Cinquième partie

Outils pour la pratique

CHAPITRE

1

Les bases des séances de TP

Avant de lire cettre section, il est suggéré de relire vos notes de cours se rapportant au système d'exploitation Unix.

1.1 Comment ouvrir une session ?

Pour ouvrir une session, il vous faut votre login (souvent votre nom de famille) et votre mot de passe (passwd). Ces deux informations vous ont été données en cours si vous avez remis votre fiche informatique.

Attention, il vous est fortement conseillé de **NE PAS** changer votre mot de passe par rapport à celui qui vous a été fourni, pour des questions de sécurité, ce mot de passe est connu de votre moniteur qui pourra ainsi vous dépanner si vous oubliez comment vous connecter.

Toutefois, si vous changez votre mot de passe, choisissez en un d'au minimum huit caractères dont au moins un **n'est pas** une lettre – et surtout NE L'OUBLIEZ PAS ! Nous ne pouvons pas retrouver votre mot de passe si vous ne vous en souvenez plus ! Le temps perdu ne pourra pas être récupéré.

FIG. 1.1 – Exemple de fenêtre de connexion (login). Le mot de passe n'apparait pas lorsque vous le tapez, ou encore il est recouvert de traits noirs. Il faut donc s'assurer de taper les bonnes touches : est-ce que le clavier est AZERTY ou QWERTY .. ?

1.2 Premiers pas

Une fois la session ouverte, vous devez ouvrir une fenêtre X, qui vous donnera accès au shell par défaut. C'est dans cette fenêtre que vous taperez les commandes Unix. Par défaut vous vous trouvez dans votre répertoire courant.

Il est important pour vous d'apprendre à utiliser cet environnement. Même si il existe un utilitaire graphique vous permettant de visualiser vos fichiers et répertoires en cliquant avec la souris comme sous Windows, nous vous déconseillons de l'utiliser. Les commandes shell sont des outils beaucoup plus puissants et si vous ne vous habituez pas à les utiliser pour des choses simples dès le début (renommer, copier ou effacer un fichier, changer de répertoire, etc ...), vous n'apprendrez pas à vous en servir et serez pénalisés par la suite.

Une bonne chose à faire à chaque début de TP est de créer un répertoire spécifique dans lequel on mettra tous les fichiers correspondants au TP en cours. Par exemple pour le premier TP, on va créer un répertoire `TP1` en tapant la commande suivante :

```
mkdir TP1
```

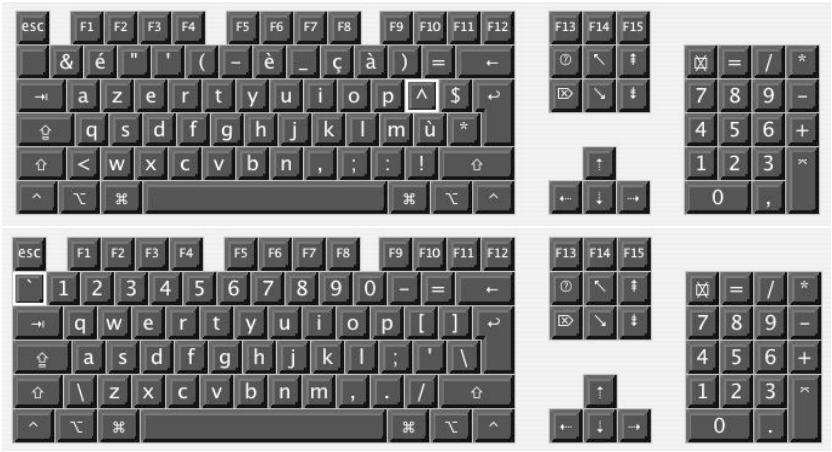

FIG. 1.2 – Exemple de claviers Qwerty/Azerty : ces lettres sont celles de la 3ième rangée à partir du haut, partie de gauche.

On souhaite ensuite aller dans ce répertoire :

```
cd TP1
```

Et on peut enfin commencer le TP ! La création d'un répertoire par TP permet

- d'y voir plus clair dans l'organisation des fichiers
- de ne pas avoir trop de fichiers dans le répertoire courant
- de ne pas effacer un fichier du TP précédent qui aurait le même nom.

