

# Introduction à LINUX

Licence de Physique/M2 ITDD

[Home|Linux|C++|Graphisme|Projets|Web|FAQ]

## Table des matières

<b>1 Linux c'est quoi ?</b>	<b>1</b>
1.1 Comment dialoguer avec l'ordinateur? . . . . .	1
1.2 Les utilisateurs . . . . .	2
1.3 Organisation des fichiers . . . . .	2
<b>2 Principales commandes</b>	<b>2</b>
2.1 Les répertoires et Fichiers . . . . .	2
2.2 Éditeurs de textes . . . . .	4
2.3 Divers commandes utiles . . . . .	4
2.4 Enchaînement de commande . . . . .	5
<b>3 Impression</b>	<b>5</b>
<b>4 Interruption, passage en background d'un exécutable</b>	<b>5</b>
<b>5 Les variables dans un shell</b>	<b>6</b>
5.1 Complément : Les variables <i>standards</i> . . . . .	6
5.2 Les Variables <i>d'environnement</i> . . . . .	6
5.3 Le bash et le .bashrc . . . . .	7
5.4 Le csh et le .cshrc . . . . .	7
<b>6 Programmation : le début</b>	<b>8</b>
6.1 Écriture . . . . .	8
6.2 Compilation . . . . .	8
6.3 Exécution . . . . .	8

## 1 Linux c'est quoi ?

*LINUX* est un **système d'exploitation** (au même titre que *Windows XP*, *7*, ...) qui permet de "dialoguer" avec l'ordinateur. Ce système est très puissant et très stable (contrairement à *Windows*). Bien qu'au premier abord il puisse paraître un peu austère, on s'y habitue très vite.

### 1.1 Comment dialoguer avec l'ordinateur ?

Ceci se fait dans une fenêtre dite **terminal**<sup>1</sup> *via un langage le shell*. Il existe, sous LINUX, 3 principaux shell : le *Bourne shell* (*sh*) qui est le plus ancien et sans doute le plus utilisé, le *C shell* (*csh*) très utilisé et le *Korn shell* (*ksh*) qui est un peu un mélange des deux autres. Hormis ces shells, il existe des versions un peu plus conviviales : ce sont par exemple le *bash* (sh amélioré) et le *tcsh* (csh amélioré).

---

1. Il existe divers types de *fenêtre terminal* selon les machines et les environnements de travail (gnome-terminal, kterm, xterm, ...). Ces fenêtres ont plus ou moins les mêmes fonctionnalités. Il existe un type de fenêtre qui est cependant standard sur toutes les machines, c'est les *aterm*.

## 1.2 Les utilisateurs

Pour vous connecter sur une machine *LINUX* vous avez besoin d'un *login*, c'est-à-dire un nom d'utilisateur et d'un *mot de passe*. Chaque utilisateur appartient à un *groupe*, ce qui permet de donner certains *privileges* (possibilité d'écrire, lire ou exécuter un programme dans une certaine zone) communs à tous les utilisateurs de ce groupe. Il existe cependant un utilisateur particulier, nommé *root* qui a tous les privilèges.

## 1.3 Organisation des fichiers

Il est important de savoir que les **fichiers** sont classés dans l'ordinateur (sur le disque magnétique ou dans une mémoire flash) selon une **structure arborescente**. Chaque nœud de l'arborescence s'appelle un répertoire ou **directory** (en anglais). Dans ces répertoires se trouvent des fichiers (ou *file* en anglais).

Par exemple `/home/phys/aaa/zorro.c` signifie que le fichier `zorro.c` se trouve dans le répertoire `aaa` qui se trouve lui-même dans le répertoire `phys`, qui se trouve lui-même dans `home` qui est dans `/`. Le **répertoire racine est /**.

## 2 Principales commandes

Toutes les commandes "*Linux*" que nous utiliserons sont des commandes à taper dans une **fenêtre terminal** (*gnome-terminal*, *xterm*, ...). Avant de regarder les commandes plus en détail, il y a deux commandes dont vous aurez besoin rapidement : *man* et *passwd*. La première vous sera très souvent utile car elle permet d'avoir un aide en ligne sur les autres commandes et même sur certaines fonctions C/C++. On tape

*man* commande

La seconde permet de changer votre mot de passe ; on tape

*passwd*

et on suit les instructions... Un mot de passe doit avoir 8 caractères (lettres, chiffres, ...).

