

Objectif À l'issue de ce TP, vous serez capable de : naviguer dans l'arborescence Linux en ligne de commande, gérer fichiers et répertoires (copie, déplacement, suppression, liens), rechercher du contenu avec grep, comprendre et modifier les droits d'accès, créer une archive compressée, gérer un processus en tâche de fond, et écrire un script shell simple et interactif.

Durée indicative : 2h30 à 3h • **Mode de travail** : individuel, sur machine virtuelle • **Rendu attendu** : voir section « Travail attendu » en fin de TP.

Prise en main de la VM

Importez la machine virtuelle fournie et démarrez-la.

Identifiant (login) : isty
Mot de passe : isty

Attention Cette VM est votre bac à sable pour tout le TP : n'hésitez pas à vous tromper, à tout supprimer et à recommencer. En revanche, ne lancez la commande finale sudo shutdown qu'au tout dernier moment, une fois votre travail sauvegardé et montré à l'enseignant.

Environnement de travail

Ouvrez une session sur votre poste de travail : en mode console ou en interface graphique, vous travaillerez « en ligne de commande ». Sous Linux, le raccourci Ctrl + Alt + T ouvre un terminal.

Créez d'abord votre répertoire de travail tp dans \$HOME/tp :

```
$ mkdir -p $HOME/tp  
$ cd $HOME/tp
```

Point de contrôle Votre invite de commande doit maintenant afficher un chemin se terminant par /tp — vérifiez-le avec la commande pwd (print working directory) avant de continuer.

La complétion automatique (touche Tab)

À chaque fois que vous tapez une commande, la touche Tab déclenche la complétion automatique. Bash essaie successivement d'interpréter le texte tapé comme une variable (commence par \$), un nom d'utilisateur (~), un nom d'hôte (@), ou une commande — et à défaut, comme un nom de fichier.

\$ c <Tab><Tab> → « Display all 142 possibilities? (y or n) » : de nombreuses commandes commencent par c.

\$ cd <Tab><Tab> → propose cd, cd-create-profile, cd-fix-profile, cd-iccdump, cd-it8...

\$ cd /h <Tab> → complète automatiquement en cd /home/

Astuce Prenez le réflexe d'appuyer sur Tab dès les 2-3 premiers caractères d'une commande ou d'un chemin : c'est le geste qui vous fera gagner le plus de temps ce semestre.

Note: **Bash** est un [shell](#), soit un interpréteur de ligne de commande : il permet de "discuter" avec l'ordinateur, en [ligne de commande](#). Sur Ubuntu c'est le *shell* utilisé par défaut dans le [terminal](#). Il est bien évidemment libre, sous licence .

La plupart des scripts que vous verrez sur la documentation comporte un [shebang](#) (représenté par #!), par exemple :#!/bin/bash, votre système utilisera donc **Bash** pour interpréter les commandes.

Remarque — l'invite `isty@isty-virtualbox` : la VM s'appelle `isty-virtualbox`, l'utilisateur courant s'appelle `isty`. Au-dessus de cet utilisateur se trouve le `root` (administrateur). La commande `su` permet de devenir un autre utilisateur (le compte `root` n'a volontairement pas été créé ici) ; la commande `sudo` permet à `isty` de s'accorder ponctuellement les droits `root` pour une seule commande.

```
$ sudo apt update
```

La liste des commandes de base est disponible dans le support de cours associé.

Étape 1 — Démonstration guidée : arborescence de fichiers

Reproduisez chacune des commandes suivantes et observez le résultat avant de passer à la suivante.

Se déplacer et créer un répertoire

```
$ cd $HOME/tp
$ mkdir temp
$ cd temp
```

Copier, lister, sauvegarder, renommer

\$ `cp /etc/passwd .` copie le fichier `passwd` dans le répertoire courant (désigné par un point).

\$ `ls` liste le contenu du répertoire courant.

Le fichier `passwd` contient la liste des utilisateurs de la machine ; le répertoire `/etc` regroupe l'ensemble des fichiers de configuration système (tous en texte ASCII).

\$ `cp ./passwd ../passwd.bak` crée une copie de sauvegarde dans le répertoire parent.

\$ `mv ./passwd ./listeUtilisateurs.txt` renomme le fichier.