1.3 Comment ouvrir un éditeur de texte ?

Pour ouvrir l'éditeur de texte `nedit`, il vous suffit de taper la commande :

```
nedit &
```

puis de taper votre texte dans la fenêtre qui vient de s'ouvrir.

Remarque sur la commande : le symbole & à la fin de la commande permet de détacher le programme (ici nedit) de la fenêtre X. Si vous ne tapez pas le & à la fin, la fenêtre reste bloquée en attendant la fin du programme (nedit) et les commandes tapées dans la fenêtre inactive n'ont pas d'effets.

Pour ouvrir un fichier déjà existant (par exemple toto.c), il suffit de taper :

```
nedit toto.c &
```

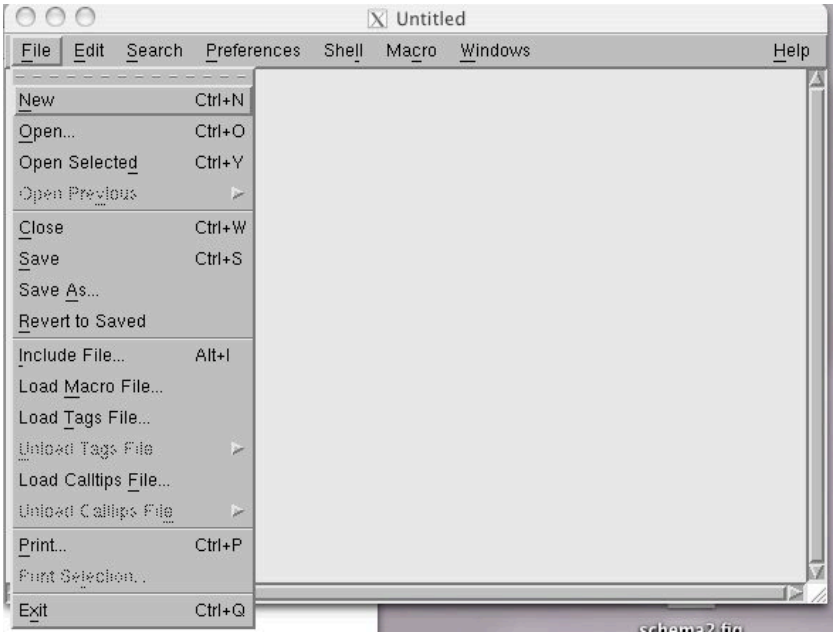

FIG. 1.3 – Exemple de session avec l'éditeur de texte NEDIT. L'ouverture de fichiers se fait à partir de l'onglet *file* ou *fichier*.

1.3.1 Comment sauvegarder un fichier ?

Une fois le texte tapé dans l'éditeur, vous pouvez vouloir sauvegarder ce que vous avez écrit dans un fichier.

Si le fichier n'a pas encore de nom ou que vous voulez en changer, cliquez sur le menu "File" et choisissez "Save As" ou "Sauvegarder sous". Vous pouvez alors choisir le répertoire dans lequel sera le fichier et taper le nom que vous souhaitez donner dans la case "New File Name", puis apputer sur OK.

Si le fichier a déjà un nom, choisissez "Save" dans le menu "File".

Attention : si vous sauvegardez un programme sous deux répertoires différents, deux fichiers différents sont crées, même si ils ont le même nom ! Vérifiez ce que vous compilez, et surtout dans quel répertoire vous travaillez !

1.4 Comment ouvrir un navigateur web ?

Il vous suffit de taper la commande :

```
netscape &
```

Le navigateur netscape va s'ouvrir dans une fenêtre indépendante. Vous pouvez ainsi aller sur le site web du cours : `http://astro.u-strasbg.Fr/`

