Commandes de base :

man : help

ls : lister le répertoire
ls -l : affichage détaillé du répertoire
ls -l -h : affichage détaillé+ taille des fichiers
ls -a : affichage des fichiers cachés
ls -lct : trier les fichiers par date de modification décroissante
ls -i : afficher l'inode du fichier
cd : revenir au répertoire utilisateur
cd – : revenir au répertoire précédent
cd .. : remonter au répertoire parent
cd / : remonter à la racine
cd /chemin : se placer dans le répertoire indiqué par le chemin

mv : déplacer ou renommer des fichiers et des répertoires
mv fichier rep/ *: déplacer fichier dans rep*
mv rep/fichier . *: déplacer fichier qui se trouve dans rep dans le répertoire courant*
mv rep1 rep2 *: renommer rep1 en rep2*

cp : copier des fichiers ou répertoires
cp -a : copier en gardant tout les droits, dates, proprios, groupes...
cp -i : demende une confirmation avant d'ecraser
cp -r : copier un répertoire et tout son contenu
cp fichier rep/ *: copier fichier dans rep*
cp -r rep1 rep2 *: copier rep1 et son contenu dans rep2*

rm : supprimer des fichiers
rm -f : demande une confirmation avant d'effacer
rm -r : supprimer récursivement (un répertoire et son contenu)
rm fichier *: supprimer fichier*
rm -r /rep *: supprimer rep et son contenu*

mkdir : créer un répertoire
mkdir -p : créer les répertoires parents s'ils n'exisent pas
mkdir rep : *crée rep*
mkdir -p rep1/rep2 *: crée rep2 et rep1 s'il n'existe pas*

rmdir : supprimer un répertoire vide
rmdir -p : supprimer les répertoires parents s'ils deviennt vides

pwd : afficher le répertoire en cours
id : affiche les informations de l'utilisateur

cat : afficher le contenu d'un fichier
cat > fichier : ajouter du texte dans fichier
cat -n : afficher les numéros des lignes

touch : créer un fichier

ln : créer un lien

ln -s : créer un lien symbolique
more : affichage page par page
less : afficher un ficher en permettant la navigation
sort : trier les fichiers
tty : afficher le numéro du terminal

« > » :

cat >fichier : ajouter du texte dans un fichier
commande fichier >fich1 : dirige le résultat de commande dans fich1
commande fichier >fich1 2>err : dirige le résultat de commande dans fich1 et les erreurs dans err
&>fich ou >>fich : ajouter dans fich sans ecraser le contenu
commande >fich1>fich2 : dirige le contenu de fich1 vers fich2

« | » : les pipes permettent d'exécuter plusieurs commandes à la fois

ls -l | sort | more : lister les fichiers et répertoire du répertoire courant, les trier et afficher page par page

Expressions régulières :

Expression	signification
*	N'importe quelle chaine de caractères
?	N'importe quel caractère
[abc]	A ou b ou c
[a-r]	Les caractères compris entre a et r
[^0-9]	N'importe quel caractère sauf ceux compris entre 0 et 9

Exemple :
ls *5 : lister les fichiers se terminant par 5
ls m* : lister les fichiers commençant par m
ls [a-d]* : lister les fichiers commençant par a, b, c ou d
ls m[^0-9]* : lister les fichiers commençant par m et ne conprenant qucun chiffre
ls ?est : lister les fichiers commençant par n'importe quel caractère et se terminant par est

Archivage et compression:

1- Archivage : créer un fichier archive (nomfich.tar) dans lequel tous les fichiers qu'on veut compresser sont mis : **tar cvf nomArchive.tar nomDossier** (nomDossier est le dossier ou le chemin du dossier qu'on va archiver)

2- Compression : créer un dossier compressé (nomfich.tar.gz) : **gzip nomArchive.tar**

tar tvf nomArchive : lister le contenu de l'archive
tar xvf nomArchive : extraire le contenu de l'archive

les commandes système :

umask : *créer un masque qui sera retranché au masque maximal (777) lors de la création d'un fichier (les droits lors de la création sont = 777 – valeur de umask)*

chmod : modifier les droits d'accès

+ : ajouter un droit
- : enlever un droit
= : n'autoriser que les droits indiqués

r : lecture ; caleur octale 4
w : écriture ; valeur octale 2
x : execution ; valeur octale 1
u : propriétaire du fichier
g : groupe propriétaire du fichier
o : les autres utilisateurs
a : tous le monde

Exemples :
chmod ugo+x fich : ajouter le droit d'exécution de fich à tous (ou : **chmod a+x**)

chmod go-wx fich : retirer les droits d'écriture et d'exécution de fich au groupe et aux autres (ou n'autoriser que la lecture : **chmod go=r fich)**

chmod u=rwx,go=r : donner tout les droits au user et uniquement celui de lire au groupe et au autres

chmod 777 : donner tout les droits à tous le monde
chmod 641 : donner les droits de lire et d'écrire au user (r=4 + w=2) , le droit de lire au groupe(r=4) et celui d'exécuter au autres (x=1)