Visualiser un fichier texte

```
$ more listeUtilisateurs.txt
$ cat listeUtilisateurs.txt
$ less listeUtilisateurs.txt
```

Astuce `more`, `cat` et `less` affichent tous un fichier texte, mais pas de la même façon : `cat` déverse tout d'un coup, `more/less` permettent de naviguer page par page. Essayez `less` puis quittez avec la touche `q` — c'est l'outil que vous utiliserez le plus souvent pour lire de longs fichiers de logs.

Rechercher un fichier

```
$ find $HOME -name listeUtilisateurs.txt -print
$ find $HOME -name *.txt -print
```

L'étoile `*` est un caractère joker (wildcard) qui remplace n'importe quelle suite de caractères.

Effacer, déplacer un fichier ou un répertoire

\$ `rm listeUtilisateurs.txt` supprime le fichier (l'option `-f` force la suppression sans confirmation — définitive !).

\$ `mv $HOME/tp $HOME` déplace le répertoire `tp`.

\$ `mv $HOME/tp/passwd.bak $HOME` déplace le fichier de sauvegarde.

\$ `rm -rf $HOME/tp` supprime définitivement le répertoire et tout son contenu (`-r` = récursif).

\$ `rm $HOME/passwd.bak` supprime le dernier fichier restant.

Point de contrôle Tapez `ls -la $HOME` : il ne doit plus rester aucune trace (fichier ou répertoire) de cette manipulation sur votre système. Si un fichier subsiste, identifiez lequel avant de continuer.

Exercice 1 — À vous de jouer

Attention Pour chaque commande, notez dans votre compte-rendu : la commande exacte que vous avez tapée, le résultat obtenu (ou une capture d'écran), et une phrase expliquant ce qu'elle fait. C'est cette explication qui est évaluée, pas seulement le fait d'avoir tapé la commande.

1. Gestion de fichiers et arborescence

1. Se placer dans son répertoire personnel.

```
$ cd $HOME
```

2. Créer et entrer dans le répertoire `exercice_linux`.

```
$ mkdir exercice_linux
$ cd exercice_linux
```

3. Copier la liste des utilisateurs dans un fichier `utilisateurs.txt`.

```
$ cp /etc/passwd utilisateurs.txt
```

4. Afficher le nombre de lignes du fichier.

```
$ wc -l utilisateurs.txt
```

5. Afficher les 5 premières lignes.

```
$ head -n 5 utilisateurs.txt
```

Astuce `head -n 5` affiche les 5 premières lignes ; `tail -n 5` affiche les 5 dernières. Essayez `tail -n 5 utilisateurs.txt` et comparez.

2. Recherche dans un fichier : `grep`

6. Lignes contenant « `bash` ».

```
$ grep "bash" utilisateurs.txt
```

7. Lignes contenant « `nologin` » (sans sensibilité à la casse).

```
$ grep -i "nologin" utilisateurs.txt
```

8. Afficher uniquement le nom des utilisateurs ayant `/bin/bash` comme shell.

```
$ grep "/bin/bash" utilisateurs.txt | cut -d: -f1
```

Astuce Le symbole `|` (pipe) enchaîne deux commandes : la sortie de `grep` devient l'entrée de `cut`. C'est l'un des concepts les plus puissants d'Unix — vous le réutiliserez toute l'année.

Question de compréhension : à quoi correspond l'option `-d` de la commande `cut`, et pourquoi `-f1` ?

3. Droits et permissions

Chaque fichier ou répertoire possède un propriétaire, un groupe, et trois catégories de droits (lecture `r`, écriture `w`, exécution `x`) pour trois catégories d'utilisateurs : le propriétaire (`u`), le groupe (`g`), les autres (`o`, pour « others »).

Position	-	rwX	r - x	r - x
Signification	type (- fichier, d répertoire, l lien)	propriétaire (u)	groupe (g)	autres (o)

Exemple : `-rwxr-xr-x` correspond à un fichier dont le propriétaire peut tout faire (lire, écrire, exécuter), tandis que le groupe et les autres ne peuvent que lire et exécuter.

9. Créer un fichier vide secret.txt.

```
$ touch secret.txt
```

10. Vérifier ses droits.

```
$ ls -l secret.txt
```

11. Retirer tous les droits aux autres utilisateurs.

```
$ chmod o-rwx secret.txt
```

12. Donner au seul propriétaire le droit d'écriture (en plus de ses droits actuels).

```
$ chmod u+w secret.txt
```

Point de contrôle Que constatez-vous en comparant les deux résultats de `ls -l secret.txt` avant et après ces deux commandes `chmod` ? Identifiez précisément quel(s) caractère(s) ont changé dans la chaîne de droits (ex. `rw-r--r--`).