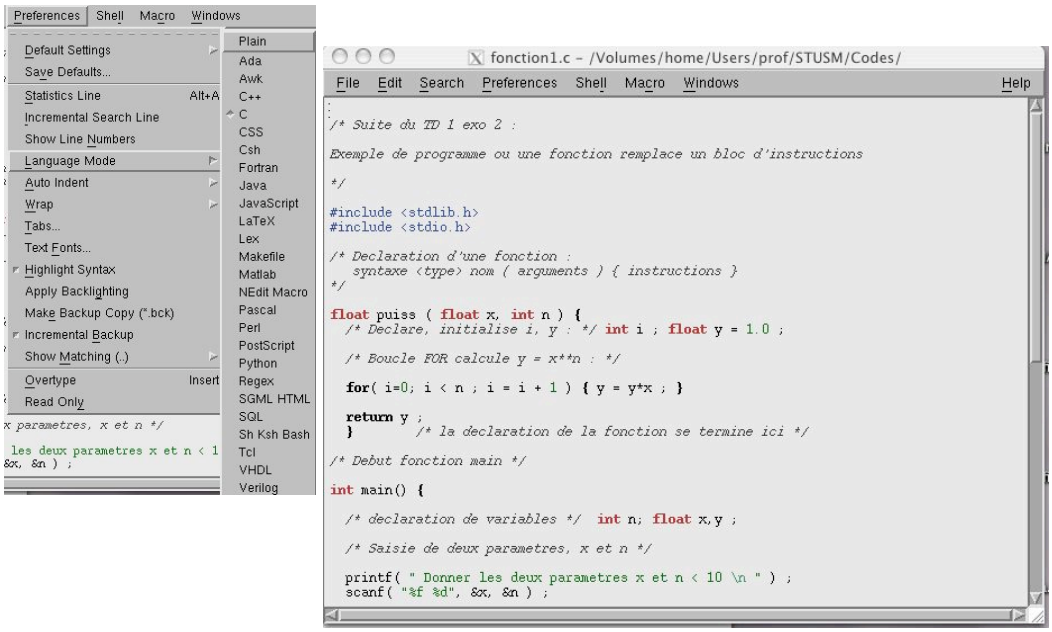

FIG. 1.4 – L'éditeur de texte NEDIT peut être configuré pour l'édition de programme C. Voir sous l'onglet 'préférences', choisr 'langage', puis cliquer sur 'C'.

`scyon/STUSM/`

Il peut être interessant de garder cette page en mémoire pour éviter de retaper l'adresse à chaque fois. Pour cela, cliquez sur "Bookmarks" puis "Bookmark this page". L'adresse de la page web s'est ajoutée à la liste disponible sous l'onglet "Bookmarks", et vous pouvez y retourner facilement quand vous voulez...

Aller sur la page web sert en particulier à récuperer des exemples de codes. Vous les trouverez sous le répertoire `Code` du site. Pour sauvegarder un fichier (par exemple Entree_sortie.c) cliquez avec le bouton droit de la souris sur le nom de fichier et choisissez "Save link target as" ou "Sauvegardez le lien sous", puis choisissez le répertoire sous lequel vous voulez avoir le fichier en question.

1.5 Comment compiler un programme ?

Une fois que vous avez sauvegardé votre programme C ou que vous en avez téléchargé un, il faut le compiler, c'est à dire le rendre compréhensible par l'ordinateur. Mettons que le programme s'appelle `titi.c`. On utilise le compilateur gcc avec la commande :

```
gcc titi.c
```

