# Introduction à UNIX

***Licence de Physique***

**[Home|Unix|C++|Graphisme|Projets|Web|FAQ]**

## Contents



## 1 UNIX c'est quoi?

*UNIX* (ou son analogue sur PC, *LINUX*) est un **système d'exploitation** (au même titre que *DOS*, *Windows*) 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.

**Remarque :**

- nous avons dit que *LINUX* est l'analogue *UNIX* pour les PC. En fait *LINUX* est une version complètement gratuite de *UNIX* qui peut s'installer sur n'importe quelles machines. Généralement les constructeurs de stations de travail (*IBM*, *HP*, *Compaq*, *SUN*) livrent leur machine avec une version d'*UNIX* alors que sur les PC on a plutôt tendance à installer *LINUX*. Les seules différences entre *LINUX* et *UNIX* sont dans certaines options de certaines commandes peu standards (mais notez qu'entre un *UNIX IBM* et *HP* ou *Compaq*, ces différences existent également).



### 1.1 Comment dialoguer avec l'ordinateur ?

Ceci se fait dans une fenêtre dite terminal[1] via un langage le *shell*. Il existe, sous *UNIX*, 3 principaux shell : le *Bourne shell* (*sh*) qui est le plus ancien, 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). C'est principalement avec ce dernier que nous travaillerons.

### 1.2 Les utilisateurs

Pour vous connecter sur une machine *UNIX* 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 *privilèges* (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) selon une ***structure arborescente***. Chaque noeud 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/u3/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 ... Le répertoire *racine* est /.

## 2 Principales commandes

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 :**

- **UNIX (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[2]

[1] Il existe divers types de *fenêtre terminal* selon les machines et les environnements de travail (par exemple sur *HP* ce sera **hpterm** ou parfois **dtterm**). 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 ***xterm***.

[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
```

- ***ls -F*** pour voir rapidement les fichiers exécutables (ils sont suivis d'une étoile), les répertoires (ils sont suivis d'un /).
- ***ls [-alsF]***[3] *filename* pour ne voir que le fichier *filename*[4]

- 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)

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 1ere 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 1er 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*
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 *Wilcards*, 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.



- ***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

Remarquez qu'**en pratique on utilise souvent un éditeur** (voir le paragraphe2.2 pour info) pour créer/visualiser un fichier.

**Exercices**

- Avant de commencer l'exercice vous devez taper : *source ~meplan/bin/debut*
- 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*.
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 exécutables UNIX (*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*[5], *vi*[6], *emacs*[7], et enfin *nedit*[8] qui permet de faire beaucoup de choses, qui est assez conviviale et instinctif. Nous utiliserons donc *nedit* pour taper tous nos fichiers. 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 *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.

[5] Cet éditeur n'est utilisable que sous windows ou macintosh ; il est destiné au traitement de texte ; il est payant, d'une utilisation souvent lourde et très mal adapté à la programmation. Il est malheureusement très répandu.
[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 étant peu "instinctif"
[8] On peut trouver cet éditeur pour diverses machines à http://nedit.org

### 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 doit utiliser ***rm -i*** *fic* qui demande une confirmation. On peut redéfinir la commande grâce à un *alias*:

  - ***alias rm "rm -i"***

  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* ***-print***: cherche tous les fichiers *filename* depuis le répertoire pointé par *chemin*.
  - ***find*** . ***-name*** *"f*.txt"* ***-print*** : 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*.C* : recherche "*petite chaine*" dans les fichiers commençant par "*fic*" et finissant par "*.C*"

- ***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)
  - 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*.

- ***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

Plusieurs imprimantes existent : ***phcarism*** (fonctionnement aléatoire), ***neel*** et ***perrin***.

- ***lpr -Pnom_imprimante*** *nom_du_fichier.ps* ou *nom_fichier.txt* : pour imprimer respectivement un fichier POSTCRIPT[9] et un fichier texte.
- Pour annuler une impression, faire ***lpq*** (donne la liste des travaux en cours) et ***lprm -Pnom_imprimante*** *numero* où *numero* est le numero de ce que vous voulez annuler.
- On peut aussi utiliser l'interface de KDE : **kprinter** *nom_fichier*

## 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 *xterm*) 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*.

**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*, mais nous ne travaillerons qu'en *csh*. 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***.

[9] C'est un format d'image directement interprétable par les imprimantes ; un fichier postcript à une extension ".ps" ou ".eps".

## 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 :