### Remarques :

- **LINUX (comme le C/C++) fait la différence entre les MAJUSCULES et les minuscules.**
- pour copier/coller un exemple donné sur la page web utilisez la souris : bouton gauche, et bouton milieu :-).

### 2.1 Les répertoires et Fichiers

- Pour connaître le répertoire où vous êtes (le répertoire courant) : *pwd*.
- Pour avoir la liste des fichiers et des sous répertoires du répertoire courant :
  - *ls*
  - *ls -als* pour avoir plus de détails<sup>2</sup>

---

2. par exemple, *ls -als* donne dans un certain répertoire :

```
1 drwxrwxr-x 6 zorro gentil 512 May 3 17 :35 .
1 drwxr-xr-x 6 zorro gentil 512 Mar 25 11 :08 ..
1 drwxrwxrwx 2 zorro gentil 512 Apr 21 16 :16 Affaire
1 drwxr-x— 2 zorro gentil 512 Apr 14 14 :58 Exemple
9 -rw-rw-r— 1 zorro gentil 8334 Apr 13 11 :06 readme
16 -rwxr-x-x 1 zorro gentil 833415387 Apr 01 10 :00 lance
11 -rw-r-r— 1 zorro gentil 10999 Apr 24 11 :55 prg.C
```

Considérons la 1<sup>ère</sup> ligne :

le 1 désigne la taille occupé en blocs de 512 octets par le directory "." ; le d signifie qu'il s'agit d'un directory ; il y a ensuite 3 groupes de 3 caractères donnant les priorités de lecture (r), d'écriture (w) et d'exécution (x) pour l'utilisateur (le 1<sup>er</sup> groupe) (ici c'est *zorro*), le groupe de l'utilisateur (ici *gentil*) et le reste (*other*). Pour ce directory, *zorro* et les membre du groupe *gentil* ont les mêmes privilèges, par contre les autres ne peuvent que lire et exécuter. Ensuite, figure la taille en octet (ici 512) puis la date de dernière modification, puis le nom du répertoire (ici ".")

De même pour le directory parent "."

Le directory *Affaire* a les mêmes privilèges pour tout le monde.

Le directory *Exemple* ne peut être lu,écrit,exécuté par des utilisateurs extérieurs au groupe *gentil*

- **ls -F** pour voir rapidement les fichiers exécutables (ils sont suivis d'une étoile), les répertoires (ils sont suivis d'un /).
  - **ls [-alsF]<sup>3</sup> filename** pour ne voir que le fichier *filename*<sup>4</sup>
  - Pour changer de répertoire :
    - **cd rep** : pour aller dans le sous répertoire appelé *rep*.
    - **cd ..** : pour aller dans le répertoire parent. (“..” est le chemin relatif qui représente le parent ; “.” est le chemin relatif représentant le répertoire courant).
    - **cd** : pour revenir à votre répertoire principal (ce répertoire s'appelle *HOME* ou *home directory*).
  - Pour créer un sous répertoire appelé *rep* : **mkdir rep**
  - Pour détruire un fichier :
    - **rm filename** : détruit le fichier *filename* (sans demander de confirmation)
    - **rm \*** : détruit tous les fichiers (sans demander de confirmation !)
    - **rm -i fi\*** : détruit tous les fichiers commençant par *fi* en demandant une confirmation
  - Pour détruire le répertoire *rep* ainsi que les fichiers qu'il contient : **rm -r rep**
  - Pour copier un fichier : **cp chemin1/fic1 chemin2/fic2** où *chemin 1/2* peut être un *chemin absolu* (/home/users/zorro/Affaire) ou un *chemin relatif* (../zorro/Affaire ou bien Exemple/essai) et *fic1/2* sont les noms des fichiers.
  - Pour copier un répertoire : **cp -r chemin1/rep1 chemin2/rep2**
  - Pour déplacer un fichier : **mv chemin1/fic1 chemin2/fic2** (en particulier, **mv chemin1/fic1 chemin1/fic2** renomme *fic1* en *fic2*)
  - Pour déplacer un répertoire : **mv chemin1/rep1 chemin2/rep2** (ceci peut aussi servir à renommer un répertoire)
- ATTENTION**, comme pour la commande **rm**, les commandes **cp** et **mv** ne demandent pas de confirmation si un fichier *fic2* existait déjà avant le **cp** ou le **mv**. Celui-ci sera perdu!
- Pour créer un fichier (voir Comment choisir un nom de fichier)
    - **ls > liste** : redirige le résultat de la commande **ls** dans le fichier *liste* (en le créant)
    - **ls >> liste** : redirige le résultat de la commande **ls** dans le fichier *liste* (en l'ajoutant à la fin)
    - **more fic** : voir le contenu d'un fichier
    - **tail [-n] fic** : voir les **n** dernière lignes d'un fichier [par défaut *n=10*]
- Remarquez qu'**en pratique on utilise souvent un éditeur** (voir le paragraphe 2.2 pour info) pour créer/visualiser un fichier.