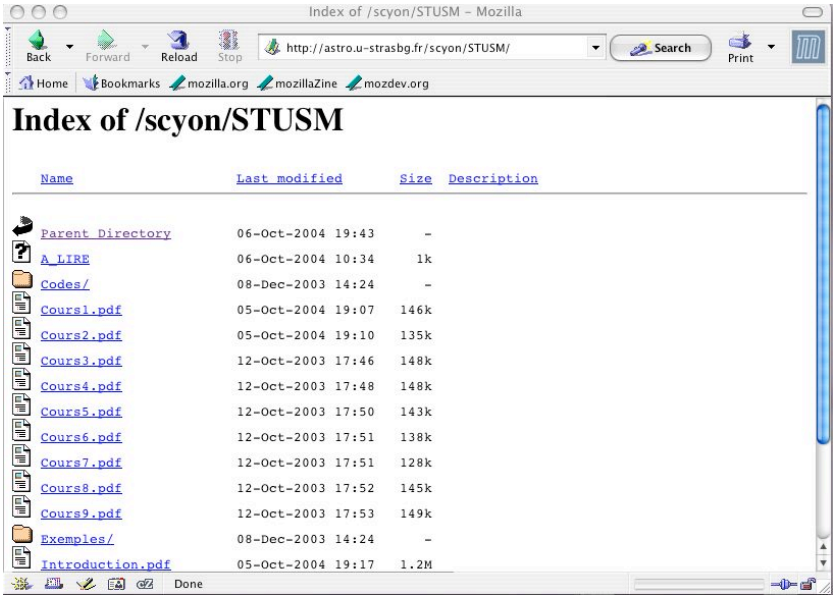

FIG. 1.5 – Le lancement d'un navigateur (netscape ou mozilla) se fait de la même manière, en tapant leur nom sur la ligne de commande. Leur apparence est très similaire.

Dans ce cas, un fichier a.out est crée. C'est l'éxécutable. Pour lancer votre programme, il vous suffit de taper `./a.out`.

Si vous souhaitez que l'éxécutable ait un nom différent (par exemple titi.out ou simplement titi) il faut la commande suivante :

```
gcc -o titi titi.c
```

Pour lancer le programme il suffit alors de taper `./titi` sur la ligne de commande.

Cette méthode est préférable puisque vous avez alors un nom d'éxécutable par programme et vous n'effacez pas systématiquement l'ancien fichier a.out.

CHAPITRE

2

Aides techniques

2.1 Rappel des commandes Unix les plus courantes

IMPORTANT : Pour savoir ce que fait une commande, toujours penser à utiliser la commande “man”, abréviation de manuel.

2.1.1 commandes utilitaires

passwd : permet de changer de mot de passe. Utilisation : passwd < votre_nom_de_login>
date : affiche la date.
man <**nom_commande**> : aide en ligne

2.1.2 commandes sur les fichiers

pwd : affiche le nom du répertoire courant.
ls : liste le nom des fichiers.
ls -l : édition du catalogue des fichiers du répertoire courant
ls -la : édition de tous les fichiers du répertoire courant (y compris les fichiers ”cachés” commenant par un .)
cd <**répertoire**> : permet de se placer dans un répertoire donné.
cd retour au répertoire d’accueil

cd retour au répertoire d'accueil

cd .. retour au répertoire père

mkdir <**répertoire**> crée un répertoire (Make directory).

rmdir <**répertoire**> supprime un répertoire (vide) (Remove directory).

cat [-n] [fich1 ... fichn] affiche le contenu des fichiers sur la sortie standard. -n pour afficher le numéro des lignes

more <**fichier**> édition du fichier au terminal en mode page

less <**fichier**> édition du fichier au terminal en mode page (équivalent de more).

head [-n] <**fichier**> affiche les n 1ères lignes du fichier (par défaut n=10)

tail [-n] <**fichier**> affiche les n dernières lignes du fichier (par défaut n=10). L'option $+n$ affiche le fichier privé des n premières lignes.

wc -l -w -c <**fichier**> retourne le nombre de lignes (l), de mots(w), de caractères(c) dans le fichier indiqué.

cp <**fichier1**> <**fichier2**> copie le contenu de <fichier1> dans <fichier2>, en détruisant <fichier2> s'il existait.

On distingue 2 usages :

- **cp [option] source destination** copie d'un seul fichier, en précisant le chemin et le nom du fichier destination
- **cp [option] ens-fichiers-source répertoire** copie l'ensemble des fichiers dans le rép. spécifié, en gardant les noms

mv <**fichier1**> <**fichier2**> renomme ou déplace <fichier1> en <fichier2>

mv <**fichier1**> <**répertoire**> Déplace (et éventuellement renomme) <fichier1> dans <répertoire>.

rm <**fichier**> supprime le fichier (Remove).

