvez faire bouger le curseur dans la direction indiquée.

  > **Attention:** Contrairement aux touches normales qui ne produisent qu'un seul caractère, les touches "flèches" envoient plusieurs caractères à la machine. Lorsque beaucoup d'utilisateurs utilisent simultanément les flèches de leur terminal, cela peut amener une saturation de la machine au niveau des E/S. Si ce phénomène se produit, l'utilisateur se retrouve avec des codes bizarres dans son fichier. Il vaut alors mieux ne pas utiliser les flèches, et utiliser la méthode décrite ci-après.

- à la place des flèches, il est possible d'utiliser les touches h, j, k, l qui ont le même effet respectivement que les touches à gauche, en bas, en haut, à droite. On utilise ces touches dans le cas d'utilisation de terminaux dont le clavier ne comporte pas les flèches.

- On dispose également des ordres `-`, `+`, *Espace* et *Back Space*. L'ordre `-` permet d'aller à la ligne précédente, `+` permet d'aller à la ligne suivante, *Espace* et *Back Space* ont respectivement le même effet que flèche à droite et flèche à gauche.

Attention

Quand vous utilisez les flèches, appuyez et relâchez comme pour une touche normale, **ne maintenez pas le doigt appuyé**.

Manipulation

- Faites bouger le curseur dans la fenêtre grâce aux flèches. Que se passe-t-il quand on est sur le premier caractère d'une ligne et que l'on fait "à gauche" ou bien sur le dernier caractère et que l'on fait "à droite" ?

- Utilisez également les lettres h, j, k et l pour vérifier que l'on obtient bien le même résultat.

- Utilisez maintenant `-`, `+`, *Espace* et *Back Space*. Quelle différence y a-t-il entre `-` et flèche en haut d'une part, et `+` et flèche en bas d'autre part?

4.5.4 Positionnement sur une ligne dont on connaît le numéro

Il arrive fréquemment que l'on connaisse le numéro de la ligne sur laquelle on désire positionner le curseur. Pour réaliser cela, il faut taper le numéro de la ligne en question suivi de G (en majuscule)

Manipulation

Tapez

```
30G
```

faites attention à ne pas taper de blanc entre le nombre et G, et à bien taper G en majuscule. L'éditeur doit afficher une nouvelle page d'écran et positionner le curseur sur la ligne:

```
Pour la soiree, il n'y avait pas de programme. Le jeune pianiste jouait, mais
```

si ca ne marche pas, c'est que vous avez fait une faute de frappe, recommencez jusqu'a y arriver.

Faites ensuite quelques essais avec des numéros pris au hasard, par exemple 15G 50G, puis essayez les deux cas particuliers 1G et G pour voir leur action.

Que se passe-t-il si on utilise l'ordre G avec un numéro de ligne incorrect, par exemple un numéro trop grand? (votre texte comporte 79 lignes) Que se passe-t-il si on utilise un numéro négatif, par exemple -12G? Expliquez ce qui se passe.

4.5.5 Déplacement du curseur dans la ligne

L'ordre `^` (accent circonflexe) permet de positionner le curseur sur le premier caractère non blanc de la ligne, l'ordre `0` (zéro) permettant de se positionner sur le premier caractère de la ligne, quel qu'il soit.

L'ordre `$` permet de se positionner sur le dernier caractère de la ligne.

Les ordres `w` et `b` permettent de faire progresser le curseur de mot en mot respectivement en avant et en arrière. L'ordre `e` permet de positionner le curseur sur le dernier caractère d'un mot. Les ordres `W`, `B` et `E` sont des variantes de `w`, `b` et `e` en prenant une définition légèrement différente de ce qu'est un mot.

Manipulation

Positionnez vous sur la ligne 65, qui est la ligne suivante:

```
"Vous croyez qu'elle en mourrait, votre mere, s'ecria durement Mme Verdurin,
```

et utilisez les ordres vus dans ce paragraphe pour vous déplacer dans la ligne. Essayez de comprendre la différence entre `w` et `W`, entre `b` et `B` et entre `e` et `E`.

Que se passe-t-il quand on est sur le dernier mot d'une ligne et qu'on fait `w`, ou qu'on se trouve sur le premier mot d'une ligne et qu'on fait `b`?

4.5.6 Recherche d'une chaîne de caractères

On dispose de deux ordres `/` et `?` permettant de rechercher dans le texte une chaîne de caractères. L'ordre `/` effectue la recherche à partir du curseur courant, en allant vers la fin du texte. L'ordre `?` effectue la recherche à partir du curseur courant, en remontant vers le début du texte. Lors de l'utilisation de l'ordre `/` (resp. `?`), si la recherche a échoué lorsqu'on atteint la fin (resp. le début) du texte, la recherche se poursuit à partir du début (resp. la fin) du texte jusqu'au curseur courant.

Manipulation

- Recherche en allant vers la fin du texte.

  1. Placez vous au début du texte.
  2. Tapez `/` et regardez votre écran: l'éditeur a effacé la dernière ligne de l'écran et y a mis le caractère `/`. Il attend que vous tapiez la chaîne de caractères que vous désirez trouver.
  3. Tapez alors `rebelles` suivi de *Return*.
  4. vérifiez que le curseur est positionné sur le mot `rebelles`.

- Recherche en allant vers le début du texte.

  1. Placez vous sur la dernière ligne du texte.
  2. Utilisez `?` pour rechercher la chaîne `Mais au fur` en tapant `?Mais au fur` suivi de *Return*.

- Que se passe-t-il quand on recherche une chaîne de caractères qui n'existe pas?

4.6 Les deux modes de l'éditeur

L'éditeur possède deux modes de fonctionnement: le mode *commande* et le mode *insertion*.

- Le mode *commande* est le mode que vous avez utilisé jusqu'à présent. Dans ce mode, les caractères tapés par l'utilisateur sont interprétés comme des ordres, et immédiatement exécutés. Ce mode est caractérisé par le fait que les caractères tapés au clavier ne sont pas affichés sur l'écran.

- Le mode *insertion*, est un mode dans lequel **tout ce qui est tapé au clavier est mis dans le fichier**. Ce mode est caractérisé par le fait que les caractères tapés au clavier sont affichés aussitôt sur l'écran.

Il existe un certain nombre d'ordres permettant de passer du mode commande au mode insertion, nous les verrons dans les prochains chapitres. Mais il n'existe qu'un seul moyen de passer du mode insertion au mode commande, c'est d'appuyer sur la touche *Escape*.

4.7 Insertion de texte

4.7.1 Insertion de texte dans une ligne

On dispose de deux ordres permettant d'insérer du texte à l'intérieur d'une ligne, ce sont les ordres `i` et `a`. L'ordre `i` a pour effet d'insérer du texte avant le curseur courant, alors que `a` a pour effet d'insérer du texte après le curseur courant.

Dans les deux cas, après avoir émis l'ordre `i` ou `a`, l'éditeur passe dans le mode insertion, l'utilisateur tape ensuite ce qu'il veut faire entrer dans le fichier, puis il signale la fin du mode insertion en tapant *Escape*. Il se trouve alors dans le mode commande.

Manipulation

- Utilisation de `i`
  1. recherchez le mot `feme`
  2. Placez le curseur sur le `e` terminal
  3. Tapez `i` (passage en mode insertion)
  4. Tapez le `m` qui manque (vérifiez qu'il s'affiche sur l'écran)
  5. Tapez *Escape* (retour au mode commande)
- Utilisation de `a`
  1. Placez vous sur la ligne 18, il faut transformer le mot `annee` en l'expression `annee-la`
  2. Placez le curseur sur le `e` terminal du mot `annee`
  3. Tapez l'ordre `a` (passage en mode insertion)
  4. Tapez `-la` (vérifiez que `-la` s'affiche sur l'écran)
  5. Tapez *Escape* (retour au mode commande)
- Entraînement: à la ligne 52 transformez le mot `camarades` en `"camarades"`.

4.7.2 Insertion de texte entre deux lignes

Quand on désire ajouter une ou plusieurs lignes entre deux lignes de texte, le plus efficace est d'utiliser soit l'ordre `o`, soit l'ordre `O` (o pour open). Ces deux ordres ont pour effet d'ouvrir une ligne vide entre deux lignes du texte, après la ligne courante pour `o`, et avant la ligne courante pour `O`. Dans les deux cas, après ouverture de la ligne vide, **l'éditeur est en mode insertion**. Tout ce que l'utilisateur tape ensuite sera donc mis dans le texte. L'utilisateur tapera donc une ou plusieurs lignes, et comme dans le cas des ordres `i` et `a`, signalera la fin de l'insertion en tapant *Escape*.

Manipulation

- Choisisez un endroit dans le texte et rajoutez une ou plusieurs lignes de texte quelconque. Vérifiez ensuite que vous êtes bien en mode commande.
- Ajoutez une ou plusieurs lignes avant la première ligne du texte.
- Ajoutez une ou plusieurs lignes après la dernière ligne du texte.

