GUIDE SIMPLIFIÉ D'UTILISATION D'UNIX À L'INTENTION DES ÉTUDIANTS

C.-A. Brunet, D. Dalle et J.-M. Dirand

31 mai 2019

Département de génie électrique et de génie informatique
Faculté de génie
Université de Sherbrooke

Auteurs: C.-A. Brunet, D. Dalle et J.-M. Dirand
Version: 9 (31 mai 2019 à 10:55:14)

Ce document est réalisé avec l'aide de LaTeX.

Table des matières

1 Introduction

Le système d'opération Unix a été conçu au début des années soixante-dix. Depuis, il est devenu le système d'opération pour la très grande majorité des compagnies offrant une gamme de stations de travail (workstations), ceci malgré des architectures complètement différentes.

Ce texte est rédigé en vue d'un usage d'Unix ou en encore de Linux. D'ailleurs, dans ce document, les termes Unix et Linux sont utilisés de manière interchangeable.

Le système d'opération Unix gère toutes les ressources disponibles à l'usager, comme la mémoire, les disques durs, les imprimantes, etc. Il fournit une interface entre l'usager et ces ressources. Les paragraphes qui suivent donnent quelques caractéristiques générales d'Unix.

Le système Unix est multitâche. Unix supervise l'exécution de plusieurs tâches actives simultanément, généralement des programmes, appelés processus. Les ressources du système sont allouées selon un mécanisme de priorité ou de partage du temps.

Le système Unix est multi-utilisateur. Plusieurs utilisateurs ont accès simultanément aux ressources gérées par Unix. Ces usagers peuvent être reliés à Unix par des terminaux ou des stations de travail à l'aide du réseau ou tout autre dispositif de communication.

Le système Unix a des catégories d'usagers. Unix est un système multi-usager complexe à gérer (installation, maintenance, etc.). Un administrateur du système est requis. C'est l'administrateur qui supervise le bon fonctionnement du système et qui crée les comptes pour les nouveaux usagers du système. Nous pouvons distinguer au moins trois types d'accès. Le premier est le super-usager qui a les pleins pouvoirs. Il peut faire ce qu'il veut, où il veut et quand il veut. Cette catégorie est réservée exclusivement à l'administrateur du système. Le deuxième est l'accès de groupe. Les usagers réguliers sont généralement regroupés. Par exemple, tous les étudiants du cours GEN241 peuvent être associés au groupe nommé GEN241. Le troisième est l'accès usager. Un usager régulier qui utilise les ressources du système, mais dont les permissions sont restreintes selon ses besoins.

Le système de fichiers Unix est hiérarchisé sous forme d'arbre. Il est composé de répertoires (*directory*) et de fichiers, comme avec Windows. Généralement, chaque usager a son répertoire principal qui porte son nom d'utilisateur. Ce répertoire est utilisé à la guise de l'usager qui peut créer autant de fichiers et de sous-répertoires qu'il a d'espace autorisé. Comme plusieurs répertoires d'usagers sont présents sur le disque (il y a plusieurs usagers) et qu'ils sont visibles pour tous, des mécanismes de protection et de permissions sont disponibles afin d'empêcher ou de permettre l'accès aux répertoires et fichiers. Chaque usager peut décider des permissions (écriture, lecture et exécution) qu'il accorde sur ses biens personnels sur le système. Ces permissions se situent à trois niveaux : personnel, groupe et monde. L'usager peut donc décider des permissions qu'il s'accorde à lui-même, à son groupe et au reste du monde. Un usager peut décider d'interdire tout accès au monde extérieur et de permettre à son groupe (d'autres étudiants d'un même cours, par exemple) l'accès en lecture seulement.

2 Syntaxe de la ligne de commande

2.1 Syntaxe d'une commande

Le système Unix est sensible aux lettres majuscules et minuscules. Les commandes `passwd`, `PASSWD` et `Passwd` sont toutes interprétées comme des commandes différentes. Cette sensibilité est présente en tout temps. Les commandes suivent une syntaxe standard : le nom de la commande, les options et les paramètres dont voici un exemple type :

```
ls -l -F fichier1 fichier2 fichier3
```

Le premier mot est la commande (`ls`), elle comporte deux options (`-l` et `-F`) et trois paramètres (`fichier1`, `fichier2` et `fichier3`).