-i demande de confirmation

-r suppression récursive (le fichier est alors un répertoire)

2.1.3 Filtres évolués

sort [options] [critère] [fichier] tri un fichier.

grep [options] [modèle] [fichier] recherche une chaîne de caractères dans un fichier donné et édite les lignes la contenant. La chaîne trouvée correspond au modèle donnée par une expression régulière.

find <**chemin**> **-name** <**fichier**> **-print** chercher <fichier> dans une liste de répertoire définis par <chemin> (expression régulière)

2.1.4 Droits d'accès aux fichiers

chmod <**droits**> <**fichier**> changement des droits d'accès d'un fichier.

droits : r (read), w (write), x (execute)

2.1.5 Processus

ps affiche les informations sur les processus en cours

<**ctrl-c**> termine le processus premier-plan courant

<**ctrl-z**> stoppe l'exécution du processus premier-plan courant (qui pourra tre relancé par bg).

<**commande**> **&** lance un programme en arrière plan.

xterm & lance une nouvelle fentre xterm

jobs affiche les jobs (et leur numéro) stoppés et/ou passés en arrière-plan.

2.1.6 Compression, encodage, archivage

gzip <**fichier**> compression. Génère un fichier du même nom mais avec l'extension .gz .

gunzip <**fichier.gz**> décompression d'un fichier nommé fichier.gz, redonnera le fichier initial moins l'extension .gz .

tar cvf <**tarfichier**> <**répertoire_à_archiver**> archive un ensemble de fichiers ou les fichiers d'un répertoire en un fichier.tar unique (extension .tar).

tar xvf <**tarfichier**> dé-archivage d'un ensemble de fichiers du fichier.tar.

CHAPITRE

3

‘‘Et quand ça ne marche pas ?”

Il existe nombre d’erreurs courantes de programmation. Certaines sont détectées par le compilateur. Il faut ensuite déchiffrer le message d’erreur pour corriger le programme. D’autres erreurs ne sont pas détectées à la compilation mais donnent un résultat faux ou un plantage du programme. Celles-là sont habituellement des erreurs algorithmiques ou de la transcription de l’algorithme en C ; elles sont plus difficiles à identifier, aussi faut-il procéder méthodiquement ..

3.1 L’erreur la plus fréquente : parse error

Le compilateur renvoie le message *parse error* : vous avez oublie un point-virgule (;) ou une parenthèse ou un crochet à la ligne indiquée.

3.2 Fautes de frappes

3.2.1 Mélange entre = et ==

Il est parfaitement accepté par le compilateur d’avoir la syntaxe suivante :

```
if (x=0){
```

```
[traitement]
}
```

mais cela ne signifie pas que si la variable x vaut 0 alors le traitement sera fait ! L'ordinateur affecte la valeur 0 à x, et donc la parenthèse *if(x=0)* est équivalente à *if(0)* qui donne toujours le résultat *faux*, puisque 0 est la transcription en C de faux.

La plupart du temps, ce que l'on veut écrire c'est (doublant le '=' pour en faire un opérateur relationnel d'égalité) :

```
if (x==0){
[traitement]
}
```

3.2.2 Priorité des opérateurs

Lorsque vous écrivez une opération complexe, mettez des parenthèses ! Beaucoup de résultats faux sont trouvés simplement parce que la priorité des opérateurs n'a pas été prise en compte.

3.3 Oubli de la condition de sortie

Dans une boucle, il ne faut pas oublier de mettre “une condition de sortie”, c'est-à-dire une manière pour l'ordinateur de sortir de la boucle sous certaines conditions. Sinon le programme tourne sans fin ! C'est fréquemment le cas lorsque :

- on oublie le `break` dans les `switch`
- la condition est toujours vraie dans les boucles `for (i=0; <condition>;i++){}` ou `do{...}while(<condition>)`

Une seule solution : s'assurer que les blocs d'instructions (ou traitement) modifient les paramètres de la condition pour rencontrer (éventuellement) un cas d'exception.

3.4 Variables locales et globales

Les variables déclarées dans une fonction ont une portée locale uniquement. Pour éviter les confusions (et les heures à debugger) essayez d'utiliser des noms de variables différents dans le main et dans les fonctions.

3.5 Allocation de la mémoire : segmentation fault

Le message d'erreur *segmentation fault* du compilateur signifie que vous avez fait référence dans votre programme à une zone de mémoire non allouée. Cela peut être parce que vous sortez de la taille d'un tableau (dans une boucle par exemple).