4.8 Effacement de texte

4.8.1 Effacement d'un caractère

L'ordre `x` permet d'effacer le caractère se trouvant à l'endroit du curseur courant. Cet ordre peut être précédé d'un nombre indiquant le nombre de caractères à effacer. par exemple `3x` efface 3 caractères à partir du curseur courant.

Manipulation

Recherchez le mot `MMme` et enlevez le `M` en trop, puis recherchez le mot `sexxxe` et enlevez les 2 `x` en trop.

4.8.2 Effacement de mots

L'ordre `dw` permet d'effacer le texte depuis le curseur courant jusqu'au début du mot suivant. Cet ordre peut donc être utilisé de trois manières différentes selon l'endroit où est positionné le curseur au moment où on émet l'ordre `dw`

- Si le curseur est positionné au milieu d'un mot, `dw` efface la fin du mot et les blancs qui le suivent.
- Si le curseur est positionné sur la première lettre d'un mot, `dw` efface le mot entier et les blancs qui le suivent.
- si le curseur est positionné entre deux mots, `dw` efface les blancs jusqu'au prochain mot.

Cet ordre peut lui-aussi être précédé d'un nombre et c'est ainsi que `3dw` fera l'effet de trois `dw` successifs.

Manipulation

Utilisez `dw` pour effacer des mots ou fins de mots de votre choix dans le texte.

4.8.3 Effacement d'une ligne ou d'une fin de ligne

Pour effacer une ligne, il suffit, quelque soit l'endroit où se trouve le curseur dans la ligne, d'émettre l'ordre `dd`. Comme on l'aura deviné, l'ordre `4dd` permettra d'effacer 4 lignes successives à partir de la ligne contenant le curseur.

Pour effacer depuis le curseur jusqu'a la fin de la ligne courante, utiliser `D`.

Manipulation

Utilisez `D` pour effacer des fins de lignes de votre choix. Placez vous à la fin du texte et effacer la dernière réplique qui est en double.

4.9 Remplacement de texte

Tout remplacement de texte pourrait se faire par un effacement suivi d'une insertion, mais remplacer du texte est si fréquent, que l'éditeur comporte des ordres particuliers pour ce faire.

4.9.1 Remplacement d'un caractère par un caractère

Pour réaliser cette modification, il suffit de positionner le curseur sur le caractère à remplacer, émettre l'ordre `r`, et taper le nouveau caractère.

Manipulation

Recherchez le mot `Verduron` et remplacez le `o` par un `i`.

4.9.2 Remplacement de n caractères par n caractères

Pour réaliser cette opération, il faut positionner le curseur sur le premier caractère à modifier, taper l'ordre `R`, taper les nouveaux caractères, et tapez *Escape* quand le remplacement est terminé.

On remarquera la différence d'utilisation des ordres `r` et `R` : puisque `R` permet de modifier un nombre quelconque de caractères, il faut, grâce à *Escape*, signifier à l'éditeur quand on a terminé le remplacement, alors que `r` ne permettant de modifier qu'un seul caractère, ceci n'est pas nécessaire.

Manipulation

Recherchez le mot Valkyrie et remplacez le `V` par un `W`. Recherchez le mot facteur et remplacez le par docteur.

4.9.3 Remplacement d'un mot par un autre mot

Il arrive fréquemment que l'on ait à remplacer un mot par un autre mot de longueur différente, et donc que l'ordre `R` soit malaisé à utiliser. Pour cela, on dispose de l'ordre `cw` qui permet de remplacer un mot par un autre quelque soient leurs longueurs respectives. L'ordre `cw` fait passer en mode insertion, l'utilisateur signale donc la fin du nouveau mot en tapant *Escape*.

Manipulation

1. Placez le curseur sur le premier caractère du mot `feminin`.

2. Emettez l'ordre `cw` et regardez attentivement votre écran pour voir ce qui a changé sur l'affichage du mot.

3. Tapez à la place le mot `fort`.
4. Signifiez à l'éditeur la fin de l'ordre en tapant *Escape*.
5. regardez maintenant ce qui est affiché sur l'écran.

Que se passe-t-il lorsqu'on émet l'ordre `cw` alors que le curseur est positionné au milieu d'un mot?

4.10 Récupération de texte perdu

Chaque fois que l'on demande à l'éditeur d'effacer une partie du texte, par exemple à l'aide des ordres `x`, `dw` ou `dd`, l'éditeur mémorise dans un espace de travail qui lui est personnel, la fraction du texte qui vient d'être effacé.

D'autre part, il existe deux ordres `p` et `P` dont le but est d'insérer à l'endroit où se trouve le curseur, la dernière fraction de texte qui a été mémorisée après effacement. L'ordre `p` insère le texte mémorisé juste après le curseur, alors que l'ordre `P` insère juste avant le curseur.

Ceci a pour conséquence que si l'on se rend compte, après avoir émis l'ordre `dw`, qu'il s'agit d'une erreur, on peut émettre l'ordre `p` ou `P` pour récupérer ce qui avait été effacé. Dans le cas général, c'est l'ordre `P` qui permettra de retrouver le texte à l'endroit précis où il se trouvait avant effacement. Ce n'est que dans le cas où après effacement, le curseur se trouve sur le dernier caractère de la ligne qu'il faut utiliser l'ordre `p`.

Manipulation

Entraînez vous à effacer des caractères, des mots et des lignes puis à les récupérer par `P` pour prendre confiance dans le mécanisme. Effacez des éléments se trouvant en fin de ligne (par exemple le dernier mot d'une ligne) pour bien voir pourquoi il faut utiliser `p` et non pas `P` dans ce cas.

4.11 Opérations diverses

4.11.1 Couper une ligne en deux

Il n'existe pas d'ordre particulier pour couper une ligne en deux. Du point de vue de l'éditeur, une telle opération consiste simplement à insérer un *Return* à l'endroit où on veut couper la ligne. Pour cela, il faut donc positionner le curseur à l'endroit où on veut couper, émettre l'ordre `i` (ou `a` selon l'endroit précis où on a mis le curseur), taper *Return*, et taper *Escape*.

Manipulation

Choisisez une ligne quelconque et coupez la en deux entre deux mots.

4.11.2 Concaténer deux lignes

Le curseur étant positionné sur une ligne, l'ordre `J` (J pour join), a pour effet de concaténer la ligne suivante à la ligne courante.

Manipulation

Concaténez les deux lignes que vous avez obtenues après en avoir coupé une en deux.

4.11.3 Déplacement de blocs de lignes

Il arrive fréquemment que l'on veuille déplacer un bloc entier de lignes, c'est à dire les effacer d'un endroit et les mettre à un autre. Ceci se fera à l'aide d'une commande d'effacement (ex `15dd`) suivi du positionnement du curseur au bon endroit, suivi de l'ordre `p` ou `P`.

Manipulation

Choisisez un paragraphe quelconque et mettez le à un autre endroit dans le texte.

4.11.4 Recopie de blocs de lignes

La commande consistant à taper un nombre suivi de `yy`[1] demande à l'éditeur de mémoriser un bloc de lignes à partir de la ligne courante. Le nombre de lignes mémorisées est égal au nombre tapé avant `yy`. Une fois mémorisé, le bloc de lignes peut être recopié ailleurs à l'aide d'une commande `p` ou `P`.

Manipulation

Choisisez un paragraphe quelconque et recopiez le à un autre endroit dans le texte.

4.12 Modifications itératives

4.12.1 Commande `n`

L'éditeur mémorise la dernière chaîne de caractères qui a été recherchée par l'ordre `/` ou par `?`. L'utilisateur dispose de l'ordre `n` (n pour next) qui a pour but de relancer la recherche de la même chaîne de caractères. Sachant qu'une recherche se fait à partir du curseur courant, on voit que des ordres `n` successifs vont permettre de découvrir toutes les occurences d'une chaîne dans le texte.

Manipulation

Trouvez toutes les occurences du mot `Verdurin` dans le texte.

4.12.2 Commande `.` (point)

L'éditeur mémorise également le dernier ordre ayant entraîné une modification du texte, par exemple `x`, `dw`, etc... L'ordre `.` permet de demander à l'éditeur de refaire cette modification.

[1] y pour yank (qui signifie tirer)

Manipulation

- Effacer un mot de votre choix puis émettez des ordres `.` pour voir le résultat.
- De la manière la plus rapide possible, transformez toutes les occurences du mot `Verdurin` en `Vilmorin`.

4.13 Commande `u`

La commande `u` (u pour undo) permet de défaire la dernière modification apportée au texte. Cette commande est très utile en cas d'erreur. Supposons que je veuille taper 2dd pour effacer deux lignes, et que mon doigt reste trop longtemps appuyé sur la touche `2`, ce qui a pour conséquence d'émettre `222dd`. Après effacement des 222 lignes, il me suffit de taper `u` pour les récupérer.

Manipulation

Emettez des commandes quelconques d'effacement, et récupérez le texte perdu à l'aide de `u`.