Les options vous permettent de spécifier un comportement désiré (qui n'est pas par défaut) pour la commande. Les paramètres, dans notre cas pour la commande `ls`, spécifient quels fichiers sont à traiter. L'ordre des options n'est (typiquement) pas significatif et lorsqu'une commande a plus d'une option elles peuvent être combinées avec un seul `-`. Par exemple, les commandes suivantes sont toutes équivalentes.

```
ls -l -F fichier1 fichier2 fichier3
ls -F -L fichier1 fichier2 fichier3
ls -Fl fichier1 fichier2 fichier3
```

La ligne de commande peut prendre plusieurs commandes d'un seul coup, il suffit de les séparer par un point virgule, par exemple :

```
date; cd; ls -l -F fichier1
```

2.2 Redirection de la sortie et de l'entrée standard

Par défaut la plupart des commandes Unix donnent leurs résultats sur la sortie standard, généralement l'écran. Nous disons généralement, car il est possible de rediriger cette sortie vers une autre à l'aide de l'opérateur de redirection de la sortie, le caractère `>`. Par exemple, au lieu d'avoir les résultats de la commande `ls` à l'écran, on peut rediriger la sortie dans un fichier avec la commande suivante :

```
ls > data.txt
```

Si le fichier spécifié est inexistant, en l'occurrence `data.txt`, il est créé, s'il existe les anciennes données sont perdues. L'opérateur `>>` permet de rediriger la sortie, tout comme l'opérateur `>`, mais ajoute les données à la fin du fichier spécifié (concaténation).

Par défaut la plupart des commandes Unix prennent leurs données sur l'entrée standard, généralement le clavier. Tout comme la sortie standard, l'entrée standard peut être redirigée à l'aide l'opérateur de redirection de l'entrée, le caractère `<`. Par exemple, la commande `cat` fait une copie de l'entrée standard dans la sortie standard, si aucun fichier n'est spécifié en paramètre. La commande suivante fait donc une copie de `fichier.in` dans `fichier.out`. Évidemment, ce n'est pas le seul moyen de copier un fichier comme nous verrons plus loin.

```
cat < fichier.in > fichier.out
```

2.3 Pipe

Avec le système Unix, il est possible d'utiliser la sortie d'une commande comme entrée d'un autre en les reliant par un *pipe*, dénoté par le caractère `|`. Par exemple, la commande

`cat data.txt` permet de faire défiler à l'écran le contenu du fichier `data.txt`. Si ce fichier est volumineux, il est avantageux d'utiliser la commande `more` afin d'afficher page par page les résultats de la commande à l'écran. Une façon d'y arriver est la commande suivante :
`cat data.txt | more`

Ainsi, le contenu du fichier `data.txt` n'est pas affiché directement à l'écran, mais est passé à la commande `more` et c'est `more` qui affiche le contenu du fichier à l'écran. La commande `more data.txt` aurait le même résultat.

2.4 Fonctionnement en arrière-plan

Si l'on fait suivre une commande par le caractère `&`, alors la commande lancée sera exécutée en *arrière-plan* et le message de sollicitation de la prochaine commande sera récupéré avant la conclusion de la commande lancée. Le fonctionnement en arrière-plan est particulièrement utile dans un environnement graphique où plusieurs programmes doivent être actifs simultanément.

3 Commandes de gestion de fichiers

La navigation dans les répertoires du système de fichiers d'Unix est généralement faite avec la commande `cd`. Pour spécifier un répertoire, il suffit de séparer les noms de répertoires d'un slash (`/`), par exemple `cd /usr/bin`. Voici quelques notes utiles sur la commande `cd` :

- Utilisée sans paramètres, elle vous ramène à votre répertoire principal.
- Le caractère tilde (~), est un synonyme de votre répertoire principal, par exemple `cd ~/GEN241`.
- Le caractère * est un *wildcard*. Par exemple, la commande `rm prog*` efface tous les fichiers, avec ou sans extension, ayant `prog` comme préfixe.

Le tableau 1 donne la liste des commandes principales pour la gestion de fichiers et de répertoires sous Unix. Les détails de chacune des commandes peuvent être obtenus à l'aide de la commande `man`, comme discuté dans la section suivante.