## Exercices

- A l'aide des commandes expliquées ci-dessus :
  1. Placez vous dans votre répertoire principal. Vérifiez que vous y êtes bien. Regardez la liste des fichiers par : **ls**, puis avec : **ls -als**
  2. Créer un répertoire *essai*. Vérifier son existence (avec **ls**).
  3. Aller dans ce répertoire. Vérifier que vous y êtes (avec **pwd**). Créer 3 fichiers vides *fic1*, *fic2* et *fic3* (on pourra utiliser la commande **touch fic1**)
  4. Revenir dans le répertoire principal par **cd ..**. Vérifiez que vous y êtes. Créer un répertoire *exemple* contenant 4 fichiers vides *ex1*, *ex2*, *fic1*, *fic2*
  5. Revenir dans votre *home directory*. Taper les commandes **ls**, puis **ls exemple/**, **ls exemple/fic\*** et enfin **ls \***.
  6. Taper ensuite **ls \* > liste1**. Vérifier avec la commande **more** le contenu de *liste1*.

Le fichier *readme* occupe 8334 octets (soit 9 blocs de 512) et peut être lu par tout le monde mais modifié que par les gens du groupe *gentil*.

...

3. Les crochets désigne une option non obligatoire

4. On peut utiliser des *Wildcards*, c'est-à-dire “\*”, “?” ou “~”. L'étoile remplace un ensemble de caractères, le “?” remplace un seul caractère et le “~” désigne le *home directory*. Ainsi **ls fic\*** donne la liste de tous les fichiers commençant par *fic*, **ls ~** donne la liste de votre répertoire principal.

7. Renommer le fichier *liste1* en fichier *liste2*. Vérifier avec la commande **ls**. Taper ensuite **mv exemple/ex1 essai/**. Observer le résultat en listant les contenus de *exemple* et *essai*. Se Placer dans *essai/* et taper **ls** puis **mv ex1 fic1** ; vérifier le résultat avec **ls** puis faire **mv -i fic1 fic2**.
8. Détruire les fichiers de *essai* par **rm -i \***. Se placer dans votre *home directory* et taper **rm -r essai** puis **rm -r exemple**.
9. Créer un répertoire *bin*, *lib* et *tmp*. Ces répertoires vous seront utiles pour la suite et vous permettront de mettre des exécutable Linux (*bin/*), vos librairies (*lib/*) ou des fichiers temporaires (*tmp/*).

## 2.2 Éditeurs de textes

Comme nous l'avons déjà souligné, pour créer ou visualiser un fichier, on utilise généralement un éditeur. Il existe un grand nombre d'éditeurs différents ; citons *Word*<sup>5</sup>, *vi*<sup>6</sup>, *emacs*<sup>7</sup>, *gedit* et enfin *nedit* qui permet de faire beaucoup de choses, qui est assez conviviale et instinctif. Nous utiliserons donc *nedit* pour tous nos exemples. On le lance en tapant **nedit** ou **nedit filename** pour éditer le fichier *filename*. Vous trouverez ici un petit aperçu de *nedit*

### Remarque

- il est souvent utile d'utiliser la commande **nedit &** ou **nedit filename &** cela permet "de garder la main" dans la fenêtre *terminal* (*gnome-terminal*, *xterm*, ...) (faire l'essai avec est sans le signe **&**). Le signe **&** signifie "lancement en *background*" (arrière plan) ; il est valable pour n'importe quelle application.