4.14 Sortie de l'éditeur

L'éditeur `vi` ne modifie pas directement le fichier sur lequel on lui a demandé de travailler, mais utilise une copie en mémoire de ce fichier. Quand on a terminé de travailler avec l'éditeur, on a donc le choix entre:

- quitter l'éditeur en lui demandant d'écraser l'ancienne version du fichier avec la nouvelle que l'on vient de faire.
- quitter l'éditeur en lui demandant d'annuler tout ce qui a été fait pendant cette session d'édition. On récupére alors le fichier dans sa version originale.

Il existe donc deux ordres permettant de sortir de l'éditeur:

1. `ZZ` permet de stopper l'éditeur en validant les modifications faites (l'ancienne version du fichier est perdue).
2. `:q!` *Return* permet de stopper l'éditeur en récupérant l'ancienne version du fichier. Dans ce cas, c'est le travail de la session qui est perdu.

Dans le cas général, le travail avec `vi` se termine donc par `ZZ`, ce n'est que dans le cas où on se rend compte que l'on a fait des modifications catastrophiques que l'on sort par `:q!` *Return*.

Chapter 5

Le système de fichiers de UNIX

5.1 Notion de répertoire

Nous avons vu que le premier instrument de structuration de l'information mémorisée sur un ordinateur est le fichier. Les fichiers permettent de ne pas conserver l'information en vrac, mais au contraire de la découper dans différents fichiers.

Mais bien évidemment, dès qu'on gère un grand nombre de fichiers, le même problème de classement se repose, cette fois-ci non plus au niveau de l'information, mais au niveau des fichiers.

Reprenons l'analogie avec le travail de secrétariat. Le premier niveau de classement consiste à mettre des feuilles dans des chemises, et lorsque le nombre de chemises gérées devient trop important, on classe les chemises dans divers dossiers.

L'informatique a copié le procédé, et a créé le concept de *répertoire*: un répertoire permettant de contenir des fichiers. L'image que l'on doit se faire de son espace de stockage d'information ne doit donc pas être un ensemble de fichiers, mais un ensemble de répertoires, chaque répertoire contenant un nombre quelconque de fichiers. Si l'on représente graphiquement la relation d'appartenance d'un fichier à un répertoire par une flèche du répertoire vers le fichier, on pourra schématiser son espace de stockage de la manière suivante:

```
      rep1               rep2
       /\                 /\
      /  \               /  \
   fic1  fic2         fic3  fic4
```

Mais les objets informatiques ne sont pas soumis à des contraintes matérielles comme peuvent l'être le papier et le carton, et l'informatique a élargi la définition du répertoire, en disant qu'un répertoire permet de contenir des fichiers **et des répertoires**.

Nous obtenons alors le schéma du type suivant:

```
        rep1                      rep2
         /\                        /\
        /  \                      /  \
       /    \                    /    \
     fic1   rep3              rep4    fic2
             /\                /\
            /  \              /  \
           /    \            /    \
         fic3  fic4        fic5   fic6
```

L'utilisateur doit donc voir son espace de stockage comme étant un ensemble d'arborescences dont les éléments terminaux sont des fichiers.

[En anglais, répertoire se dit "directory"]

5.2 Notion de répertoire courant

Un utilisateur possède généralement une arborescence très grande, mais à un instant donné il fait un travail qui ne met en jeu qu'un petit nombre d'objets (fichiers et répertoires). Un utilisateur ne désire donc généralement pas avoir la visibilité de tous les objets qu'il possède, mais simplement d'un petit nombre de ceux-ci. Dans le travail de secrétariat, ce résultat est obtenu en extrayant d'une armoire un dossier et en étalant les chemises qu'il contient sur la table de travail. Pour donner à l'utilisateur de moyens informatiques le même type de service, on a inventé le concept de *répertoire courant*. A tout instant, le système connaît un répertoire particulier de l'utilisateur comme étant son répertoire courant, et les commandes ont un effet relatif à ce répertoire.

Par exemple la commande `ls` qui a été vue précédemment ne liste pas tous les fichiers et répertoires de l'utilisateur, mais seulement ceux qui se trouvent dans le répertoire courant.

[En anglais, répertoire courant se dit "working directory"]

5.3 Les informations associées aux fichiers et répertoires

Un utilisateur gère deux types d'objets informatiques: les fichiers et les répertoires. La première des choses qu'il doit savoir faire est déterminer le type de chaque objet.

Pour cela, il faut utiliser la commande `ls` sous une forme plus élaborée que ce que nous avons fait jusqu'a présent, en utilisant l'argument `-l` de la commande. Cet argument (l pour long) demande à `ls` de lister les objets en donnant plus d'informations que simplement leur noms.

Tapez:

`ls -l` *Return*

et vous devez obtenir quelque chose du style de:

```
total 53
-rw-r--r--   1 cassagne  public      2721 Sep 15 13:23 LISEZ_MOI
-rwxr-xr-x   1 cassagne  public      9216 Sep 15 13:23 bravo
-rw-r--r--   1 cassagne  public        74 Sep 15 13:23 bravo.c
-rw-r--r--   1 cassagne  public      5276 Sep 15 13:23 brouillon
-rw-r--r--   1 cassagne  public      5272 Sep 15 13:23 chap1
drwxr-xr-x   2 cassagne  public       160 Sep 15 13:27 courrier
drwxr-xr-x   2 cassagne  public       192 Sep 15 13:27 livre
drwxr-xr-x   3 cassagne  public       144 Sep 15 13:27 shell
```

Cette fois-ci, `ls` a listé les objets du répertoire courant en en mettant un par ligne, le nom de l'objet se trouvant à la dernière colonne. Les autres colonnes contiennent d'autres informations que le système mémorise avec chaque objet.

- la colonne 3 contient le nom du propriétaire de l'objet, ici `cassagne`.
- la colonne 4 contient le nom du groupe auquel appartient le propriétaire de l'objet, ici `public`.
- la colonne 5 contient la taille de l'objet exprimée en octets, ici 5272 octets pour le fichier `chap1`.
- les colonnes 6, 7 et 8 contiennent la date et l'heure de dernière modification de l'objet, par exemple le 15 septembre à 13 heures 27 pour `chap1`.
- nous n'expliciterons pas ici toutes les informations données par la première colonne, nous ne nous intéresserons qu'au premier caractère de cette colonne. Ce premier caractère est soit la lettre `d` indiquant ainsi que l'objet considéré est un répertoire, soit le caractère `-`, indiquant ainsi que l'objet est un fichier. Vous pouvez donc voir que les objets ayant pour nom bravo, `bravo.c` et `chap1` sont des fichiers, alors que les objets de nom `courrier` et `livre` sont des répertoires.

5.4 Changement de répertoire courant

Nous avons vu que la notion de répertoire courant permettait de nous donner une visibilité partielle de notre arborescence, ce qui est très pratique, mais bien évidemment, il faut pouvoir changer de répertoire courant. Pour cela, on dispose de la commande `cd` (Change Directory). Vous avez vu que vous avez 2 répertoires courrier et livre, vous allez vous placer sur le répertoire courrier. Pour cela tapez:

`cd courrier` *Return*

Maintenant faites:

`ls -l` *Return*

et vous voyez que ce répertoire contient 8 objets qui sont tous des fichiers. D'autre part, vous voyez que la commande `ls` n'a pas listé les fichiers `chap1`, `bravo.c`, etc... et que votre visibilité actuelle est bien restreinte au contenu du répertoire `courrier`.

5.5 Notion de *répertoire d'origine*

Nous avons vu que chaque utilisateur possède dans le système un ensemble de fichiers et d'arborescences, et bien évidemment le système est obligé de gérer tous ces objets pour n utilisateurs. La manière la plus simple pour lui, est de relier tous les objets appartenant à un utilisateur, à un même répertoire. Ce répertoire sera nommé le *répertoire d'origine* de l'utilisateur. Votre espace de fichier a donc la structure suivante:

```
                 xxx   <- repertoire d'origine
                 /|\
                / | \
               /  |  \
              /   |   \
             /    |    \
        courrier livre  bravo.c etc...
          /\     /\
         /  \   /  \
    lettre1    chap1
```

Lorsque vous faites `login`, le système vous donne comme répertoire courant, votre répertoire d'origine. Vous ne voyez ainsi que les objets qui vous appartiennent.

[En anglais, répertoire d'origine se dit "home directory"]

La commande `cd` possède une forme particulière permettant de se positionner sur son répertoire d'origine sans en connaître le nom. Il suffit de taper `cd` sans argument. Tapez:

`cd` *Return*

et tapez ensuite:

`ls` *Return*

pour vérifier que vous êtes bien revenu dans votre répertoire d'origine.

5.6 Création de répertoire

Les répertoires sont un moyen très efficace de classement, encore faut-il savoir en créer. Il existe pour cela une commande qui s'appelle `mkdir`, abréviation de make directory. On l'utilise en lui donnant comme paramètre le nom du répertoire que l'on désire créer. Tapez par exemple:

`mkdir TP` *Return*

pour créer un répertoire de nom `TP`. Utilisez ensuite la commande `ls` pour vérifier que vous avez bien maintenant un autre objet de nom TP et de type répertoire. Utilisez la commande `cd` pour vous positionner sur ce répertoire, et ensuite utilisez `ls` pour voir ce qu'il contient. La commande `ls` ne répond rien, indiquant ainsi que le répertoire est vide.