Programmation C :

Génération du fichier objet et exécution

1- **gcc –c fichier.c –o fichier.o** : Générer le fichier objet (fichier.o)
2- **gcc fichier.c –o fichier :** générer l'exécutable
3- **./fichier :** exécuter

Liens dynamiques et statiques

1- **gcc –static test.o –o test.stat:** édition des liens statiques
2- **gcc –dyn test.o –o test.dyn :** édition des liens statiques
3- **objdump –d test.dyn** : Visualisation de test.dyn (même chose pour test.stat), le résultat est un code en assembleur

Comparaison des tailles de **test.dyn** et **test.stat :** « ls –l test.dyn test.stat »
La taille du fichier statique est beaucoup plus grande que celle du fichier dynamique

Explication :

Statique : lors le la création des liens statiques, toutes les fonctions de la bibliothèque utilisée sont copiées, la bibliothèque est liée à l'exécutable généré, ce qui fait qu'on obtient un fichier exécutable relativement "lourd".
Dynamique : l'édition des liens dynamique ne nécessite pas la copie de toute la bibliothèque, le nom de la bibliothèque est placé dans le fichier binaire, et la liaison se fait lors de l'exécution, ainsi le fichier exécutable généré est plus léger.

Création d'une bibliothèque

1- créer le header add.h contenant le prototype de la fonction (***ex*** : int add(int x, inty) ;)
2- créer le fichier add.c contenant le code la fonction donc le prototype est dans bib.h (***ex*** : int add(int x, int y){return x+y ;})
3- générer le fichier objet (***ex*** : **gcc –c bib.c –o bib.o**)
4- utiliser la bibliothèque en l'incluent dans le code du programme : **#include ''add.h ''**

bibliothèque statique : « libarithm.a »

1- **ar r libarithm.a add.o**: création de la bibliothèque statique
2- **gcc test.o libarithm.a –o test** : complier le programme avec la bibliothèque statique (il faut indiquer le chemin vers la bibliothèque si elle ne se trouve pas dans le même répertoire que le programme)
3- **./test :** exécuter

bibliothèque dynamique : « libarithm.so »

1- **gcc – shared –o libarithm.so add.o:** création de la bibliothèque dynamique
2- édition des liens en utilisant la bibliothèque dynamique
   1- placer libarithm.so dans le même répertoire que libarithm.a
   2- exécuter la commande **ldconfig –n nomRep** pour créer le lien entre les bibliothèques, l'option –n pour préciser le répertoire
   3- recompiler et exécuter

Gestion des processus :

bg %N : exécuter le programme suspendu du numéro system N en arrière plan (background)

fg % N : déplacer le programme du numéro system N en premier plan (foreground)

ps : lister les processus en cors d'exécution ainsi que ceux en pause et afficher leurs PID

ps –f : même affichage que « ps » + les PPID (PID des processus parents)

ps –A : lister tout les processus (même ceux exécutés sur un autre terminal)

ps –o stat NumPID : affiche l'état du processus dont le PID est « NumPID »

Les états :

Pause : STAT T
Arrêt : STAT S
Actif : STAT R

Jobs : énumérer les tâches qui s'exécutent en premier et arrière plan

nohup : lancer un processus qui restera actif même après la déconnexion de l'utilisateur

top : montre la charge du CPU

pstree : affiche une arborescence des processus en cours d'exécution

trap ' ' Numsignal : permet de désactiver le signal (trap ' ' 2 désactive ctrl+c)

trap numsignal : réactiver le signal

kill [-9] PID : terminer le programme, qu'il soit en cours exécution ou en pause, l'option -9 est pour forcer l'arrêt.

Kill [–nomSignal] PID : envoie le signal désigné par nomSignal (*ex* : kill –SIGUSR1 PID , envoie le signal numéro 15)

killall nomProcessus: envoyer un signal à tous les processus

./programme& : exécuter programme en arrière plan

Ctrl+z : mettre en pause le processus

Ctrl+c : arrêter le processus

Les commandes et fonctions (dans les programmes C):

System *: prend en entrée une chaine de caractère représentant une commande linux, retourne 1 s'il y a erreur d'exécution de la commande, 0 sinon (l'exécution du programme contenant la fonction exécutera la commande donnée en entrée)*

Fork() : *permet de dupliquer le processus faisant appel à elle. Elle crée un processus équivalent (processus fils) qui sera exécuter à partir de la ligne ou se trouve la fonction fork()*

Elle renvoie -1 en cas d'erreur, 0 si le processus exécuté et le processus fils, le PID du fils si le processus exécuté et le père

Getpid() : *renvoie le PID*

Getppid() : *renvoie le PPID*