4 Obtenir de l'information

Traditionnellement, sous Unix, l'apprentissage se fait d'abord en faisant appel à la débrouillardise. Vous apprenez par vous-même. Ensuite, si après avoir fait plusieurs tentatives votre problème n'est pas résolu, alors vous ferez appel à un utilisateur plus expérimenté qui sera heureux (normalement) de vous conseiller. Évidemment, lorsque vous-même deviendrez un usager expérimenté, on s'attend à ce que vous aidiez les usagers moins expérimentés que vous.

Unix est équipé d'un énorme manuel de l'usager accessible en ligne. Avant de demander de l'aide, on s'attend à ce que vous ayez déjà consulté ce manuel à l'aide de la commande `man`. Par exemple, pour consulter à l'écran la documentation de la commande `cp`, tapez `man cp`. La commande `man` est l'une des manières les plus utilisées pour obtenir de l'information, à part une recherche sur internet, évidemment. L'avantage de cette commande, c'est que les informations affichées sont celles qui sont spécifiques à votre environnement.

Tableau 1: Commandes principales de gestion de fichiers d'Unix

Nom	Description
`chmod`	Commande permettant de modifier les droits d'accès au fichier (lecture, écriture et exécution)
`cmp`	Permet de comparer le contenu de deux fichiers
`cd`	Permet de changer de répertoire
`cp`	Permet de faire la copie d'un fichier
`diff`	Permet de comparer le contenu de deux fichiers et d'afficher les différences
`find`	Permet de rechercher des fichiers récursivement dans des répertoires
`grep`	Permet de trouver dans un fichier les lignes contenant un motif donné
`lp`	Permet d'imprimer des fichiers
`ls`	Permet d'afficher les fichiers et sous-répertoires si le paramètre spécifié est un répertoire, si le paramètre n'est pas un répertoire (un fichier) affiche le nom du fichier
`mkdir`	Permet de créer un sous-répertoire
`more`	Permet d'afficher un fichier page par page
`mv`	Permet de changer le nom d'un fichier (ou répertoire). Cette commande permet aussi de déplacer un ficher d'un répertoire à un autre
`pwd`	Permet d'afficher le répertoire courant
`rm`	Permet de supprimer un fichier
`rmdir`	Permet de supprimer un répertoire
`touch`	Permet de modifier la date et l'heure d'accès d'un fichier

Les informations sur internet ne sont pas toujours exactes, car elles ne reflètent pas nécessairement la version spécifique avec laquelle vous travaillez.

Les informations affichées par `man` remplissent souvent plus d'un écran et elle affiche alors les informations à l'aide de la commande `more` qui permet d'afficher le texte un écran à la fois et en faisant une pose entre chacun. La commande `more` affiche à l'écran du texte suivi d'un message ressemblant à celui-ci :

```
--More--(25%)
```

Ce message indique que 25% de l'information a été affichée. Plus d'information peut être lu en tapant un espace ou sur `enter`. On peut quitter `more` en tout temps en tapant sur la lettre `q` ou en faisant la séquence `ctrl-c`.

5 Compilation

5.1 Étapes de la compilation

La séquence de commande pour créer et exécuter une application est souvent la suivante : édition, compilation et exécution. Un éditeur comme `nedit` est l'outil de l'étape d'édition, la commande `g++` permet de réaliser de la compilation et l'exécution d'un programme que vous avez créé est gérée par les commandes d'Unix.

La compilation mérite d'être étudiée plus en détail. Cette partie peut être subdivisée en quatre étapes pour le langage C++ : prétraitement, compilation, assemblage et édition des liens. Cette séquence est montrée à la figure 1 et elle est détaillée dans ce qui suit.