5.7 Désignation du répertoire père

On entend par répertoire père d'un objet, le répertoire qui contient cet objet. Lorsqu'à partir du répertoire courant on désire désigner un répertoire fils, on donne son nom. par exemple, a partir de votre répertoire d'origine quand vous avez voulu aller sur le répertoire `TP`, vous avez émis la commande:

`cd TP` *Return*

Par contre, lorsqu'on désire désigner le père d'un répertoire, il existe une notation conventionnelle qui est `..` (point point). Il n'y a pas d'ambiguïté puisqu'un répertoire ne peut avoir qu'un seul père. Depuis le répertoire `TP` remontez à votre répertoire d'origine en émettant la commande:

`cd ..` *Return*

et vérifiez à l'aide de la commande `ls` que vous êtes bien revenus au bon endroit.

5.8 Destruction de répertoire

On peut détruire un répertoire depuis le répertoire père, à l'aide de la commande `rmdir` (remove directory). Cette destruction ne peut avoir lieu que si le répertoire à détruire est vide. Sans bouger de votre répertoire d'origine, émettez la commande:

`rmdir TP` *Return*

puis vérifiez à l'aide de la commande `ls` que l'objet `TP` de type répertoire a bien disparu.

5.9 Organisation de l'arborescence UNIX

Puisque nous savons remonter au répertoire père du répertoire courant, nous allons en profiter pour voir si il y a quelque chose au dessus du répertoire d'origine d'un utilisateur.

1. Si vous n'êtes pas sûr d'être sur votre répertoire d'origine, faites:

   `cd` *Return*

2. Remontez d'un niveau dans l'arborescence en faisant:

   `cd ..` *Return*

   Utilisez alors `ls` pour voir ce qui se trouve à ce niveau. Vous devez trouver des répertoires, chacun ayant comme propriétaire un utilisateur du système. L'ensemble des arborescences des utilisateurs se trouve donc regroupée dans un seul répertoire.

3. Remontez à nouveau d'un cran dans l'arborescence, et regardez grâce à `ls -l` ce qui se trouve à ce niveau. Vous devez y trouver un certain nombre de fichiers et de répertoires. Vous devez en particulier trouver:

   - un directory de nom `kernel`. Descendez dans ce répertoire et regardez son contenu. Vous devez y trouver un fichier de nom `unix` : c'est le contenu de ce fichier qui est chargé en mémoire lorsque l'on met la machine en marche, il contient le système d'exploitation UNIX.
   - un répertoire de nom `users`, c'est lui qui contient les sommets des arborescences des utilisateurs que nous venons de voir.
   - un répertoire de nom `bin`. Descendez dans ce répertoire par une commande `cd` appropriée, et regardez ce qui s'y trouve. Vous devez voir des fichiers portant des noms dont certains vous sont familiers. Vous trouvez en particulier `cp`, `rm`, `ls`, `more`, etc... qui sont des noms de commandes. En fait, tous les fichiers de ce répertoire contiennent des programmes exécutables standard d'UNIX que l'on appelle des commandes. Chaque fois que vous tapez `ls`, vous exécutez le programme contenu dans le fichier de nom `ls` se trouvant dans le répertoire `bin`.

On voit donc que l'ensemble des objets appartenant au système, aussi bien que les objets appartenant aux utilisateurs, participent à une même énorme arborescence dont la structure est:

```
                    /    <- racine

                   //\\
                 /  /  \  \
               /    /    \    \
             /      /      \      \
           /        /        \        \
         /          /          \          \
      kernel       bin         etc       users
         |         /|\         /|\        //\\
         |        / | \       / | \      / /  \ \
         |       /  |  \               /  /    \  \
        unix    ls  cp  rm           /   /      \   \
                                    /   /        \   \
                                 user1  user2   user3  user4   etc...
                                  /|\    /|\     /|\    /|\
                                 / | \  / | \   / | \  / | \
```

Conventionnellement, la racine de l'arborescence globale porte le nom de `/`.

5.10 Désignation des objets

Jusqu'à présent, pour désigner un objet, nous nous sommes placés dans la situation la plus simple possible, à savoir, à partir d'un répertoire, nous avons désigné des objets de ce répertoire, en utilisant simplement le nom de l'objet. Mais comment faire pour désigner à partir d'un répertoire, un objet ne se trouvant pas dans ce répertoire? La désignation se fera à l'aide de ce que l'on appelle un *nom-de-chemin* et qui est tout simplement la description du chemin qu'il faut parcourir dans l'arborescence, pour, à partir d'un certain répertoire, atteindre l'objet à désigner.

[En anglais, nom-de-chemin se dit “pathname”]

Il existe deux types de nom-de-chemin selon le répertoire qui est pris comme point de départ.

- Si le répertoire de départ est le répertoire courant, on parlera de *nom-de-chemin relatif* (sous entendu au répertoire courant).
- Si le répertoire de départ est la racine de l'arborescence UNIX, on parlera de *nom-de-chemin absolu*.

Prenons des exemples. Supposons que de votre répertoire d'origine, vous désiriez désigner le fichier `chap1` dans le répertoire `livre`.

- Désignation à l'aide d'un nom-de-chemin relatif. Le chemin à parcourir depuis le répertoire courant comprend un seul intermédiaire: le répertoire `livre`, et le nom-de-chemin relatif de `chap1` sera donc `livre/chap1`. (On voit que l'on utilise de manière conventionnelle le caractère `/` comme séparateur des éléments du nom-de-chemin. Si on désire lire le contenu de ce fichier, on tapera donc:

  `more livre/chap1` *Return*

- Désignation à l'aide d'un nom-de-chemin absolu.

  Le chemin à parcourir depuis la racine de l'arborescence UNIX consiste à descendre sur le répertoire `users`, puis sur le répertoire `user1` (en supposant que vous soyez `user1`), puis sur le répertoire `livre`.

  D'autre part, pour faire la distinction entre nom-de-chemin relatif et nom-de-chemin absolu, il a été décidé conventionnellement qu'un nom-de-chemin absolu commencerait par le signe `/`.

  Le nom-de-chemin absolu du fichier `chap1` est donc `/users/user1/livre/chap1`. Vous pouvez lire ce fichier avec la commande:

  `more /users/user1/livre/chap1` *Return*

  Faites-le pour vérifier que ca marche.

Entraînez vous à utiliser nom-de-chemin relatifs et absolus pour désigner des fichiers se trouvant n'importe où dans l'arborescence. Si vous vous trouvez dans votre répertoire d'origine essayez d'atteindre le fichier `chap1` du répertoire `livre` d'un autre utilisateur que vous à l'aide d'un nom-de-chemin relatif et à l'aide d'un nom-de-chemin absolu. On rappelle qu'un répertoire père se désigne par `..`

5.11 Comment connaître son répertoire courant

Quand on se déplace dans l'arborescence, il arrive parfois que l'on soit perdu, et que l'on ne sache plus sur quel répertoire on se trouve. Dans ce cas, il existe une commande permettant de savoir quel est le répertoire courant, c'est la commande `pwd` (Print Working Directory). Tapez cette commande, et vous interprèterez sans peine ce qu'elle va vous répondre.

Chapter 6

L'interpréteur de commandes

6.1 Introduction

Lorsque vous travaillez sur un terminal connecté à une machine UNIX, dès que vous avez terminé la phase de login, vous dialoguez avec le système, ou plus précisément avec l'interpréteur de commandes. C'est l'interpréteur de commandes qui vous permet d'exécuter les commandes comme `ls`, `cp`, etc... Dans le jargon UNIX, un interpréteur de commande porte le nom de *shell*. Selon les versions d'UNIX, il existe différents shells, les principaux étant:

- `sh` dit le Bourne shell, qui est le shell standard de UNIX version ATT.
- `ksh` dit le Korn shell, version améliorée de `sh`, compatible avec celui-ci.
- `csh` dit le C shell, qui est le shell standard de UNIX Berkeley.
- `tcsh` version améliorée de `csh`, compatible avec celui-ci.

Tous ces shells sont historiquement issus du shell `sh`. Ils sont incompatibles entre eux, mais à cause de cette filiation, ils ont une base commune. Dans ce chapitre, nous ne parlerons que des caractéristiques les plus basiques de ces shells, et qui sont de ce fait communes aux quatre.

Un shell est un énorme programme offrant à l'utilisateur un grand nombre de services, nous allons passer en revue les services les plus fondamentaux.

Attention

Nous supposerons dorénavant que l'utilisateur a compris que toute commande doit être terminée par *Return*, nous ne l'indiquerons donc plus dans les exemples.

6.2 Exécution d'un programme

Le service le plus important rendu par le shell est la capacité qu'il a de lancer l'exécution d'un programme. Lorsque l'utilisateur désire exécuter un programme, il lui suffit de taper le nom du fichier qui contient ce programme.