## 2.3 Divers commandes utiles

- **alias/unalias** : cette commande permet de définir un raccourci : par exemple, nous avons vu que la commande **rm fic** détruit le fichier *fic* sans demander confirmation ; pour plus de sûreté, on peut utiliser **rm -i fic** qui demande une confirmation. On peut redéfinir la commande grâce à un *alias* :
  - **alias rm "rm -i"** en *csh* ou
  - **alias rm="rm -i"** en *bash*
 ainsi quand vous taperez **rm** la commande exécutée sera en fait **rm -i**
- alias** sans argument donne la liste des *alias* déjà existant et **unalias mon\_alias** annule la commande **alias**. Cette commande très utile est à utiliser avec beaucoup de précaution (imaginer l'effet de commande comme : **alias ls "cd"**)
- **find** : comment rechercher un fichier dans une arborescence ?
  - **find chemin -name filename** : cherche tous les fichiers *filename* depuis le répertoire pointé par *chemin*.
  - **find . -name "f\*.txt"** : recherche tous les fichiers commençant par "f", se finissant par ".txt" à partir du répertoire courant (".")
- **grep** : recherche dans un fichier du répertoire courant une chaîne.
  - **grep "petite chaine" fic\*.cxx** : recherche "petite chaine" dans les fichiers commençant par "fic" et finissant par ".cxx"
- **chmod** : change les privilèges d'un fichier ou d'un répertoire.
 

**chmod ugo nom** : *ugo* sont 3 nombres correspondant à un privilège donné au fichier/répertoire pour le USER (u) le groupe auquel appartient le USER (g) et les autres (o) ; ils correspondent aux lettres *rwX* obtenu lors de la commande **ls -als**. Les valeurs possible pour ces nombres sont :

  - 0 : aucun privilège
  - 1 : exécution autorisée (x pour execute)

---

5. Cet éditeur n'est utilisable que sous windows ou mac ; il est destiné au traitement de texte et il est payant. On ne peut utiliser un éditeur-traitement de texte pour la programmation. Une version gratuite (moins performante) existe avec *Libreoffice* ou *Openoffice* disponible aussi sous Linux.

6. Éditeur de base disponible sur toutes les machines unix/linux et très puissant mais fort peu conviviale.

7. offrant de nombreuses possibilités, largement répandu mais peu "instinctif"

- 2 : écriture autorisée (w pour write)
  - 4 : lecture autorisée (r pour read)
- ou toute combinaison.

**Exemple :** `chmod 754 fic` permet au *USER* d’avoir tous les privilèges (7=1+2+4), aux membres de son groupe de lire et exécuter le fichier *fic* (5=4+1) et aux autres, uniquement une lecture de *fic*. On peut aussi utilisé la commande `chmod u+w fic` pour donner au *user* les droits d’écriture sur le fichier *fic* ou `chmod o-r fic`, pour enlever les droits de lecture aux *autres* ou `chmod a+x fic` pour donner les droits d’exécution à tout le monde (*user*, *groupe* et les *autres*) et ainsi de suite...

- **gzip/gunzip** : compression/décompression d’un fichier. La place ainsi occupée par le fichier compressé est généralement beaucoup plus petite.
  - `gzip fic.ps` : compresse *fic.ps* et le renomme *fic.ps.gz*
  - `gunzip fic.ps.gz` :décompresse *fic.ps.gz* et le renomme en *fic.ps*
- **tar** : création/extraction d’archives composées de plusieurs fichiers et /ou directory
  - `tar cvf monarchive.tar mydir` : met dans *monarchive.tar* (un fichier) l’ensemble du contenu (arborescence et fichiers) du directory *mydir*
  - `tar xvf monarchive.tar` : extraction de l’ensemble des fichiers et/ou répertoires contenu dans *monarchive.tar*
- **source** : exécute un script shell sans lancé un nouveau shell.

## 2.4 Enchaînement de commande

Il est possible de taper sur une même ligne plusieurs commandes indépendantes qui seront exécutées séquentiellement en les séparant par un point virgule.

