

# TP 1 : prise en main

Tous les TP vont s'appuyer sur trois outils principaux :

- Une interface de commande en ligne (**CLI** - Command Line Interface) ou **shell**
- Un éditeur de texte (comme **nano**, emacs, vi, ou autre de votre choix ...)
- Un outil de compilation (**gcc**)

L'interface nous permettra de taper les différentes commandes nécessaires à l'écriture, la compilation et l'exécution de notre code ; l'éditeur à créer ou modifier notre code et l'outil **gcc** à traduire notre code en langage machine.

## 1. Commandes Unix

### 1.1 Qu'est-ce qu'un shell ?

Le shell (littéralement « coquille ») est la liaison la plus élémentaire entre vous (l'utilisateur) et le système d'exploitation (**OS** – **O**perating **S**ystem).

Vous allez taper des commandes qui seront interprétées par le shell et transmises à l'OS. Il est intéressant de connaître les possibilités offertes par le shell. Il est ainsi possible d'accomplir (assez) facilement de nombreuses opérations complexes.

Un shell se reconnaît d'abord par son « prompt » caractéristique. Selon les réglages, il peut ressembler à ceci :

```
[user@ordinateur repertoire]$ _  
[repertoire % _  
% _  
# _  
...
```

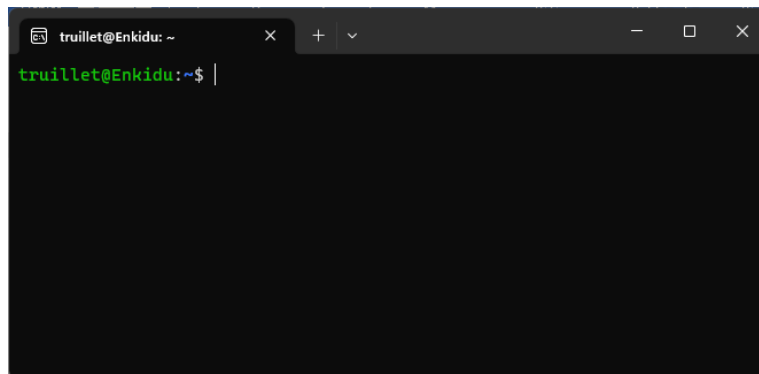


Figure 1 – Exemple de fenêtre shell

### 1.2 Saisie des commandes

En affichant le prompt, le shell montre qu'il est prêt à recevoir des commandes. Le shell interprète la série de caractères tapés à chaque fois qu'un « retour chariot » (touche Entrée) est tapée.

Lors de la saisie des commandes, vous aurez à respecter un certain **nombre de règles**. Ces règles sont regroupées dans ce qui est appelée « la syntaxe de commande ». En voici quelques éléments.

- Dans la ligne en cours, le shell essaye de reconnaître des « mots ». Pour les différencier, **vous les séparerez par des espaces**
- Le premier mot de la ligne est le nom de la commande qui est systématiquement suivi par une espace. Les noms de la plupart des commandes sont en **lettres minuscules** (Unix fait la différence entre majuscules et minuscules)

- Les autres mots faisant partie de la ligne de commandes sont des paramètres. Ils déterminent le mode de travail des commandes. On en distingue deux sous-groupes : les options précédées du signe « - » et les arguments.

Quelques exemples :

```
% ls -l
% cd /home
```

De nombreuses commandes (mais pas toutes 😊) se comportent de la même manière. Chaque fois que vous avez un doute, essayez de taper le nom de la commande suivie de `--help` ou `-h`

## 2. Quelques commandes systèmes utiles

### 2.1 Répertoires

Lorsqu'un utilisateur ouvre un terminal ou se connecte à une machine via un terminal, il est positionné par défaut dans un répertoire qualifié de **home directory** (souvent noté `~`).