Manipulation

Placez vous dans le répertoire `shell`. Faites `ls` pour vérifier que se trouve dans ce répertoire un fichier qui a pour nom `bravo`. Ce fichier contient un programme dont le but est d'imprimer un petit message. Activez-le en tapant:

```
bravo
```

6.3 Interruption de programme

Il arrive souvent qu'un programme, par suite d'une erreur de programmation, ne se termine jamais. Il est donc nécessaire que l'utilisateur puisse l'interrompre de manière à pouvoir continuer de travailler. Pour ce faire, le shell reconnaît un caractère particulier *Control C* comme étant l'ordre de tuer le programme qui est en train de s'exécuter.

Manipulation

Faites `ls` pour vérifier l'existence dans le répertoire courant d'un fichier ayant pour nom `boucle`. Ce fichier contient un programme dont l'exécution ne finit jamais. Lancez le en tapant:

```
boucle
```

après quelques secondes d'exécution, tapez *Control C* et vérifiez que le programme a bien été interrompu.

6.4 Génération de noms

A. Dans les exemples d'utilisation des commandes que nous avons donnés, une commande n'admettait en paramètre qu'un ou deux noms de fichier. Cependant, certaines commandes peuvent accepter en paramètre une liste comportant un nombre quelconque de noms de fichiers. Prenons le cas de la commande `ls`. Nous l'avons utilisée jusqu'à présent sans lui fournir en paramètre de nom de fichier, mais il est possible de l'activer en lui donnant en paramètres une liste de noms de fichiers.

Manipulation

Faites:

```
ls bravo.c boucle.c
```

et vérifiez que `ls` a bien fait le travail qu'on lui a demandé à savoir lister les noms des fichier `bravo.c` et `boucle.c`. Utilisé de cette manière, cette possibilité ne présente aucun intérêt. Par contre, tapez:

```
ls -l bravo.c boucle.c
```

pour vous rendre compte que cette forme est intéressante pour obtenir certains renseignements (par exemple date de la dernière modification) sur un sous-ensemble des fichiers du répertoire courant.

B. Il existe comme cela beaucoup de commandes qui acceptent en paramètre une liste de noms d'objets (fichiers ou répertoires). Un des grands services offert par le shell est donc un moyen puissant de désignation d'ensemble de noms d'objets. L'idée consiste à permettre à l'utilisateur d'écrire non pas strictement les noms des objets qu'il veut référencer, mais un modèle de ces noms. Le shell remplacera dans la commande le modèle par l'ensemble des noms des objets du répertoire courant qui sont conformes au modèle. Ceci se réalise à l'aide de certains caractères qui ont une signification spéciale.

- le caractère `?` dans un modèle, désigne n'importe quel caractère. Le modèle `fic?` sera donc remplacé par la liste des noms des objets du répertoire courant dont le nom comporte 4 caractères dont les trois premiers sont `fic`.
- le caractère `*` dans un modèle, désigne n'importe quelle chaîne de caractères (même vide), de telle sorte que le shell remplacera `fic*12` par la liste des noms des objets du répertoire courant dont le nom commence par `fic`, se termine par `12`, avec n'importe quoi entre les deux.
- une suite de caractères entourés par `[` et `]` (comme par exemple `[abc]`) désigne un seul des caractères de la suite. Par exemple, le shell remplacera le modèle `fic[123456789]` par la liste des noms des objets du répertoire courant dont le nom commence par `fic` et se termine par un chiffre.
- deux caractères séparés par `-` et entourés par `[` et `]` (comme par exemple `[a-z]`), désigne un seul des caractères se trouvant lexicalement entre les deux caractères. Par exemple, le modèle `fic[1-9]` est équivalent à `fic[123456789]`.

Exercice

Placez vous dans le répertoire `noms`, et utilisez la commande `ls` pour vérifier que ce répertoire contient bien les fichiers suivants:

```
aa ab1 bb cc cd2 ef3 f.c gg.c gh4 hhh.c ij5
```

Tous ces fichiers sont vides, nous allons uniquement utiliser leurs noms pour vous entraîner à l'aide de la commande `ls` à désigner des sous-ensembles de ces fichiers.

- comment désigner seulement `aa bb cc`?
- comment désigner seulement `f.c gg.c hhh.c`?
- comment désigner seulement `ab1 cd2 ef3 gh4`?

6.5 Redirection des Entrées Sorties

Un programme effectue une transformation sur de l'information: il lit de l'information en entrée, effectue un traitement, et donne de l'information en sortie. Schématiquement, nous pouvons représenter les choses de la manière suivante:

```
                    ---------
                    |       |
entree   --->  |  Pg   |  --->   sortie
                    |       |
                    ---------
```

Le programme lit l'information sur *l'entrée standard*, écrit de l'information sur *la sortie standard*, sachant que l'entrée standard est généralement affectée au clavier, et la sortie standard est généralement affectée à l'écran.

Il existe cependant de nombreuses situations ou on éprouve le besoin de faire travailler un programme en lui faisant lire l'information non pas à partir du clavier mais à partir d'un fichier, et/ou en lui faisant écrire l'information en sortie non pas sur l'écran mais dans un fichier.

Quand on réalise une telle opération, on dit que l'on a redirigé l'entrée et/ou la sortie.

Le shell permet de réaliser cette redirection de manière très simple. Quand on désire exécuter la commande *commande* en redirigeant son entrée sur le fichier *nom-de-fichier-entrée*, il suffit de taper:

commande `<` *nom-de-fichier-entrée*

Quand on désire exécuter la commande *commande* en redirigeant sa sortie sur le fichier *nom-de-fichier-sortie*, il suffit de taper:
commande `>` *nom-de-fichier-sortie*

Si on désire rediriger l'entrée et la sortie, il suffit de taper:
commande `<` *nom-de-fichier-entrée* `>` *nom-de-fichier-sortie*

Exercice 1: redirection de la sortie

Pour cet exercice, nous allons utiliser une nouvelle commande, la commande `cat`, qui admet en paramètre un ensemble de noms de fichiers, et qui a pour effet de sortir sur la sortie standard, la concaténation du contenu des fichiers.

Revenez dans le répertoire `shell`, et regardez par `more` le contenu des fichiers `m1` et `m2`, puis exécutez la commande `cat m1 m2` pour bien comprendre le fonctionnement de `cat`.

Maintenant, réexécutez `cat` de fac/on à mettre la concaténation des contenus de `m1` et `m2` dans un fichier de nom quelconque, par exemple `resu1`, et vérifiez en suite à l'aide de `more`, que le fichier `resu1` contient bien le résultat espéré.

Exercice 2: redirection de l'entrée

Pour cet exercice, nous allons utiliser une commande de comptage. Cette commande a pour nom `wc` (pour Word Count), et est capable de compter non seulement les mots, mais également les caractères et les lignes du texte qui lui est fourni en entrée standard.
`wc -c` compte les caractères
`wc -w` compte les mots
`wc -l` compte les lignes.

On va d'abord utiliser cette commande sans rediriger son entrée. Tapez:

```
wc -w
```

A ce moment, le programme de comptage est en attente sur son entrée standard, c'est à dire le clavier, tapez donc quelques mots, et terminez par *Control D* juste après un *Return* pour indiquer la fin des données. Après cela, `wc` doit vous indiquer le nombre de mots que vous avez tapés. Utilisé de cette manière, `wc` ne sert à rien, mais on peut l'utiliser, en redirigeant son entrée, pour compter le nombre de mots (par exemple) contenu dans un fichier.

Utilisez `wc -w` pour compter le nombre de mots du fichier `chap1` qui se trouve dans votre répertoire d'origine, sans quitter le répertoire `shell`.

Redirection de l'entrée et de la sortie

Comptez le nombre de mots du fichier `chap1`, en envoyant le résultat dans le fichier `resu2`. Vérifiez à l'aide de `more` que `resu2` contient bien le résultat espéré.

6.6 Notion de *pipe*

Nous avons vu qu'un programme lisait de l'information sur son entrée standard, effectuait un certain traitement et écrivait le résultat sur sa sortie standard. Il arrive très souvent que l'on désire faire coopérer deux programmes de manière à ce que le second effectue son travail sur l'information produite par le premier selon le schéma suivant:

```
                 ---------          ---------
                 |       |          |       |
     entree ---> |  Pg1  | -------> |  Pg2  | --->  sortie
                 |       |          |       |
                 ---------          ---------
```

Une telle organisation est ce que l'on appelle communément du travail à la chaîne.

Pour réaliser cela, le système établi une espèce de tuyau à information entre les deux programmes, et connecte la sortie du premier programme à l'entrée du tuyau, et l'entrée du second programme à la sortie du tuyau. Dans le jargon UNIX, un tel tuyau à information porte le nom de *pipe*.