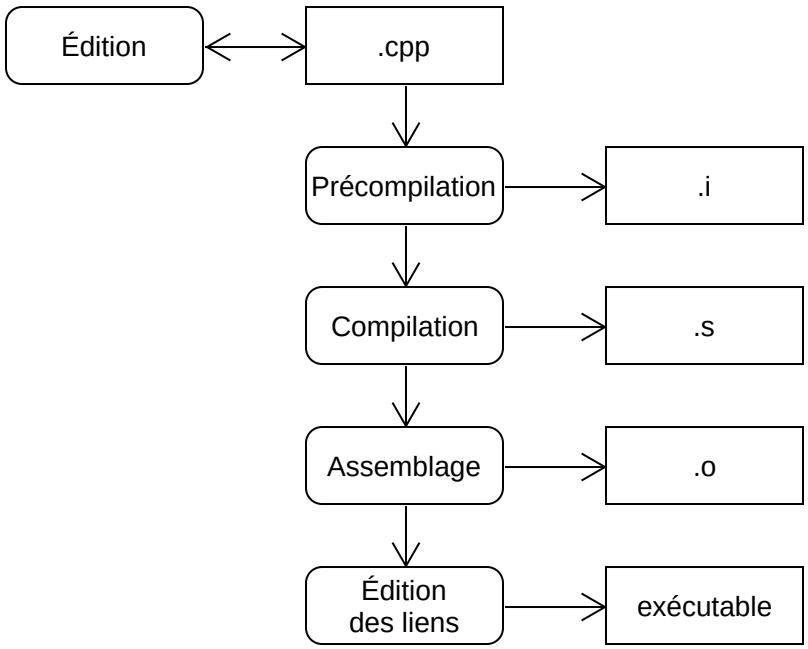

Figure 1: Compilation C++ sous Unix

1. La précompilation est faite par le préprocesseur C++ sur les fichiers de code (`.cpp`). Le préprocesseur traite, entre autres, les directives `#include` et `#define`. Il génère des fichiers avec l'extension `.i` lorsque la demande en est faite, car ces fichiers ne sont typiquement pas générés.
2. La compilation qui prend les résultats de l'étape précédente pour les compiler et générer du code en langage assembleur. Les fichiers en assembleur ne sont typiquement pas générés, mais lorsque la demande en est faite, ils ont souvent l'extension `.s`.
3. L'assemblage, qui est fait par l'assembleur, prend les résultats de l'étape précédente pour générer des fichiers objets. Les fichiers objet ne sont typiquement pas générés, mais lorsque la demande en est faite, ils ont souvent l'extension `.o`.
4. L'édition des liens, qui est faite par l'éditeur de liens, prend les fichiers objet de l'étape précédente et d'autres fichiers, comme des librairies, et génère un fichier exécutable. Les exécutables sous Unix n'ont pas nécessairement l'extension `.exe` comme sous Windows.

5.2 Exemple de compilation

La compilation en langage C++ sous Unix se fait généralement avec la commande `g++`. D'autres compilateurs peuvent être utilisés, s'ils sont disponibles. Une commande typique de compilation avec `g++` est :

```
g++ triBulle.cpp
```

Cette commande fait toute la séquence pour générer un exécutable, compilation et édition des liens, comme montrés à la figure 1. Cependant, le seul fichier en sortie est l'exécutable, car les autres fichiers intermédiaires n'ont pas été exigés. Par défaut, le fichier exécutable est nommé `a.out`. On peut spécifier le nom du fichier exécutable généré avec l'option `-o` de la commande `g++`. Par exemple, si l'on désire que le fichier exécutable se nomme `triBulle` alors la commande est :

```
g++ -o triBulle triBulle.cpp
```

Si un projet est réalisé dans plusieurs fichiers, la compilation pour générer un exécutable peut aussi être effectuée avec la commande `g++`. Une commande typique est alors :

```
g++ -o testProg f1.cpp f2.cpp f3.cpp
```

5.3 Compilation séparée et make

Lorsqu'un projet est réalisé dans plusieurs fichiers, il est avantageux de faire de la compilation séparée afin de compiler ou de recompiler uniquement les parties du projet qui en ont besoin et ainsi sauver du temps. Les étapes de la compilation séparée sont montrées à la figure 2.

Toutes les étapes nécessaires à la compilation séparée sont réalisées, incluant l'édition des liens, afin de générer le fichier exécutable. Pour gérer des projets comportant plusieurs fichiers, l'environnement Unix offre au programmeur plusieurs outils permettant d'automatiser et faciliter la tâche du programmeur. Un des plus utilisés est l'utilitaire make. Nous exposons dans cette section les notions de base de make, vous trouverez la documentation complète de make avec la commande `man`.

L'utilisation de make pour la gestion de petits projets permet d'automatiser la compilation et libère le programmeur de la tâche répétitive de taper des commandes lors du développement d'une application. L'utilitaire make démontre vraiment toute sa puissance et son utilité avec de gros projets lorsque les dépendances entre les fichiers sources et les fichiers d'entêtes sont multiples et lorsque l'on prend avantage des règles de compilation prédéfinies.