### Exemple :

Essayer de taper cette ligne :

```
echo "echo un petit texte" > fic; ls -als fic; chmod 755 fic; ls -als fic; fic
```

On peut également utiliser le résultat d’une commande comme argument d’une autre en utilisant la commande “|” (ce prononce *pipe*).

### Exemple :

Essayer de taper :

```
touch fic1; touch ex1; touch ex2
ls | grep fic
ls | grep 1
```

## 3 Impression

- **lpr -Pimprimante** *nom\_du\_fichier.ps* ou *nom\_fichier.txt* : pour imprimer respectivement un fichier POST-CRIPT<sup>8</sup> et un fichier texte.
- Pour annuler une impression, faire `lpq` (donne la liste des travaux en cours) et `lprm -Pimprimante numero` où *numero* est le numero de ce que vous voulez annuler.

## 4 Interruption, passage en background d’un exécutable

**Passage en background** Nous avons déjà vu que le symbole “&” permet de lancer en *background* (i.e., en gardant la main dans une fenêtre terminal) un exécutable (par exemple, un éditeur). Cependant on a parfois oublié de mettre ce symbole et on souhaite récupérer la main : on tape alors `Ctrl+z` ; cela suspend l’exécution de la commande ; on peut alors l’envoyer en background en tapant `bg`. Essayer avec *nedit*.

8. C’est un format d’image directement interprétable par les imprimantes ; un fichier postscript à une extension “.ps” ou “.eps”.

**Interruption d'un exécutable** Il est parfois nécessaire d'interrompre l'exécution d'un programme ; pour cela on peut utiliser dans la fenêtre où il a été lancé les touches **Ctrl+c**. Ceci termine violemment l'exécution. Bien sûr, **Ctrl+c** ne peut être effectif que si l'exécutable ne tourne pas en background. Si c'était le cas, il est possible de le "tuer". La commande **ps** donne le nom des exécutables qui tournent et le numéro qui leur a été attribué (*PID*). Soit *X* ce numéro ; on arrête l'exécution du programme en tapant **kill -9 X**. Essayer avec *nedit*.

## 5 Les variables dans un shell

Nous donnerons à titre indicatif certaines syntaxes propres à *sh* (ou *bash*). Le shell est comme nous l'avons dit un moyen de communiquer avec l'ordinateur. Outre les instructions de bases que vous avez déjà vues, celui-ci permet d'utiliser des variables. Ces variables sont de 2 types ; il y a les *variables standards* et les *variables d'environnement*.

### 5.1 Complément : Les variables *standards*

Ce sont des variables que l'utilisateur se définit ; elles sont généralement utilisées dans des *scripts* ou programmes écrits dans un shell (*csh* ou *sh*). Elles ne "vivent" que dans le shell courant et sont perdues si on lance un nouveau shell. Leur affectation dépend du shell utilisé :

- en *sh* : *var=valeur*
- en *csh* :
  - si la variable est un nombre on utilise *@ var=valeur*
  - si c'est une chaîne de caractères *set var=valeur*

On utilise une variable ainsi affectée par ***{ var }*** ou plus simplement ***\$var***

On peut afficher la valeur d'une variable par la commande ***echo***

**Exemples :**

Essayer :

```
var1="ceci est une chaine"
var2="cela aussi"
echo "voici la variable var1: $var1 et la variable var2 : $var2"
```

Essayer aussi

```
echo 'voici la variable var1: $var1 et la variable var2 : $var2'
```

Pour les nombres :

```
a=1
b=2
let c=$a+$b
echo "$a + $b = $c"
```

### 5.2 Les Variables *d'environnement*

Les variables d'environnement sont particulières dans la mesure où elles sont utilisées par le shell lui même. Certaines sont déjà définies et sont indispensables, d'autres ne servent que dans certaines circonstances. Contrairement aux variables standards, elles "s'exportent" d'un shell au shell fils. Leur affectation dépend également du shell utilisé :

- en *sh* : *VAR=valeur ; export VAR* ou plus simplement *export VAR=valeur*
- en *csh* : *setenv VAR valeur*

La commande ***printenv VAR*** permet de visualiser la variable (***printenv*** sans argument donne la liste complète des variables d'environnement affectées). La commande ***echo*** peut aussi être utilisée (***echo \$VAR***)

## Exemples

- PATH : ensemble des répertoires où le système recherche un exécutable (commande LINUX ou programme)
- USER : votre login
- HOME : votre répertoire principal
- PWD : le répertoire courant
- SHELL : votre shell par défaut
- LD\_LIBRARY\_PATH : chemin où le système recherche les librairies dynamique (*libxxx.so*).