On a alors le schéma suivant:

```
            ---------         pipe          ---------
            |       |     -------------     |       |
entree ---> |  Pg1  | ---> |           | ---> |  Pg2  | ---> sortie
            |       |     -------------     |       |
            ---------                       ---------
```

Le shell permet de créer deux programmes s'exécutant selon une telle méthode. Il suffit de taper les textes d'activation des deux commandes séparés par le signe conventionnel `|` qui demande l'établissement d'un pipe. On aura donc:
commande1 `|` *commande2*

Manipulation

Pour cet exercice nous allons voir une nouvelle manière d'utiliser la commande `more`. Jusqu'à présent, nous avons utilisé la commande `more` en lui fournissant en paramètre le nom d'un fichier que l'on voulait voir s'afficher à l'écran, l'affichage se bloquant page par page. On peut utiliser `more` en ne lui donnant pas de nom de fichier en paramètre, auquel cas, il va afficher sur l'écran ce qu'il lira à partir de son entrée standard. Ceci a pour conséquence, que si on exécute un programme, en envoyant sa sortie standard dans l'entrée de `more`, la sortie du programme sera bloquée page par page.

Relançons le programme `boucle` que l'on a déjà utilisé, mais cette fois-ci en le faisant collaborer avec `more` de la manière suivante:

```
boucle | more
```

vérifiez que `more` bloque bien la sortie du programme `boucle`, puis stoppez cette exécution par *Control C*.

Exercice

Faites coopérer `ls` et `wc` pour compter le nombre d'objets qui se trouvent sur votre répertoire courant.

6.7 Les variables

Le shell gère des variables, c'est à dire des couples (nom,valeur).

1. Une variable du shell n'a pas besoin d'être déclarée, elle prend existence dès qu'on lui affecte une valeur. Pour affecter une valeur à une variable, il faut émettre la commande:
   nom-de-variable`=`*valeur*

   Attention

   Ne pas mettre de blanc entre *nom-de-variable* et le signe `=`, ni entre le signe `=` et *valeur*. Exemple:

   ```
   dir1=/users/cassagne/enseignement
   ```

2. La rérérence à la valeur d'une variable se fait par `$` *nom-de-variable*. Exemple:

   ```
   $ echo $dir1
   /users/cassagne/enseignement
   ```

3. Si on désire connaître l'ensemble des variables et leurs valeurs à un instant donné, on utilise la commande `set`. Emettez cette commande, vous voyez que le système a créé un certain nombre de variables en plus de celles que vous avez pu créer.

Les variables sont un moyen agréable de réaliser des abréviations. Supposons que l'on prévoie d'avor à plusieurs reprise à référencer le fichier `/usr/include/stdio.h`, on peut se définir la variable `std=/usr/include/stdio.h`. Il sera ensuite agréable d'utiliser `$std` à la place du nom véritable.

Chapter 7

Eléments de réalisation de programme

Par réalisation de programme, nous entendons réalisation au sens pratique du terme, c'est à dire de création de fichier contenant un programme, de compilation et de mise au point. Sur chacune de ces questions, nous ne donnerons qu'un minimum permettant au lecteur de commencer à travailler.

7.1 Création d'un fichier contenant un programme

Cette création se fera à l'aide de l'éditeur `vi` auquel on passe en paramètre le nom d'un fichier inexistant; `vi` créera le fichier vide, et l'utilisateur n'aura plus qu'a taper l'ordre `i` (insertion) et ensuite taper le texte de son programme.

Un programme doit absolument être indenté de manière à ce que la mise en page soit porteuse d'information. Cette information ne sera pas utilisée par le compilateur, mais sera un élément précieux pour les programmeurs.

De manière pratique, un programme ne doit pas ressembler à ça:

```
function determax(list : tpnoeud) : tpnoeud;
var max : integer; lmax,lcour : tpnoeud;
begin
if list = nil then lmax := nil
else begin
lmax := list;
max := list^.valeur;
lcour := list^.suivant;
while (lcour <> nil) do
begin
if lcour^.valeur > max
then begin
lmax := lcour; max := lcour^.valeur;
end;
lcour := lcour^.suivant;
end;
end;
```

```
determax := lmax;
end;
```

mais plutôt à c/a:

```
function determax(list : tpnoeud) : tpnoeud;
var
max : integer;
lmax,lcour : tpnoeud;
begin
  if list = nil
  then lmax := nil
  else begin
     lmax := list;
     max := list^.valeur;
     lcour := list^.suivant;

      while (lcour <> nil) do
      begin
         if lcour^.valeur > max
         then begin
            lmax := lcour; max := lcour^.valeur;
         end;
         lcour := lcour^.suivant;
      end;
   end;
   determax := lmax;
end;
```

Cet exemple est très petit, et voyez cependant à quel point la version non indentée est illisible.

7.1.1 Paramètrage de l'éditeur

Il est possible de paramétrer l'éditeur de manière à rendre l'indentation extrêmement aisée à écrire. Il suffit pour cela, d'émettre à `vi` les commandes suivantes:

```
:set ai
:set sw=3
```

La première commande positionne l'indicateur ai (auto indentation). La deuxième commande donne la valeur 3[1] à l'indicateur sw (shift width).

Grâce à ces deux commandes, chaque fois que l'utilisateur tapera *Return* la nouvelle ligne débutera à l'alignement de la ligne précédente.

1. Pour les lignes devant être décalées à droite par rapport à la ligne précédente, l'utilisateur tapera *Control T* après le *Return*.

[1] L'utilisateur peut taper la valeur de son choix à la place de 3, mais il faut que le décalage à droite ne soit ni trop petit (pour être efficace), ni trop grand (pour ne pas être trop décalé à droite). Les valeurs 3,4 ou 5 semblent raisonnables.

2. Pour les lignes devant être décalées à gauche par rapport à la ligne précédente, l'utilisateur tapera *Control D* après le *Return*.

Exemple

Pour obtenir le résultat suivant:

```
begin
   if lcour^.valeur > max
   then begin
      lmax := lcour; max := lcour^.valeur;
   end;
   lcour := lcour^.suivant;
end;
```

il faudra taper:
`begin` *Return*
Control T `if lcour^.valeur > max` *Return*
`then begin` *Return*
Control T `lmax := lcour; max := lcour^.valeur;` *Return*
Control D `end;` *Return*
`lcour := lcour^.suivant;` *Return*
Control D `end;` *Return*

7.1.2 Fichier d'initialisation de `vi`

Il existe un moyen permettant d'éviter de taper les deux commandes `:set` à chaque appel de `vi`.

A l'appel, `vi` teste l'existence dans le répertoire courant, d'un fichier de nom `.exrc`. Si un tel fichier n'existe pas, `vi` ne fait rien de spécial. Par contre, si un tel fichier existe, `vi` lit le contenu du fichier qu'il interprète comme une suite d'ordres. Ce fichier est donc l'endroit idéal pour y mettre les deux commandes `set` (sans le `:`) qui seront automatiquement exécutées à chaque appel de `vi`.

7.2 Compilation pascal

La compilation consiste à traduire un fichier contenant un programme écrit en pascal, en un fichier contenant un programme de même sémantique, mais écrit en instructions machines. Le programme écrit en Pascal est généralement appelé programme source, et le programme écrit en instructions machine, programme objet.

Le compilateur Pascal sur cette machine exige que le fichier source porte un nom se terminant par `.p` (p pour Pascal).

La commande d'appel du compilateur est:
`pc` *fichier-source* `-o` *fichier-objet*
ce qui peut s'exprimer par: compiler le fichier *fichier-source* et mettez le résultat (o pour output) dans le fichier *fichier-objet*.

Il est d'usage de prendre comme nom de fichier objet la racine du nom du fichier source, bien que ce ne soit pas une obligation.

Exemple

Supposons que l'on ait créé un fichier source pascal de nom `pgcd.p`, on le compilera par la commande:

```
pc pgcd.p -o pgcd
```

7.3 Exécution d'un programme

Pour lancer l'exécution du programme objet, il suffit de taper le nom du fichier qui le contient.

Exemple

Si on reprend l'exemple précédent, on exécutera le programme objet en tapant:

```
pgcd
```

Chapter 8

Le shell tcsh

8.1 Présentation du shell tcsh

Le shell (`tcsh`) est un interpréteur de commandes disposant d'un certain nombre de caractéristiques qui en font un outil beaucoup plus confortable à utiliser que `sh`. Malheureusement, `tcsh` n'est pas compatible avec `sh` ; seules les caractéristiques les plus basiques restent inchangées, en particulier :

1. la génération des noms,
2. la redirection de l'entrée et la sortie d'un programme,
3. le pipe.

8.2 Les variables

L'affectation d'une valeur à une variable se fait par `set` *nom-de-variable* `=` *valeur*. Cette fois ci la syntaxe est libre, on peut mettre un nombre indifférent de blancs entre les divers éléments. Exemple:

```
set ens=/users/cassagne/enseignement
set src     =         /users/cassagne/src
```

Comme avec `sh`, la référence à la valeur d'une variable se fait par `$`*nom-de-variable*, et la liste des variables avec leurs valeur est obtenue par `set`.

8.3 Terminaison des noms de fichiers

Le shell `tcsh` est capable de terminer de lui-même les noms de fichiers dès qu'il dispose qu'un début non ambigu. Pour donner au shell l'ordre de terminer un nom de ficher, il suffit de taper le caractère *tab* (tabulation.)