```
set var1="ceci est une chaine"
set 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
@ c=$a + $b
echo "$a + $b = $c"
```

En *sh* la différence ne viendrait qu'à l'affectation.
On remarquera que *l'espace* de part et d'autre du signe "+"est *nécessaire*.

## 5.2 Les Variables *d'environnement*

Les variables d'environnement sont particulières dans la mesure ou 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*
- 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 UNIX ou programme)
- USER: votre login
- HOME: votre répertoire principal
- PWD: le répertoire courant
- SHELL: votre shell par défaut
- DISPLAY: l'adresse où s'affiche les fenêtres (si l'adresse est *zorro.disney.fr* on écrira ***setenv DISPLAY*** *zorro.disney.fr:0.0*).



**Exercice**

- Faire afficher le contenu de votre variable ***DISPLAY*** (le résultat ressemble à *zorro.disney.fr:0.0)*
- Demander à votre voisin le contenu de sa variable ***DISPLAY*** et affecter le à votre variable ***DISPLAY***. Taper alors *nedit* ou *xterm*
- Affecter à votre ***DISPLAY*** sa valeur initiale.

## 5.3 Complément : Écriture de scripts shell

Nous allons, à titre d'illustration écrire quelques petits scripts en *csh*. Cela vous permettra de vous familiariser avec certaines des commandes que vous avez vues et vous rendra des services par la suite. Un script shell est un fichier contenant une suite de commande (ici *csh*). Il commence par la ligne
#!/bin/csh
indiquant que ce script est écrit en *csh*. Le "#" est en principe la façon de noter un commentaire en sh/csh ; cependant dans ce cas particulier ce commentaire est interprêté par *UNIX*. Il est possible de passer des arguments à un script ; dans ce cas leur noms dans le script est **$1** pour le premier argument, **$2** pour le second, etc... Par exemple, si le script "*essai*" est appelé par *essai 12 texte*, **$1** aura la valeur *12* et **$2** la valeur *texte*.

1. Écrire un script affichant sur une ligne "*Bonjour"* et sur la suivante "*1er paramètre*" suivi de l'argument passé. Compléter ce script pour que, s'il n'y a pas d'argument, on affiche le message "*pas d'argument*". Une condition se note :

   ```
   if ( expression ) then
       commande
   endif
   ```

   où *expression* est un booléen (les tests ont la même syntaxe qu'en C : ==, ***!***=, <=,>=,>,<)

2. Vous allez introduire dans le script, 3 variables *nom_complet*, *extention* et *nom_ss_extension*. Le but est d'affecter chacune de ces variables avec la partie utile de l'argument (par exemple, *essai toto.txt*, doit mettre *toto.txt* dans *nom_complet*, *toto* dans *nom_ss_extension* et *txt* dans *extention*). Pour cela, vous aurez besoin de la commande ***cut*** ainsi que des "*back quote*": ` ` ; ces derniers provoque l'exécution prioritaire de la commande qui est entre (par exemple ***set*** *a=*`***ls***` affectera à *a* le résultat de la commande ***ls***, i.e. le contenu du répertoire courant). La commande ***cut*** a 3 arguments et elle permet de découper un texte en différentes parties (***-f*** *i*), suivant un délimiteur (***-d****"char"*). Vous l'utiliserez en vous inspirant des exemples suivants :

   ```
   echo "ceciXestXunXexemple" | cut -d"X" -f1
   echo "ceciXestXunXexemple" | cut -d"X" -f2
   ```

3. Modifier le script précédent de sorte que:
   - si *nom_ss_extension* est égale à *nom_complet* alors on impose à *extension* "C" et enfin *nom_complet* prend la valeur *$nom_ss_extension.$extension*

## 5.4 Le .cshrc

A chaque fois qu'une ***xterm*** 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.

    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.C*

## 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.C*

Cela a pour effet de créer un nouveau fichier exécutable appelé *essai*, à partir de votre programme C++ *essai.C* (***-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, saugarder 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 *xterm*.