


<p align="center"><b>Institut Supérieur d'Informatique et de Multimédia de Gabes</b></p> 	<b>Année Universitaire : 2025/2026</b>
	<b>Module : Système d'exploitation 2</b>
	<b>Enseignante : Yomna BEN JMAA</b>
	<b>TP1 : Outils de développement en C sous Linux</b>

## Objectifs

1. Syntaxe des commandes Unix : principe d'exécution.
2. Utilisation des commandes de base: manipulation des fichiers et des dossiers.
3. Compilation et exécution des programmes en langage C

### I. Syntaxe des commandes Unix

L'exécution des commandes Unix s'effectue par la saisie des commandes sur l'invite de commande (appelée aussi terminal). Ces commandes sont interprétées par un interpréteur de commandes appelé shell (sur les distributions ubuntu, le shell par défaut est le shell bash).

Les commandes Unix s'exécutent d'une façon simple: on saisit la commande sur la console et on appuie la touche entrée pour avoir le résultat sur le terminal.

On peut faire l'exécution d'une suite de commandes. Il suffit de les taper les unes à la suite des autres séparées par ;.

On distingue deux manières d'exécution: certaines commandes peuvent s'exécuter avec ou sans arguments. Les arguments sont les entités sur lesquelles s'appliquent l'action. Un argument peut être un fichier, un texte, un dossier...

Certaines commandent nécessitent l'utilisation de plusieurs arguments.

**Exemple:** la commande **cp** permet de créer une copie d'un fichier. Elle a besoin comme paramètres le nom du fichier à copier et le nom de la copie: **cp fichier1 fichier2**

D'une façon générale, la syntaxe des commandes Unix s'écrit comme suit:

**nom\_commande [-options] argument1 argument2 .....**

Le résultat obtenu suite à l'exécution d'une commande peut varier grâce à l'utilisation des options. Les options sont souvent des lettres précédées d'un tiret (**-l, -s, ...**).

Une commande peut être utilisée avec plusieurs options en les séparant par des espaces (**ls -l -a**) ou les combinant ensemble (**ls -la**).

Lorsque les commandes exigent l'utilisation des arguments, la non-saisie de ces derniers engendrent la génération des erreurs.

Afin de savoir le rôle des commandes et leurs options, il suffit d'accéder à l'aide avec la commande **man** suivie du nom de la commande.

Afin de quitter l'aide, on tape la lettre **q**.

---

## II. Manipulation des fichiers

---

Les commandes de base sont destinées pour la création, la modification et l'affichage des informations relatives aux fichiers.

### La création: commandes **touch** et **gedit**

La commande **touch** suivie du nom du fichier permet la création d'un fichier texte vide.

La commande **gedit** suivie du nom du fichier permet l'ouverture de l'éditeur texte pour faire la saisie du texte. Si le fichier existe déjà, le fichier sera ouvert dans l'éditeur.

### La modification:

- **La suppression:** commande **rm** suivie du nom du fichier.
- **La modification** du contenu:
  - commande **gedit** suivie du nom du fichier qui existe déjà.
  - commande **echo texte\_à\_saisir > nom\_fichier** pour remplacer le contenu du fichier par le texte saisi (s'il y a déjà un texte existant dans le fichier, il va être écrasé).
  - commande **echo texte\_à\_saisir >> nom\_fichier** pour ajouter le texte saisi à la fin du fichier (l'ancien contenu est conservé).
- **Le renommage:** commande **mv** avec la syntaxe suivante **mv nom\_du\_fichier nouveau\_nom**.
- **Le déplacement:** commande **mv** avec la syntaxe suivante: **mv nom\_du\_fichier chemin\_nouveau\_emplacement**.
- **La copie:** commande **cp** avec la syntaxe suivante: **cp nom\_fichier chemin\_destination\_copie**.

**L'affichage:** commandes **more**, **less**, **tail**, **head** et **cat** suivies du nom du fichier.  
Commande informative: la commande **ls** suivie du nom du fichier permet d'afficher les détails du fichier (taille, droits d'accès, date de la dernière modification...).

---

## III. Commandes de base pour la manipulation des dossiers

---

Sur UNIX, les noms des répertoires sont séparés par le '/' . Le répertoire home (de l'utilisateur) est représenté par le symbole '~'.

- Pour connaître le nom du dossier actuel, on tape la commande **pwd**: "Print Working Directory".