Aller plus loin : la notation octale

La notation symbolique (u/g/o, +/-, rwx) a un équivalent en notation octale, plus rapide à écrire une fois maîtrisée : chaque droit vaut une puissance de 2, et on additionne les valeurs pour chaque catégorie.

Droit	r (lecture)	w (écriture)	x (exécution)
Valeur	4	2	1

Ainsi, `rw-r--r--` s'écrit 755 (7 = 4+2+1 pour le propriétaire, 5 = 4+1 pour le groupe, 5 = 4+1 pour les autres), et `rw-r--r--` s'écrit 644.

13. Afficher les droits de secret.txt en notation octale.

```
$ stat -c "%a %n" secret.txt
```

14. Fixer les droits de secret.txt à `rw-r--r--` (644) en une seule commande, en notation octale.

```
$ chmod 644 secret.txt  
$ ls -l secret.txt
```

Point de contrôle Vérifiez que la chaîne `rwx` obtenue avec `ls -l` correspond bien à 644. Convertissez maintenant vous-même 750 et 600 en notation `rwx` sur le papier avant de tester avec `chmod`.

Droits par défaut : le umask

Quand vous créez un nouveau fichier, ses droits ne sont pas 777 (tout autorisé) mais filtrés par une valeur appelée `umask`, propre à votre session.

15. Afficher votre `umask` courant.

```
$ umask
```

16. Créer un nouveau fichier et observer ses droits par défaut.

```
$ touch nouveau.txt  
$ ls -l nouveau.txt
```

Question de compréhension : un fichier nouvellement créé n'a jamais le droit `x` par défaut, même avec un `umask` à 000 — pourquoi, à votre avis, cette règle existe-t-elle pour la sécurité du système ?

Changer le propriétaire ou le groupe

chmod modifie les droits, mais pas le propriétaire ni le groupe d'un fichier : c'est le rôle de chown et chgrp. Changer le propriétaire d'un fichier vers un autre utilisateur nécessite les droits root (sudo) ; changer le groupe est possible si vous appartenez au groupe cible.

```
$ groups
$ chgrp <un_groupe_auquel_vous_appartenez> secret.txt
$ sudo chown isty:isty secret.txt
```

Astuce N'exécutez jamais chmod -R 777 sur un répertoire « pour que ça marche » : cette commande rend tous les fichiers et sous-répertoires accessibles en lecture/écriture/exécution à absolument tout le monde, y compris les scripts qu'elle contient. C'est l'une des erreurs de configuration les plus fréquentes en entreprise.

Un exemple concret : le bit setuid

Certaines commandes système ont besoin de droits root le temps de leur exécution, même lancées par un utilisateur normal — c'est le cas de passwd, qui doit écrire dans un fichier système protégé.

```
$ ls -l /usr/bin/passwd
```

Point de contrôle Vous devez observer un s à la place du x dans les droits du propriétaire (par exemple rwsr-xr-x). C'est le bit setuid : il fait que la commande s'exécute avec les droits de son propriétaire (root) plutôt qu'avec les vôtres. C'est ainsi qu'un utilisateur normal peut changer son propre mot de passe sans avoir les droits root en permanence.

4. Liens symboliques et liens physiques

Un lien permet de faire pointer plusieurs noms de fichiers vers un même contenu, sans le dupliquer. C'est très utile pour éviter de recopier un gros fichier, pour créer un « raccourci » vers un chemin profond dans l'arborescence, ou pour gérer plusieurs versions d'un même programme.