L'utilitaire make lit le contenu d'un fichier contenant les dépendances entre les fichiers de l'application à créer et les commandes nécessaires pour la créer. Typiquement, pour un projet en C++, cela veut dire de compiler des fichiers ou de recompiler ceux qui en ont besoin, et de faire l'édition des liens. Lorsque la commande make est lancée, elle cherche dans le répertoire courant un fichier qui se nomme makefile ou, si ce fichier n'existe pas, elle tente de trouver un fichier se nommant Makefile (Unix est sensible aux majuscules). Lorsqu'un makefile est trouvé, make exécute alors les commandes dans ce fichier en accord avec les dépendances identifiées.

Avant de voir un exemple de makefile, les définitions des éléments principaux en faisant partie sont données.

Cible : une cible est quelque chose qui doit être construit par make. Une cible peut être un fichier exécutable ou encore un fichier intermédiaire, comme un fichier

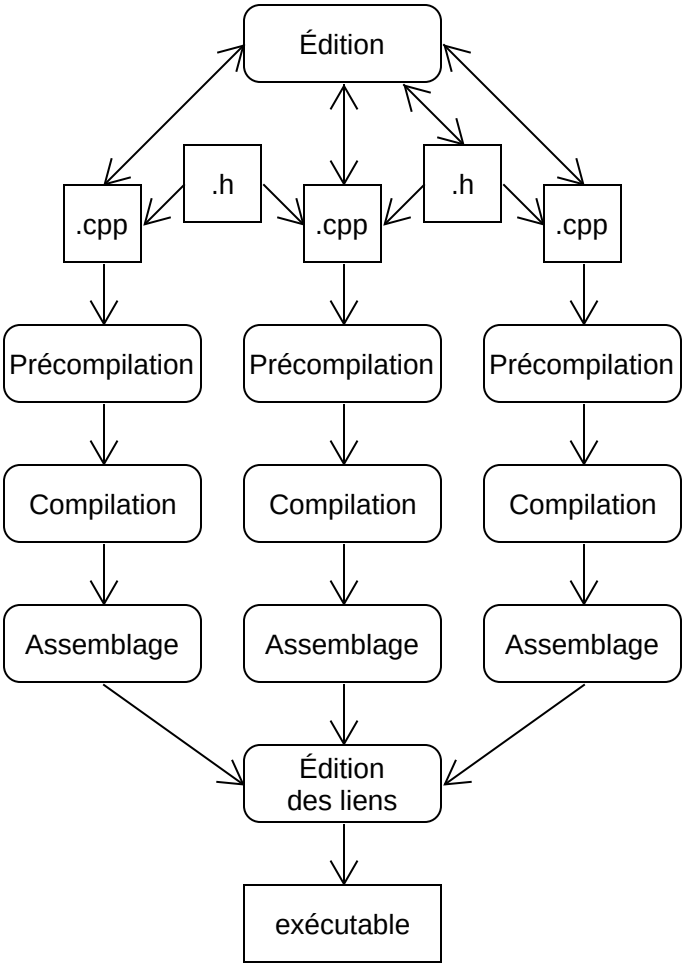

Figure 2: Compilation séparée en C++ sous Unix

objet. Certaines cibles sont seulement une série de commandes exécutées et ne construisent rien. Par exemple, la cible `clean` permet en général d'effacer du répertoire les fichiers intermédiaires résultant de la compilation. C'est une cible utilitaire qui aide à la gestion du projet.

Dépendance : elle établit la relation entre une cible et ses constituants.

Règle : une règle décrit comment bâtir une cible à partir d'une liste de dépendances. Typiquement, une règle invoque un appel à un compilateur ou un éditeur de liens. Il existe un ensemble de règles prédéfinies ou implicites (*implicit rules*). Un exemple de règle implicite est la relation existant entre un fichier de code source C++ (`.cpp`) et un fichier objet (`.o`), ils sont reliés par une compilation en faisant un appel au compilateur.

Macro : une macro est une variable. Elle peut être utilisée dans les dépendances et dans les règles. Un makefile utilisant des macros est généralement plus flexible et plus facilement modifié. Un exemple de macro est :

```
OBJECTS = main.o fichier1.o fichier2.o
```

Le nom de la macro est `OBJECTS` et sa valeur est `main.o fichier1.o fichier2.o`. À chaque fois que make rencontre la chaîne `$(OBJECTS)`, la macro est alors remplacée par sa valeur. La syntaxe `$(...)` indique une substitution à faire. Une série de macros prédéfinies existent dans make dont plusieurs qui sont très utiles pour la compilation en langage C++. Il faut se référer à la documentation de make.