Quelques commandes utiles :

- **pwd** (**P**rint **W**orking **D**irectory) pour donner votre position dans l'arborescence
- **ls** (**L**i**S**t) pour lister les noms de fichiers du répertoire courant
- **mkdir** *dir* (**M**a**K**e **D**IRectory) pour créer le répertoire *dir*
- **cd** *dir* (**C**hange **D**irectory) pour aller dans le répertoire *dir*
- **cp** *fich1* *fich2* (**C**opy) pour copier le fichier *fich1* vers le nouveau fichier *fich2*
- **mv** *fich1* *fich2* (**M**ove) pour changer le nom du fichier *fich1* vers *fich2*

Pour chacune des commandes, il existe un descriptif détaillé, accessible via la commande **man**, par exemple **man ls**.

### 2.2 Exercice

Connectez-vous sur une session Unix. Depuis le terminal :

- trouvez votre position dans l'arbre depuis la racine /
- créez le répertoire **TP1** puis allez dans ce répertoire
- listez les fichiers du répertoire (y compris les fichiers cachés) ?

### 2.3 Redirection des entrées-sorties et « pipes »

Comme vous avez pu le constater, toutes les commandes émises depuis une fenêtre terminal, effectuent leurs sorties sur le terminal (**stdout**). Pour les rediriger sur d'autres supports on peut les faire suivre, entre autres, des symboles `>` ou `>>` :

`>` indique que la sortie n'est plus le terminal mais un autre support (fichier, etc.)

`>>` est utilisé pour concaténer la sortie d'une commande à un support déjà existant (rallongement)

De façon analogue, on peut rediriger le résultat d'une commande en entrée d'une autre commande. Le symbole utilisé est `|` et il s'agit d'un « pipe ».

### 2.4 Exercice

Les commandes qui vont être utilisées ici :

- **date** pour afficher la date
- **cat** pour afficher le contenu d'un fichier

Placez-vous dans votre sous-répertoire **TP1**. Envoyez le résultat de la commande **date** dans le fichier **output.txt**. Vérifiez le contenu du fichier créé avec la commande **cat**. Quels sont les droits du fichier **output.txt** ?

Vous allez maintenant utiliser un « pipe », outil très utile pour rediriger ou filtrer les sorties vers une autre commande. Pour cela, revenez dans votre répertoire « home » (**cd** ou **cd ~** par exemple). Localisez votre fichier **output.txt** en utilisant la commande **ls -lR | grep output.txt**

## 2.5 Archiver des données

Il peut être utile d'archiver et de compresser des données dans une archive unique afin de pouvoir facilement l'envoyer par mél ou les mettre sur clés USB ou disque dur.

Deux commandes sont fréquemment utilisées : les commandes (**tar** – **T**aper **AR**chiver) et **gzip/gunzip** (**GNU** (**un**)**zip**).

**tar** permet de manipuler des archives (à l'origine, conçue pour l'archivage sur bande magnétique). Les options les plus courantes manipulées par **tar** sont **c** (**C**reate), **t** (**l**is**T**) et **x** (**eX**tract), **v** (**V**erbose) et **z** (**Z**ip)

la commande **gzip** remplace un nom de fichier par le même nom avec l'extension **.gz** (celle-ci a le plus souvent une taille inférieure). C'est un format compressé sans perte d'informations.

## 2.6 Exercice

Positionnez-vous sur le répertoire **~**. Tapez la commande **tar cvf TP1.tar TP1**. Que se passe t'il ? Notez la taille de ce fichier.

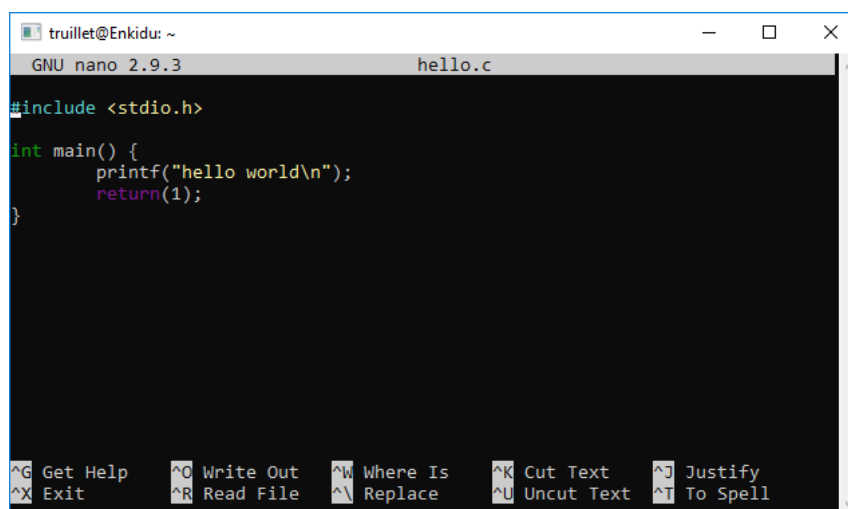
Tapez maintenant la commande **gzip TP1.tar**. Notez la taille et comparez-là avec **TP1.tar**