## 5.3 Le bash et le .bashrc

Si votre shell est le bash, à chaque fois qu'une *terminal* est ouverte, *bash* lance par défaut un certain nombre de script ; le dernier de ceux-ci est, s'il existe, le `~/bashrc`<sup>9</sup>. Si ce fichier n'existe pas vous pouvez le créer en vous inspirant du modèle ci-dessous :

```
# .bashrc
# User specific aliases and functions
# Source global definitions
if [ -f /etc/bashrc ]; then
. /etc/bashrc
fi
export EDITOR=nedit
PS1='\h[\W] \!: '      #command prompt.
ulimit -s unlimited   #may be useful for some codes
#
# Alias section
#
alias lpq="lpstat -o"
alias rm="rm -i"
alias dir="ls -alsh"
alias ls="ls --color=auto"
alias cls="clear"
alias h="history"
alias pss="ps -edlaf | grep $USER"
export PATH="/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin"   # where to look for executables
export PATH="${PATH}:/bin:."
```

Ce script ne doit pas être exécuter directement mais par la commande *source .bashrc* afin de conserver tous les *alias* et autres variables. Si cela vous semble utile vous rajouterez des commandes ou *alias*.

## 5.4 Le csh et le .cshrc

Si votre shell est le csh, à chaque fois qu'une *terminal* est ouverte, *csh* lance par défaut un certain nombre de script ; le dernier de ceux-ci est, s'il existe, le `~/cshrc`. En faisant *ls -als* dans votre *home directory* vous verrez que ce fichier n'existe pas. Vous allez en créer un suivant le modèle suivant :

```
set path=( $path $HOME/bin .) # where to look for executables
if ( $?prompt ) then         # shell is interactive.
set history=40               # previous commands to remember.
set savehist=40              # number to save across sessions
set system='hostname|cut -d"." -f 1' # name of this system.
set prompt='%m[%c1] \!: '    # command prompt.
```

---

9. Il est parfois nécessaire qu'un fichier `~/bash_profile` existe ; ce fichier contient alors

```
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi
```

```

alias xterm "xterm -sl 1000 -sb -n $HOST &"
alias dir "ls -als"
alias cls clear
alias pss "ps waux | grep $USER"
alias lpq lpstat -o
alias rm rm -i
alias cp cp -i
alias mv mv -i
endif

```

Ce script ne doit pas être exécuter directement mais par la commande *source .cshrc* afin de conserver tous les *alias* et autres variables. Si cela vous semble utile vous rajouterez des commandes ou *alias*.

## 6 Programmation : le début

Nous allons utiliser un langage évolué de programmation afin d'écrire des programmes un peu plus compliqués que les scripts shell. Ce langage est le *C++* (analogue au C).

### 6.1 Écriture

La première des choses est de taper le programme (en utilisant *nedit*).  
Voici un exemple que vous taperez dans un directory *tp1*

```

#include <iostream>
using namespace std;
int main()
{
    cout<<"Bonjour ca va?"<<endl;
}

```

Sauvegarder le sous *essai.cxx*

### 6.2 Compilation

Il faut ensuite le compiler, c'est-à-dire transformer le code compréhensible par vous en un code compréhensible par l'ordinateur. Pour cela on se place dans le répertoire *tp1* et on tape

```
g++ -o essai essai.cxx
```

Cela a pour effet de créer un nouveau fichier exécutable appelé *essai*, à partir de votre programme C++ *essai.cxx* (*-o* signifie *output name*). Si vous oubliez *-o xxx*, l'exécutable créé aura pour nom *a.out*.

- Si votre programme contient des erreurs, elles apparaissent expliquées avec le numéro de la ligne du programme correspondante. Il faut alors les corriger, sauvegarder le programme et le compiler à nouveau, jusqu'à ce qu'il n'y ait plus d'erreur.
- Il existe divers compilateur C++ ; nous utiliserons *g++*.

### 6.3 Exécution

Exécutez votre programme en lançant la commande *essai* dans la fenêtre terminal.