Voici un exemple de makefile et il est suivi d'explications.

```
1  # makefile pour le programme de pile
2  CC = g++
3
4  testpile: testpile.o pile.o
5    $(CC) testpile.o pile.o -o testpile
6
7  testpile.o: pile.h testpile.cpp
8    $(CC) -c testpile.cpp
9
10 pile.o: pile.h pile.cpp
11   $(CC) -c pile.cpp
```

Le caractère # introduit un commentaire ; la ligne 1 est donc un commentaire. À la ligne 2, la macro prédéfinie `CC` est redéfinie pour indiquer que le compilateur à utiliser est le compilateur C++ de GNU, le compilateur `g++`.

Les lignes 4 et 5 introduisent la première règle. Dans cette règle, la ligne 4 introduit la cible `testpile`. Le nom de la cible est suivi d'un deux-points et de ses dépendances, les fichiers `testpile.o` et `pile.o`. Suivent ensuite les commandes (ou la commande) permettant de bâtir la cible. Si une ou plusieurs des dépendances de `testpile` sont reconstruites alors `testpile` doit être reconstruit en exécutant les commandes spécifiées (ligne 5). **Il est à noter que les commandes sont obligatoirement indentées par un caractère de tabulation (TAB) et non pas des espaces.** Cette règle peut-être lue de la façon suivante : si un ou plusieurs des fichiers objets (les dépendances) ont été modifiés, il faut refaire l'édition des liens (la commande) pour rebâtir le fichier exécutable `testpile` (la cible).

La règle suivante (lignes 7 et 8) peut-être lue comme suit : si un ou plusieurs fichiers parmi `pile.h` ou `testpile.cpp` ont changé (les dépendances), il faut rebâtir le fichier objet `testpile.o` (la cible) avec la commande de la ligne 8. La règle aux lignes 10 et 11 peut être interprétée de manière similaire.

Il est important de réaliser que :

- Il est préférable de ne pas mettre d'espace dans le nom des cibles.
- make se base sur la date et l'heure des fichiers pour rebâtir une cible.
- make ne connait pas le C++.
- make ne remonte donc pas la chaine des `#include` dans des `#include`.

Ce makefile permet d'automatiser la compilation de l'application `testpile`. Si des changements ont été faits sur certains des fichiers constituant l'application seulement ceux-ci seront compilés et l'application est reconstruite. Si aucun changement n'a été fait dans les fichiers, la commande make ne fait rien.

6 Contrôle de l'exécution

Le *shell* reconnaît quelques séquences de clés permettant de contrôler l'exécution d'un programme ou de l'affichage. Le tableau 2 présente les plus importantes. Toutes ces séquences sont modifiables et peuvent ainsi changer d'un système à l'autre. La commande `stty -a` permet d'afficher la configuration courante pour votre terminal.

Tableau 2: Séquences de contrôle de l'exécution

Nom	Séquence	Description
intr	`<Ctrl-c>`	Stopper un programme en exécution
erase	`<Ctrl-h>`	Effacer le dernier caractère tapé
werase	`<Ctrl-w>`	Effacer le dernier caractère tapé
kill	`<Ctrl-u>`	Effacer la ligne au complet
quit	`<Ctrl-\>`	Stopper un programme, plus fort que `intr`
stop	`<Ctrl-s>`	Pause le défilement à l'écran
start	`<Ctrl-q>`	Poursuit le défilement à l'écran
eof	`<Ctrl-d>`	Indique la fin des données

Historique des versions

— Version originale (1994) : C.-A. Brunet, D. Dalle et J.-M. Dirand
— Révision (2001) : C.-A. Brunet et D. Dalle
— Révision (2005) : C.-A. Brunet et D. Dalle
— Révision (2012) : C.-A. Brunet et D. Dalle
— Révision (2019) : C.-A. Brunet

Liste des références

[1] P. Abrahams et B. Larson, *UNIX for the impatient*. Addison-Wesley, 1992.

[2] H. Hahn, *A student's guide to UNIX*. Mc Graw-Hill, 1993.