

Cahier de TP

Loïc Weber & Thibault Gounant

December 19, 2023

Table des matières

1	TP1 : Unix, le système de fichier	1
1.1	Manuel	1
1.2	Hierarchie	1
1.3	Affichage	2
1.4	Gestion	2
2	TP2 : Commandes utilisateur Unix	2
2.1	Droits des fichiers	2
2.2	Système	2
2.3	Redirection	3
2.4	Recherche	3
3	TP3 : Commandes utiles	3
3.1	Traitement des fichiers	3
3.2	Compression et archivage	3
4	TP4 : Le Shell	4
4.1	Les Tests	4
4.2	Les Boucles	5
4.3	Parcourir les arguments	5
4.4	Lire un fichier	6
5	TP5 : Les processus	6
5.1	Les signaux	6

1 TP1 : Unix, le système de fichier

1.1 Manuel

```
man <commande>      # Afficher les pages du manuel correspondant à la commande
whatis <commande>    # Afficher la description du manuel correspondant à la commande
apropos <mot>        # Afficher les descriptions du manuel dont les pages contiennent le mot
history              # Afficher l'historique des commandes
alias <commande>=<expr> # Créer une commande correspondant à une expression
```

1.2 Hierarchie

```
/      # Répertoire racine
..     # Répertoire parent
~      # Répertoire maison

ls <dossier>      # Afficher le contenu du dossier
pwd              # Afficher le chemin absolu du dossier courant
cd <dossier>      # Changer de dossier
tree             # Afficher l'arborescence de répertoires
```

1.3 Affichage

```
file <fichier>  # Afficher une description du type du fichier
cat <fichier>   # Afficher le contenu du fichier
sort <fichier>  # Afficher le contenu du fichier trié
more <fichier>  # Visualiser le contenu du fichier par le haut
less <fichier>  # Visualiser le contenu du fichier par le bas
```

1.4 Gestion

```
mkdir <dossier>          # Créer le dossier
rmdir <dossier>          # Supprimer le dossier vide
touch <fichier>          # Créer le fichier
rm <fichier>             # Supprimer le fichier
cp <fichier> <dossier>   # Copier le fichier dans le dossier
mv <fichier> <dossier>   # Déplacer le fichier dans le dossier
ln <fichier1> <fichier2> # Créer un raccourci fichier1 vers le fichier2
```

2 TP2 : Commandes utilisateur Unix

2.1 Droits des fichiers

```
chmod <droit> <fichier>  # Changer les droits du fichier
```

Unix attribue à tous les fichiers deux choses :

- Un créateur et un groupe.
- Une liste de droits pour le créateur, le groupe et pour tous les autres (les triplets)

Exemple de droit pour le dossier "Projects" visible avec la commande "ls -l" :

```
drwxr-xr-x 17 justalternate root          4096 Aug  2 18:37 Projects
```

Les 10 premiers caractères représente :

- Le type de fichier (d = directory, - = regular file, l = symbolic link, p = pipe, s = socket ..)
- Les 3 premiers droits créateur (rwx = tous les droits)
- 3 droits du groupe (r-x)
- 3 droits pour tous les autres (r-x)
- le créateur (justalternate)
- le groupe associé (root)

Afin de modifier les droits d'un fichier, on peut d'abord agir sur les droits créateur, groupe et autre :

```
chmod +x fichier  # Donne à tous les utilisateurs la permission d'exécution
chmod u+r fichier  # Donne au propriétaire la permission de lecture
chmod g+w fichier  # Donne au groupe la permission d'écriture
chmod o+x fichier  # Donne au autre la permission d'exécution
chmod a-r fichier  # Enlève à tous (u,g,o) les permissions de lecture
chmod u+s fichier  # Donne les mêmes droits que le propriétaire à l'utilisateur
chmod o+t fichier  # Seul le propriétaire a la permission d'exécution
```

On peut ensuite modifier le groupe ou bien le créateur :

```
chgrp IDIA2026 Systeme_Info # Change le groupe du fichier "Systeme_Info" en "IDIA2026"
chown IDIA2026 Systeme_Info # Change le createur du fichier "Systeme_Info" en "IDIA2026"
```

2.2 Système

- /etc/fstab : liste les montages disponibles
- /etc/mtab : liste les points actuellement montés

```
df <fichier>  # Occupation disque du fichier
mount         # Monte un systeme de fichiers dans un répertoire de l'arborescence
```

2.3 Redirection

Dans Unix, on peut rediriger la sortie d'une commande dans un fichier ou bien utiliser un fichier en tant qu'arguments pour une commande.

```
ls > listefichiers.txt
```

Cette commande crée (ou écrase) le fichier "listefichiers.txt" avec le résultat de la commande "ls".
D'autres types de redirection :

- `>> #` Permet d'ajouter à la fin (append)
- `< #` Permet d'utiliser le contenu d'un fichier pour exécuter la commande.
- `2 > &1 #` Permet d'ajouter les potentielles erreurs de la commande dans le fichier

Le pipe

```
<commande1> | <commande2>
```

la sortie de la première commande devient l'entrée de la deuxième commande

2.4 Recherche

```
find <expr>           # Rechercher un fichier
grep <expr> <fichier> # Rechercher une expression dans le fichier
```

3 TP3 : Commandes utiles

3.1 Traitement des fichiers

```
head <fichier>      # Premières lignes du fichier
tail <fichier>      # Dernières lignes du fichier
split <fichier>     # Fractionnement du fichier en plusieurs
cut <fichier>       # Fractionnement vertical du fichier
```

sed Permet de remplacer des occurrences de mots dans un fichier.

```
sed 's/Hello/Bonjour/g' fichier.txt #-> remplace 'Hello' par 'Bonjour' dans le fichier.txt
```

tr Permet de remplacer à petite échelle.

```
echo toto | tr o a      #-> tata
echo hello | tr heo abc #-> abllc
```