Exemple concret : sur la plupart des distributions Linux, la commande python3 n'est pas un vrai binaire mais un lien symbolique vers une version précise, par exemple /usr/bin/python3 -> python3.11. Le jour où le système passe à Python 3.12, il suffit de refaire pointer ce lien vers le nouveau binaire : tous les scripts qui appellent python3 utilisent alors automatiquement la nouvelle version, sans qu'un seul fichier n'ait été déplacé ni recopié. Vous pouvez vérifier ce lien sur votre VM :

```
$ ls -l /usr/bin/python3
```

Il existe deux façons de créer ce genre de raccourci, avec des comportements assez différents :

	Lien physique (hard link)	Lien symbolique (soft link)
Commande	ln cible lien	ln -s cible lien
Ce qu'il pointe	directement les données sur le disque (même inode)	le nom du chemin de la cible (comme un pense-bête)
Numéro d'inode	identique à celui de la cible	différent (c'est un fichier à part entière)
Si la cible est supprimée	les données restent accessibles via le lien	le lien devient « cassé » (rouge, illisible)
Traverse les partitions/disques ?	non, même système de fichiers uniquement	oui, aucune contrainte
Peut cibler un répertoire ?	non (sauf cas très particuliers)	oui

Astuce Retenez l'image suivante : un lien physique est une deuxième étiquette collée sur le même carton (le carton n'existe qu'une fois, mais a deux noms) ; un lien symbolique est un post-it sur lequel est écrite l'adresse du carton (si le carton disparaît, le post-it ne pointe plus vers rien).

17. Créer un lien physique (hard link).

```
$ ln utilisateurs.txt lien_physique.txt
```

18. Créer un lien symbolique (soft link).

```
$ ln -s utilisateurs.txt lien_symbolique.txt
```

19. Comparer les numéros d'inode des fichiers.

```
$ ls -li
```

Point de contrôle Quels fichiers partagent le même numéro d'inode ? Que se passe-t-il si vous supprimez utilisateurs.txt : lien_physique.txt et lien_symbolique.txt restent-ils lisibles tous les deux ? Testez-le, puis restaurez le fichier si besoin.

5. Archivage et compression

20. Revenir au répertoire personnel.

```
$ cd $HOME
```

21. Créer une archive du répertoire exercice_linux.

```
$ tar -cvf sauvegarde.tar exercice_linux/
```

22. Compresser l'archive.

```
$ gzip sauvegarde.tar
```

Astuce Pour vérifier le contenu d'une archive sans la décompresser : tar -tzvf sauvegarde.tar.gz. C'est un réflexe utile avant de la transférer ou de la restaurer sur une autre machine.

6. Processus

23. Lancer une commande longue en arrière-plan.

```
$ sleep 200 &
```

24. Identifier son PID (identifiant de processus).

```
$ ps aux | grep sleep
```

25. Terminer le processus.

```
$ kill <PID>
```

Question de compréhension : quelle est la différence entre kill <PID> et kill -9 <PID> ? Dans quel cas utiliseriez-vous le second ?

7. Script shell

26. Créer un fichier de script.

```
$ nano bonjour.sh
```

27. Contenu du script :

```
#!/bin/bash
echo "Bonjour $USER, nous sommes le $(date +%d/%m/%Y) il est $(date +%H:%M)."

```

28. Rendre le script exécutable.

```
$ chmod +x bonjour.sh
```

29. Lancer le script.

```
$ ./bonjour.sh
```

Astuce Si ./bonjour.sh échoue avec « Permission denied », vérifiez avec `ls -l` que le x est bien présent dans les droits. S'il échoue avec « bad interpreter », vérifiez qu'il n'y a pas de caractères invisibles Windows (CRLF) dans le fichier — utilisez `dos2unix bonjour.sh` pour corriger.

Exercice 2 — Script interactif : évaluation d'une note

Écrivez un script `note.sh` qui demande à l'utilisateur de saisir une note (sur 20) et affiche un message selon le barème suivant :

Note	Appréciation
16 à 20	très bien
14 à 16	bien
12 à 14	assez bien
10 à 12	moyen
< 10	insuffisant

- Utilisez `read` pour lire la saisie de l'utilisateur.
- Utilisez des structures `if / elif / else` pour tester la note.
- Ajoutez un émoticône ou une petite « animation » texte (par exemple avec `sleep` entre plusieurs `echo`) pour rendre le résultat plus vivant.

Astuce Pensez à valider la saisie : que fait votre script si l'utilisateur tape une lettre au lieu d'un nombre, ou une note négative ? Un script robuste vérifie toujours ses entrées.

Pour aller plus loin (bonus)

Proposez votre propre mini-exercice Linux (l'énoncé et sa correction), sur un thème de votre choix parmi : gestion des utilisateurs et groupes, tâches planifiées (cron), variables d'environnement, ou redirections (`>`, `>>`, `2>`). Ce bonus peut rapporter des points supplémentaires.