- Pour changer de dossier, on utilise la commande **cd**: "Change Directory". Il existe en fait 2 façons de changer de dossier : en indiquant un chemin relatif (relatif au dossier actuel ) ou absolu (commence toujours par la racine '/').

On tape :

- **cd /** : Pour aller à la racine.
- **cd ..** : Revenir au dossier précédent (dossier parent)
- **cd ../..** : Reculer de 2 dossiers parents.
- Pour lister le contenu du répertoire courant, on tape la commande **ls**.
- Pour lister le contenu d'un répertoire, on tape la commande **ls nom\_répertoire**.
- Pour créer un répertoire, on utilise la commande **mkdir**.
- Pour supprimer un répertoire, on utilise commande **rmdir** s'il est vide. Sinon on utilise la commande **rm -r** pour le supprimer avec son contenu.

---

### Exercice d'initiation pour les commandes UNIX

---

- Quel est votre répertoire courant?
- Lister le répertoire courant.
- Créer dans votre répertoire personnel un dossier dont le nom est tp1.
- Créer deux dossiers rep1 et rep2 dans le dossier tp1.
- Se positionner sous rep1.
- Revenir sous le répertoire tp1 et supprimer rep1.
- Créer dans votre répertoire personnel un fichier ex1.
- Copier le fichier ex1 dans le dossier rep2.
- Renommer le fichier ex1.

---

### Exercice 1 :

---

- 1- Créer le fichier source **salut.c** ci-dessous :

```
#include <stdio.h>

int main(void) {

    printf("salut\n");
    return (0);
}
```

- 2- Compiler **salut.c** sans spécifier le nom du fichier exécutable ? Comment l'exécuter dans ce cas?
- 3- Compiler salut.c en spécifiant salut comme nom du fichier exécutable ? Comment l'exécuter dans ce cas?
- 4- Générer le fichier assembleur puis objet de votre source
- 5- Recompiler ce programme en spécifiant l'option -v ? en déduire le rôle de cette option

---

### Exercice 2 :

---

- 1) Ecrire une fonction bonjour qui affiche autant de fois le texte bonjour que le nombre passé en argument.

- 2) Enregistrer cette fonction dans un fichier **b.c**
- 3) Ecrire une fonction aurevoir qui affiche autant de fois le texte aurevoir que le nombre passé en argument
- 4) Enregistrer cette fonction dans un fichier **a.c**
- 5) Compiler ces fichiers afin d'avoir les fichiers objets
- 6) Créer la librairie statique **libsalut.a** à l'aide de la commande **ar**
- 7) Vérifier le contenu de votre librairie
- 8) Créer le fichier exemple.c dont le contenu est :

```
#include <stdio.h>
int main(void) {
    int a=0;
    while(a<1 || a>5) {
        printf("donner un nombre entre 1 et 5:");
        scanf("%d",&a);
        printf("\n");
    }
    bonjour(a);
    aurevoir(++a);
    return(0);
}
```

- 9) Compiler ce programme

---

### Exercice 3 :

---

On sait que tout programme C contient une fonction main() et c'est la première fonction exécutée.

Le shell lance l'exécution d'un programme, et en plus il lui transmet une liste de paramètres éventuels fournis sur la même ligne commande. Ces paramètres sont transmis, après analyse et prétraitement, à la fonction main() du programme considéré.

Comment les récupérer, sachant qu'en plus ils sont en nombre quelconque?

Il suffit tout simplement, ce que fait le shell, d'en indiquer le nombre et d'en fournir la liste. Ce nombre est conventionnellement désigné **argc** (argument count) et la liste est un tableau de longueur variable désigné **argv[]** (argument value). Il contient la liste de tous les paramètres de la commande, y compris le nom de celle-ci. La fin du tableau est marquée par la chaîne vide NULL ('\0').

D'autres informations sont éventuellement récupérables par **main()**. Ce sont les variables d'environnement comme **PATH**, **TERM** qu'on peut récupérer par l'intermédiaire d'un troisième argument **arge** (argument environment).

```
main (int argc, char* argv[], char* arge[]) {...}
```

Où `argv`, comme `argv`, est un tableau de chaînes de caractères (dernier élément `NULL`) de forme `VARIABLE=valeur`.

Question : Ecrire un programme qui affiche la liste de ses arguments et la liste des variables de son environnement