3.2 Compression et archivage

```
gzip <fichier>      # Compression du fichier
gunzip <fichier>    # Decompression du fichier
zip <fichier>       # Compression et archivage du fichier
unzip <fichier>     # Extraction du contenu
tar <fichiers>      # Archivage des fichiers
```

diff Permet de comparer deux fichiers lignes par lignes et d'afficher les lignes différentes.

```
diff fichier1 fichier2
```

uniq Permet de ne pas tenir compte des répétitions.

```
uniq fichier
```

comm Permet de comparer deux fichiers triés ligne par ligne et d'afficher les lignes communes.

```
comm fichier1 fichier2
```

4 TP4 : Le Shell

Tous les scripts bash ont une extension .sh pour les exécuter dans l'invite de commande ont fait ./mon_script.sh attention de bien avoir la permission d'exécuter le fichier...

Pour passer des arguments a notre script ont fait :

```
./mon_script arg1 arg2 arg3
```

Et pour retrouver ces arguments ont fait :

```
$1 # arg1
$2 # arg2

## # le nombre d'arguments
$@ # tous les arguments sous la forme de liste
$* # tous les arguments sous la forme d'une chaîne de caractere

$0 # le nom du script (avec le ./)
$? # 1 si la commande précédente a fait une erreur sinon 0
```

Pour créer une variable ont fait :

```
ma_var=1 # Attention à bien coller 'ma_var' au '=' sinon ça ne marche pas.
mon_entier=-4
mon_string="chaîne de caractère"
```

Pour utiliser une variable déjà définie, on mettra systématiquement un \$ ou bien, on encapsulera la variable demandée par des ``

```
echo $ma_var #-> affiche 1
echo `ma_var` #-> affiche 1
var2=$ma_var #-> copie ma_var dans var2
```

Afin de former une expression avec nos variables, on utilise le mot 'expr' :

```
echo $(expr $ma_var + 1) #-> affiche 2
ma_var=$(expr $ma_var + 1) #-> incrémente la variable "ma_var"
```

ATTENTION le \$ derrière la parenthèse du "expr" est indispensable !

4.1 Les Tests

Syntax pour les comparaisons :

```
if [ <condition> ]
then
    <commande>
elif [ <condition> ]
then
    <commande>
else
    <commande>
fi #obligatoire pour terminer un if
```

Les options de comparaisons :

Tests sur les fichiers	
Options	Signification
if [-e \$fich]	fich existe
if [-f \$fich]	fich est un fichier
if [-d \$fich]	fich est un dossier
if [-r \$fich]	on a les droits lecture sur fich
if [-w \$fich]	on a les droits ecriture sur fich
if [-x \$fich]	on a les droits execution sur fich

Tests sur les Entiers	
Options	Signification
if [\$n1 -eq \$n2]	$n1 == n2$
if [\$n1 -ne \$n2]	$n1 \neq n2$
if [\$n1 -lt \$n2]	$n1 < n2$
if [\$n1 -gt \$n2]	$n1 > n2$
if [\$n1 -le \$n2]	$n1 \leq n2$
if [\$n1 -ge \$n2]	$n1 \geq n2$

Tests sur les chaines de caractères	
Options	Signification
if [-z \$string]	string est vide
if [\$string1 = \$string2]	string1 est identique a string2
if [\$string1 != \$string2]	string n'est pas identique a string2

4.2 Les Boucles

Pour faire des boucles dans bash, on peut faire comme dans python, soit un while, soit un for :

Les boucles while :

```
i=0
while [ $i -le 5 ] #On peut evidemment mettre n'importe quelle condition booléene entre les []
do
    echo "$i"
    i=$((expr $i + 1))
done
```

Les boucles for :

```
for i in {0..5}
do
    echo "$i"
done
```

Pour, c'est deux exécutions de code, on obtiendra donc :

```
0
1
2
3
4
5
```

ATTENTION DANS L'EXEMPLE DE LA BOUCLE FOR AU-DESSUS ON NE PEUT PAS REMPLACER 5 PAR UNE VARIABLE POUR REMPLACER PAR UNE VARIABLE ON FERRA CECI :

```
x=5
for i in $(seq $x) #Créer une séquence de 1 a 5
do
    echo "$i"
done
```

Sinon, on ferra un while. Les boucles for, mais en mieux

Avec la version 4.0 de Bash, on peut faire des trucs plus propre genre :

```
#!/bin/bash
for i in {0..10..2} #{départ..arrivé..incrémentation}
do
    echo "$i"
done
```

4.3 Parcourir les arguments

```
for i in "$@" #Permet de parcourir les arguments en entrées.
do
    echo "$i"
done
```

./parcours_arguments.sh Salut toi, tu es très beau

```
Salut
toi
tu
es
tres
beau
```

4.4 Lire un fichier

On peut aussi lire toutes les lignes d'un texte :

```
while read ligne
do
    echo "$ligne"
done < "Harry_Potter.txt"
```

À faire : Ajouter une manière alternative

Harry Potter and the Sorcerer 's Stone

CHAPTER ONE

THE BOY WHO LIVED

Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense.

5 TP5 : Les processus

5.1 Les signaux

```
ps # Affiche le pid (et autres infos) des processus courant
kill -sig pid # Envoie un signal à un processus
```

SIGINT (2) Interruption du processus SIGQUIT (3) Abandon du processus SIGKILL (9) Termine le processus immédiatement
SIGTERM (15) Termine le processus SIGCONT (18) Continue le processus SIGSTOP (19) Suspension du processus