Travail attendu

À la fin de la séance, votre répertoire personnel doit contenir :

- l'archive compressée `sauvegarde.tar.gz` ;
- le script exécutable `bonjour.sh` ;
- le script exécutable `note.sh` (exercice 2) ;

- un compte-rendu (texte ou capture d'écran) expliquant, pour chaque question de l'exercice 1, la commande utilisée et ce qu'elle fait — vous devez être capable de justifier à l'oral les notions vues : liens, grep, chmod, processus, script.

Point de contrôle Avant de conclure la séance, relisez votre compte-rendu et vérifiez que chaque commande est bien expliquée avec vos propres mots, pas seulement recopiée depuis l'énoncé.

Attention Ce n'est qu'une fois votre travail sauvegardé et montré à l'enseignant que vous exécutez la commande suivante pour éteindre la VM (adaptez l'heure à votre contexte, par exemple 13:01) :

```
$ sudo shutdown 13:01
```

Ce qui est utile pour les projets à l'ISTY

Lister les périphériques USB

- `lsusb` — liste tous les périphériques USB (fabricant, ID produit)
- `lsusb -t` — vue arborescente (bus, hubs, vitesse de chaque port)
- `lsusb -v` — détails complets (verbeux, à filtrer avec `grep`)

Voir ce qui apparaît dans `/dev`

- `ls /dev/tty*` — ports série (Arduino, STM32, adaptateurs USB-série type FTDI/CH340)
- `ls /dev/sd*` — disques et clés USB en mode bloc
- `ls -la /dev/serial/by-id/` — très utile : liste les périphériques série par identifiant stable (nom du modèle), plutôt que `ttyUSB0` qui change selon l'ordre de branchement

et le fichier `.bashrc` ?

Le fichier `.bashrc` est un script shell exécuté automatiquement à chaque ouverture d'un terminal interactif non-login sous Bash. Il vit dans le répertoire personnel de l'utilisateur : `~/ .bashrc` (donc `$HOME/ .bashrc`).

À quoi il sert ?

Il personnalise l'environnement de travail en ligne de commande : alias, variables d'environnement, apparence de l'invite, fonctions utiles, etc. Tout ce que vous y mettez est rejoué à chaque nouveau terminal.

Quand il est exécuté

C'est le point le plus souvent mal compris :

- **Shell interactif non-login** (ex. vous ouvrez un nouvel onglet de terminal sur votre bureau) → `.bashrc` est lu.
- **Shell de login** (ex. connexion SSH, ou connexion en console texte) → c'est `.bash_profile` (ou `.profile`, ou `.bash_login`) qui est lu en premier, pas `.bashrc` directement. En pratique, la plupart des distributions font en sorte que `.bash_profile` appelle lui-même `.bashrc`, pour unifier le comportement.
- **Script non interactif** (ex. `bash monscript.sh`) → `.bashrc` n'est généralement pas chargé du tout, sauf si le script le source explicitement.

Ce qu'on y trouve typiquement

```
bash

# Alias pratiques
alias ll='ls -la'
alias grep='grep --color=auto'

# Variables d'environnement
export PATH="$HOME/bin:$PATH"
export EDITOR=nano

# Personnalisation de l'invite (prompt)
PS1='\u@\h:\w\$ '

# Historique des commandes
HISTSIZE=2000
HISTFILESIZE=2000

# Fonctions personnalisées
extract() {
    tar -xzf "$1"
}
```

Comment l'utiliser en pratique

\$ nano ~/.bashrc ou \$ micro ~/.bashrc (nano et micro sont 2 éditeurs de texte)

Éditez, sauvegardez, puis rechargez sans fermer le terminal :

\$ source ~/.bashrc

ou son raccourci équivalent :

\$. ~/.bashrc

Un point utile pour les TP : quand nous ajoutons un alias ou une variable et qu'« elle ne marche pas », c'est très souvent parce qu'il a oublié de faire **source ~/.bashrc** (ou de rouvrir un nouveau terminal) — le fichier n'est lu qu'au démarrage du shell, pas en continu.

Différence avec /etc/bash.bashrc : **.bashrc** est propre à un utilisateur ; **/etc/bash.bashrc** (ou **/etc/bashrc** selon la distribution) est la version globale, appliquée à tous les utilisateurs du système, et généralement lue avant le fichier personnel

Ce TP est mon TP, écrit par moi-même, revu et corrigé par une IA.