Détruisez le répertoire **TP1** (et tous les fichiers) en tapant la commande **rm -R TP1**. Que s'est-il passé ?

Enfin, tapez la commande **tar xzvf TP1.tar.gz**. Quel est le résultat ?

## 3. Première compilation d'un programme C

Tapez dans un fichier **hello.c** le code (**mythique**) suivant (en utilisant votre éditeur de texte préféré cf. Figure 2) :



```
truillet@Enkidu: ~
GNU nano 2.9.3      hello.c

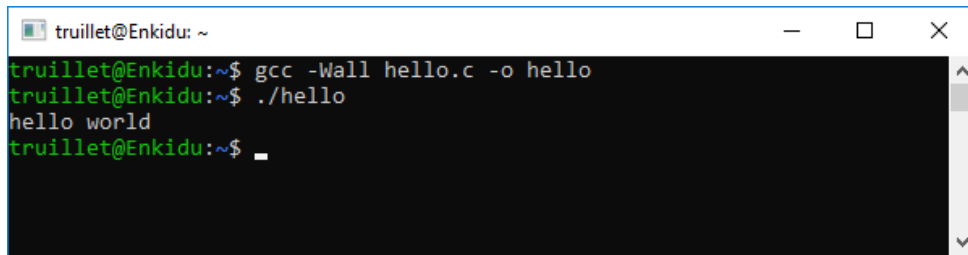
#include <stdio.h>

int main() {
    printf("hello world\n");
    return(1);
}

^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify
^X Exit          ^R Read File     ^\ Replace       ^U Uncut Text    ^T To Spell
```

Figure 2 – « Hello world »

Compilez et exécutez ensuite le code produit (cf. Figure 3).

A terminal window titled 'truillet@Enkidu: ~' with standard window controls. It shows the following commands and output:

```
truillet@Enkidu:~$ gcc -Wall hello.c -o hello
truillet@Enkidu:~$ ./hello
hello world
truillet@Enkidu:~$
```

Figure 3 – chaîne de compilation et d'exécution

**Nota1** : **gcc** est l'outil de compilation que nous utiliserons en TP. Son fonctionnement précis sera précisé ultérieurement. L'option **-o hello** indique à **gcc** que le fichier produit devra s'appeler *hello* (et non **a.out** par défaut).

**Nota2** : pourquoi faut-il appeler l'outil **gcc** avec la commande « **gcc** », alors qu'il faut appeler notre programme avec la commande **./hello** ?

A quoi sert ce « **./** » ? La raison est assez simple. Afin de pouvoir exécuter un programme en ligne de commande, il faut donner le chemin de ce programme sur le disque. C'est ce que nous faisons avec **./hello** en référant un chemin relatif grâce à « **./** ».

Comme **gcc** est un outil couramment utilisé, et il a été placé dans un répertoire que l'environnement d'exécution des commandes connaît. L'ensemble des chemins dans lesquels l'environnement de commande cherche les binaires que vous lancez est stocké dans la variable d'environnement **PATH**. Pour connaître la liste des répertoires dans lesquels l'environnement en ligne de commandes cherche les binaires, tapez **echo \$PATH**.

Vous voyez s'afficher une liste de chemins, séparés par « **:** ». Lorsque vous lancez **gcc**, l'environnement d'exécution des commandes cherche alors dans chaque chemin précisé, l'un après l'autre, s'il contient un fichier appelé **gcc**. Dès qu'il en trouve un (le premier donc !), il l'exécute.