3.6 Entrées/sorties standard

3.6.1 scanf

- les paramètres de scanf() sont les adresses des variables à lire, pas les variables elles-mêmes. Il faut écrire `scanf(``%d'',&n);` et non `scanf(``%d'',n);`
- attention à lire une variable avec le bon format. Si on veut lire un entier, il faut utiliser `%i`, pour un réel `%f`, etc ...

3.6.2 printf

L'erreur la plus courante est de vouloir afficher une chaine de caractères avec un type entier ou réel.

3.7 Debuggons ensemble

Voici un exemple pas à pas de compréhension des messages d'erreurs du compilateur et de la correction de ces erreurs dans le programme.

Prenons le programme toto.c (plein d'erreurs) suivant

```
#include <stdo.h>
#include <stdlib.h>

int main(void)
 {
  printf("Salut tout le monde!\n")
  return 1;
 }
```

À la compilation, le message suivant est affich :

```
toto.c :1 : stdo.h : No such file or directory
```

Décortiquons ce message :

- l'erreur se situe dans le fichier toto.c

– l’erreur se situe à la ligne 1 (`:1`)
  Remarque : Dans les éditeurs de texte courants (nedit, emacs, . . .) il existe la possibilité d’afficher les numéros des lignes et/ou de se déplacer directement à une ligne connaissant son numéro.
– le programme ne trouve pas le fichier stdo.h

La première erreur à corriger est une faute de frappe : il fallait écrire stdio.h au lieu de stdo.h.

Une fois l’erreur corrigée, voici le programme toto.c :

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  printf("Salut tout le monde!\n")
  return 1;
}
```

En recompilant, un nouveau message d’erreur apparait :

```
toto.c : In function ‘main’ :
toto.c :7 : parse error before ‘return’
```

Le premier message nous indique simplement dans quelle fonction se trouve l’erreur. Le second dit qu’ la ligne 7 le compilateur n’arrive pas analyser syntaxiquement notre programme avant le mot `return`, en d’autres termes qu’il y a une erreur de grammaire. En effet il manque le ; à la ligne 6 qui doit terminer une instruction en C.

Corrigeons à nouveau le programme :

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  printf("Salut tout le monde!\n");
  return 1;
}
```

Cette fois-ci la compilation va jusqu’au bout et produit le fichier exécutable.

3.8 Avertissement sur les messages d'erreur

Il arrive qu'à la compilation une grande quantité de messages d'erreurs s'affiche. Il est important de corriger ces erreurs **une à une**, et **en commencant par la première affichée**. En effet, les autres messages d'erreurs ne peuvent être que les conséquences d'une mauvaise interprétation du programme, due à la première erreur. Cet effet de dominos se rencontre très souvent.

3.9 Et si ça ne marche toujours pas ?

Un programme qui ne marche pas est aussi parfois un programme qui est mal construit. Prenez une feuille de papier et ecrivez ce que fait le programme pas à pas et les résultats que vous attendez à chaque étape ... il y a des chances que l'erreur vous saute aux yeux !

3.10 Et si ca marche vraiment toujours pas et que j'ai tout essayé ?

Appelez nous ! On est là pour ça !

CHAPITRE

4

Petite bibliographie

La liste ci-dessous n'a pour but que de vous donner des lieu où chercher si vous voulez en savoir plus et aller plus loin, ou que vous ne comprenez pas un point précis. Elle est très courte mais les ouvrages ou liens mentionnés ont un rapport direct avec le cours et sont faciles d'accès (livres disponibles à la bibliothèque des Sciences de l'ULP ou sites internets).

- Les livres
  - Le livre du premier langage C pour les vrais dénutants en programmation, de Claude Delannoy, éditions Eyrolles
  - Langage C, de Claude Delannoy, éditions Eyrolles
  - Unix Shell avec 160 exercices corrigés, éditions Eyrolles
- Sur le web
  - Un cours de langage C de l'IPST de Strasbourg : `http://www-ipst.u-strasbg.fr/pat/program/tpc.htm`
  - Le guide superflu de la programmation en C `http://www.laas.fr/\%7Ematthieu/cours/c-superflu/index`