Supposons que le répertoire courant contiennent les fichiers suivants:

```
LISEZ_MOI       brouillon       courrier        shell
bravo.c         chap1           livre
```

Si on veut faire `vi chap1`, il suffit de taper `vi ch` *tab*, le shell complétera en `vi chap1`.

Si au moment de l'émission de la commande *tab*, le début du nom est ambigu, le shell affiche tous les noms possibles puis réaffiche le début de la commande. Il suffit de taper juste ce qu'il faut de caractères pour rendre le nouveau début non ambigu et de retaper *tab* pour obtenir le nom désiré.

8.4 mécanisme d'historique

8.4.1 Présentation

Il arrive très souvent que l'on ait a utiliser tout ou partie du texte d'une commande précédemment tapée, soit qu'on veuille refaire une commande, soit qu'on veuille légèrement modifier une commande erronée. Le shell `tcsh` permet de réaliser cela de manière très agréable. Toutes les commandes que l'on tape sont mémorisées dans un fichier d'historique. L'utilisateur contrôle le nombre de commandes mémorisées en affectant une valeur à la variable `history`. Pour mémoriser les 100 dernières commandes, il faut taper:

```
set history=100
```

Pour obtenir l'historique, il suffit de taper:

```
history
```

ou, si l'on ne désire que les 20 dernières commandes, de taper:

```
history 20
```

8.4.2 Réexécution de commandes

Dans l'historique, les commandes sont précédées d'un numéro et de l'heure à laquelle la commande à été émise. Le numéro est dénommé *numéro d'événement* et il est très pratique pour réexécuter une commande précédemment émise. Supposons que nous ayons l'historique suivant:

```
124  10:30   vi data
125  10:31   ls
126  10:31   vi essai.c
127  10:32   cc -o essai essai.c
128  10:32   essai
```

Si on désire réexécuter la commande `cc -o essai essai.c`, il suffit de taper `!127`. Le caractère `!` est un caractère spécial permettant d'indiquer au shell un numéro d'événement.

Il existe d'autres manières de référencer une commande de l'historique grâce au caractère `!` :

- `!!` désigne la dernière commande émise.
- `!`*chaîne* désigne la plus récente commande ayant *chaîne* comme préfixe. Dans l'exemple précédent, il suffit de taper `!c` pour réexécuter la commande `cc -o essai essai.c`
- `!?`*chaîne*`?` désigne la plus récente commande comportant *chaîne* (n'importe où.) Toujours dans le cas de l'historique précédent, pour ré-exécuter `vi data`, il suffit de taper `!?data?`.

Remarque

Ce mécanisme est très efficace quand on fait du travail répétitif, ce qui est souvent le cas. Par exemple un programmeur en train de mettre au point un programme boucle sur les 3 phases : édition du fichier, compilation, exécution.

8.4.3 Réexécution de commandes avec modification

Il arrive souvent que la commande que l'on vienne d'émettre soit erronée, mais que l'erreur ne porte que sur un très petit nombre de caractères (cas typique: la faute de frappe). Il est donc intéressant de pouvoir modifier une commande précédemment émise. Le shell `tcsh` permet de réaliser la sélection et la modification de commande à la manière de l'éditeur `vi`.

Du point de vue de la frappe des commandes, tout se passe comme si on était en train d'utiliser `vi` en mode input pour créer le fichier d'historique. Lorsqu'on désire récupérer du texte d'une commande précédemment tapée, il suffit de taper *Escape* pour sortir du mode input et passer en mode commande de `vi`. A partir de ce moment, on peut utiliser un certain nombre de commandes de `vi` pour se déplacer dans le fichier d'historique et faire des modifications.

- pour retrouver la commande comportant le texte intéressant, on peut utiliser:

  1. les ordres `-` `+` (ou les flèches) pour respectivement remonter et redescendre d'une ligne dans l'historique.
  2. les ordres `/`*chaîne* et `?`*chaîne* pour retrouver la première occurrence de *chaîne* respectivement en remontant et en descendant dans le fichier d'historique.
  3. l'ordre `n` pour recommencer la recherche précédente.

  A chaque fois, l'éditeur affiche sur l'écran la commande ainsi sélectionnée.

- pour se déplacer dans la ligne sélectionnée, on peut utiliser:

  1. les ordres `w` et `b` pour se déplacer de mot en mot respectivement vers l'avant et vers l'arrière.
  2. les ordres `0` et `$` pour aller respectivement au début et à la fin de la ligne.

- pour modifier la ligne sélectionnée, on peut utiliser pratiquement toutes les commandes de modification de `vi`, en particulier:

  1. `i` et `a` pour passer en mode *input*.
  2. `r`, `R`, `cw` et `C` pour remplacer du texte.
  3. `x`, `dw` et `D` pour effacer du texte.

Une fois que la ligne courante a la forme désirée, il suffit de taper *Return* pour que cette ligne soit exécutée en tant que commande. C'est la ligne entière, telle qu'elle est affichée sur l'écran, qui sera exécutée, quelque soit la position du curseur dans la ligne.

Exemple

Supposons qu'on tape la commande suivante:

```
grep cast ~cassagne/documentation/C/chapitre[0-9].tex
```

pour chercher des occurrences du mot “cast” dans les fichiers `chapitre[0-9].tex` se trouvant dans l'arborescence de l'utilisateur `cassagne`. Supposons qu'on fasse une faute de frappe, et qu'on tape le nom de répertoire `C` en minuscule au lieu de le taper en majuscule. Pour corriger l'erreur, il suffit de taper *Escape*, `-` pour se positionner sur la commande précédente, taper 2 fois `b` pour se positionner sur le `c` fautif, puis `rC` pour le corriger. Il ne reste plus qu'à taper *Return* pour que la commande ainsi modifiée soit réexécutée.

8.5 Alias

1. On peut définir un nom comme étant un alias (c'est à dire une abréviation) pour une chaîne de caractères. Si par exemple, on a souvent besoin d'exécuter un programme de nom `analyse` on peut émettre la requête suivante:

   ```
   alias ana analyse
   ```

   ce qui a pour effet de faire mémoriser par `tcsh` que lorsque le **premier** mot d'une commande est `ana`, il doit le remplacer par `analyse`. Une activation du programme `analyse` pourra donc se faire par:

   ```
   ana fic1 fic2
   ```

   (en supposant que `fic1` et `fic2` soient deux arguments nécessaires à l'exécution du programme.)

2. On peut faire en sorte qu'un alias soit remplacé par une chaîne contenant des blancs, mais cela nécessite de mettre entre *quotes* le texte de l'alias. Supposons que l'on prévoie d'avoir à recompiler plusieurs fois le fichier `pgcd.p`, on peut se définir un alias de la manière suivante:

   ```
   alias compil 'pc pgcd.p -o pgcd'
   ```

   on peut ensuite lancer la compilation en tapant simplement:

   ```
   compil
   ```

3. Lorsque l'expansion d'un alias a été réalisée, le premier mot de la commande obtenue n'est plus susceptible de subir à nouveau une expansion d'alias. On peut donc sans crainte définir des alias “récursifs”, comme par exemple:

   ```
   alias ls 'ls -F'
   ```

8.6 La convention tilde

1. Après l'expansion d'un éventuel alias, tous les mots d'une commande sont scrutés pour déterminer si ils commencent par le caractère `~`. Si oui, le mot entier (ou le début du mot jusqu'au premier caractère `/`) est interprété comme un nom d'utilisateur, et remplacé par le *répertoire d'origine* de cet utilisateur. Supposons qu'on désire lire le contenu du fichier `data` se trouvant dans le *répertoire d'origine* de l'utilisateur `rouverol`, on peux émettre la commande:

   ```
   more ~rouverol/data
   ```

   que `tcsh` transformera en:

   ```
   more /users/rouverol/data
   ```

2. Si le nom d'utilisateur suivant le signe `~` est vide, le nom pris par défaut sera celui de l'utilisateur lui-même. Supposons que l'utilisateur `cassagne` désire lire un fichier se trouvant dans son *répertoire d'origine*, il peut émettre la commande:

   ```
   more ~/data
   ```

   que le shell transformera en:

   ```
   more /users/cassagne/data
   ```

Chapter 9

La messagerie électronique

9.1 Envoi d'un message

Pour envoyer un mail à *destinataire*, tapez:

`mailx` *destinataire* *Return*
`Subject:` *sujet-du-message* *Return*
première-du-texte-du-message *Return*
...
dernière-ligne-du-message *Return*
Control D

destinataire peut être un nom d'utilisateur si le destinataire possède son login sur la même machine que celle où on envoie le message. Dans le cas contraire, *destinataire* doit être de la forme *utilisateur*`@`*machine*.

ex:

```
mailx rouverol
mailx cassagne@ensisps4
```

Au cours de l'écriture du message si vous tapez `~?` en début de ligne vous aurez une explication succincte sur les commandes de `mailx` en mode envoi de message.
En particuliers l'appel de `vi` pour corriger ou mettre en page le message se fait à l'aide de la commande `~v` en début de ligne. Il est recommandé de passer en mode `vi` dès que la ligne *sujet* a été tapée, à cause des facilités de modifications qu'offre `vi`. Pour terminer le message, il faudra d'abord sortir de `vi` (par `ZZ`) puis sortir de `mailx` (par *Control D*).

9.2 Lecture et gestion de sa boite aux lettres

Taper `mailx`

Si vous n'avez pas de message à lire la commande `mailx` répond :

`No mail for` *nom-utilisateur*

Si vous avez des messages à lire la commande `mailx` imprime la liste de la partie sujet des messages de votre boite aux lettres et se met en attente de commande.
ex:

```
mailx
mailx version 20.0 02/05/85  Type ? for help.
"/users/spool/mail/rouverol": 2 messages
>   1 skfp              Wed Oct 26 11:20   12/274   probleme avec open
2 guitin              Wed Oct 26 13:21   13/283   annee speciale
?
```

- En tapant ? vous aurez une explication succincte sur les différentes commandes de `mailx` en mode lecture de message.
- Si vous taper *Return* le texte du message courant de votre boite aux lettres sera imprimé.
- Les commandes les plus couramment utilisés sont les suivantes :
  - `s` *nom-de-fichier* *Return* pour sauver un message dans un fichier
  - `r` *Return* pour répondre à l'expéditeur du message.
  - `R` *Return* pour répondre à l'expéditeur ainsi qu'à tous les destinataires du message.
  - `d` *Return* pour détruire un message déjà lu
  - *no-message* *Return* pour imprimer le message de numéro *no-message*
  - `h` *Return* pour imprimer la liste de la partie sujet des messages de la boite aux lettres
  - `q` *Return* pour sortir de l'éditeur en sauvant les messages non détruits dans le fichier `mbox` sur votre *répertoire d'origine*
  - `x` *Return* pour sortir de `mailx` sans sauver de message dans le fichier `mbox`. Les messages resteront cependant mémorisés par le système (même ceux que l'utilisateur aura détruits par `d`!) et pourront être relus ultérieurement en rappelant `mailx`.

Les messages stockés dans le fichier `mbox` sur le *répertoire d'origine* peuvent être relus avec la commande :

```
mailx -f mbox
```

Chapter 10

Les News

Les News sont une sorte de forum, où des individus du monde entier discutent de tous les sujets possibles : d'informatique, de musique, de sport, etc...
Ces sujets sont groupés par thèmes et forment des *newsgroups.*
A l'intérieur de chaque newsgroup, se trouvent des messages contenant l'information destinée au mone extérieur, nommés *articles.*
Aprés avoir lu ce chapitre, vous saurez comment lire les articles d' un newsgroup et poster un article dans un newsgroup.

10.1 Comment ça marche

Les News sont un service offert par un ensemble de sites (pour la plupart universitaires) disséminés dans le monde entier et connectés via le réseau Internet. Chacun de ces sites dispose d'un ordinateur abonné aux News, c'est-à-dire qui recoit et transmet des articles postés par des utilisateurs. C'est un *serveur de News.*

En fait, un serveur de News n'envoit pas ses articles à tous les sites du monde entier, mais à un ou plusieurs sites, qui diffusent à leur tour ces informations à d'autres, etc... les serveurs de News forment donc entre eux une grande chaîne, et un article peut mettre plusieurs jours à arriver d'un bout à l'autre de cette chaîne.
Les articles reçus sont stockés sur un disque du serveur. Ils sont effacés périodiquement, sans quoi il n'y aurait plus assez de place pour en recevoir de nouveaux.

A l'ENSIMAG, le serveur de News est la machine `oldensun`).
Chaque étudiant de l'Ensimag peut donc avoir accès à toutes ces informations reçues du monde entier en interrogeant le serveur de news oldensun, à l'aide du programme *rtin.*

10.2 Lire et organiser les newsgroups avec `rtin`

Taper `rtin`.
La première fois que vous lancez `rtin`, celui-ci affiche une page d'explications sur son fonctionnement, mais les fois suivantes il affichera un resumé des commandes valides en bas de l'écran.
`rtin` crée également un répertoire nommé `News` dans votre répertoire courant, ainsi qu'un

fichier nommé `.newsrc`, dont nous verrons bientôt l'utilité.

Quittez cette page, la liste des newsgroups s'affiche à présent.
La première ligne de cette liste est en inversion vidéo. Cet attribut inversion vidéo indique le newsgroup sélectionné ou courant.

Il suffit de retenir les commandes suivantes :

- sélectionner un newsgroup dans la liste avec les flèches, on peut cependant se déplacer plus vite dans la liste en avant et en arrière avec la barre d'espace et la touche `b` (back)
- sélectionner un newsgroup avec la touche `/`. `rtin` invite alors à taper un nom (partiel ou entier).
- consulter les articles du newsgroup sélectionné en appuyant sur *Return* ou sur la touche de tabulation (son usage sera détaillé plus loin). `rtin` affiche alors la liste des articles du newsgroup, leur numéro d'ordre et leur titre. On sort avec la touche `q`.
- se désabonner d' un newsgroup, on peut supprimer le newsgroup courant de la liste initiale en appuyant sur la touche `u` (unsubscribe). En cas d'erreur on peut se réabonner immédiatement en appuyant `s` (les changements ne sont pris en compte qu'une fois que l'on quitte `rtin`)
- changer la place dans la liste initiale du newsgroup courant avec la touche `m` (move). `rtin` invite alors à indiquer un numéro d'ordre dans la liste.
- on quitte `rtin` avec la touche `q`

Exercez-vous à sélectionner le groupe `ensimag.etudiants.general`, et à le déplacer en tête de la liste.

Lors de la consultation des articles d'un newsgroup, l'article courant apparait en inverse vide. Les commandes principales sont les suivantes :

- changer l'article courant avec les fléches, ou plus rapidement avec la barre d'espace et la touche `b`
- lire l'article courant en appuyant sur *Return* ou la touche de tabulation, on en sort avec `q`
- visualiser la liste des réponses à un article avec la touche `l`. Nous reviendrons sur ce sujet dans le paragraphe suivant.
- sauvegarder l'article courant dans un fichier en appuyant sur `s` (save) puis sur `a` (article), puis nom de fichier, créé dans le répertoire `News` de votre répertoire principal. A la question "Process", répondre `n` (none). L'article sauvegardé pourra être relu plus tard avec `more`, un éditeur comme `vi` ou bien avec la commande `mailx -f` *nom-du-fichier*

  car les mails et les articles de newsgroup ont des structures (entêtes) similaires.

Essayez de lire un article dans un newsgroup (s'il n'y en pas, le message `*** No Articles ***` s'affichera en bas de l'écran, sortez par `q`), puis sortez du newsgroup et entrez-y à nouveau. Que remarquez-vous?

En effet `rtin` tient à jour la liste des newsgroups courants, et pour ceux auquels vous êtes abonnés, la liste des articles que vous avez déjà lus, tout cela dans un fichier, nommé `.newsrc` se trouvant dans votre répertoire principal, qui est lu à chaque appel de `rtin`. De plus, `rtin` vous informe de l'apparition de nouveaux groupes sur le serveur et vous propose d'y souscrire (répondre `y` ou `n`) à chaque appel, et met à jour votre .newsrc.

A propos de la touche de tabulation : Nous avons vu que les articles étaient éffacés regulièrement du disque où ils étaient stockés, pour des problèmes d'espace disque. Il peut arriver qu'un ou plusieurs articles soient éffacés, donc inaccessibles, mais qu'ils apparaissent toujours dans un newsgroup (nous verrons pourquoi dans le paragraphe suivant). Dans `rtin`, la touche de tabulation permet, contrairement à *Return*, de sauter ces articles, plutôt que de rester bloqué sur un message `Article unavailable`. C'est pourquoi il est préférable de l'utiliser.

10.3 Poster un article

Il faut tout de suite apporter une précision à propos des News à l'ENSIMAG. Le serveur de News oldensun est en bout de chaine de diffusion des news. Il reçoit mais ne diffuse pas vers l'exterieur. Les articles que vous posterez ne pourront être consultés que par les étudiants de ENSIMAG.

Il y a deux manières de poster un article dans un newsgroup. Il faut tout d'abord sélectionner un newsgroup, puis utiliser la commande `w` (write). L'editeur `vi` est alors lancé, et on édite son article comme un fichier quelconque.

La deuxième manière est de poster une réponse à l'article qu'on lit. On utilise la commande `f` (follow-up). De même, l'éditeur `vi` est lancé, et le texte de l'article auquel on veut répondre est recopié. Une fois posté, cet article apparaîtra dans la liste des réponses, accessible par la touche `l` dans un newsgroup.

Remarque : Si un ou plusieurs articles ou réponses sont éffacés, mais que des articles continuent à être postées en réponse, les articles éffacés continueront à apparaître dans le newsgroup, mais ne seront bien sur plus accessibles. Seules les réponses non éffacées seront encore consultables. La touche de tabulation permet alors de sauter ces articles fantômes, et d'afficher le premier article disponible.

Pour essayer : Un newsgroup local a été spécialement crée dans ce but, il se nomme `ensimag.etudiants.test`. N'hésitez pas à l'utiliser!

Contents