NANO TUTO UNIX & PYTHON SOUS UNIX

SYSTÈMES D'EXPLOITATION DE TYPE UNIX

Les TP se font dans un environnement UNIX qui est un paradigme multi-utilisateur et multitâche à la base de systèmes d'exploitation comme **Linux**, Android ou MacOS. Il repose sur 3 principes :

1. **Tout est fichier** : documents, répertoires, processus, etc. tout est codé par des fichiers.
2. **Hiérarchie des fichiers** : les fichiers sont organisés selon une arborescence **unique**.
3. **Commandes** : des **processus** exécutent des tâches spécifiques et peuvent être combinés.

Un utilisateur peut se connecter sur un tel système grâce à son identifiant (**login**) et son mot de passe (**passwd**). Il dispose d'un dossier personnel (**répertoire de travail)** pour y ranger ses **fichiers**.

Arborescence des fichiers

L'organisation des fichiers suit une structure arborescente avec une entrée unique (**la racine**). Voici les principaux répertoires, leur rôle et ce qu'ils contiennent :

- `/` : racine de l'arborescence, contient *tous* les répertoires.
- `/usr/bin` : contient les commandes (programmes) utilisables par tous, comme `ls` ou `cp`.
- `/dev` : contient les périphériques (clefs usb, disques, etc).
- `/usr/sbin` : contient les programmes pour l'administration du système.
- `/etc` : contient les fichiers de configuration du système (ex : fichier `passwd` des utilisateurs).
- `/home` : contient les répertoires personnels des utilisateurs où ils stockent leurs fichiers.
- `/var` : contient les fichiers variables, les journaux système (exemple : `/var/log`)
- `/usr` : contient les bibliothèques et les logiciels pour les utilisateurs.
- `/tmp` : contient les fichiers temporaires.

Un **chemin** est une chaîne de caractères qui code l'emplacement d'un fichier dans l'arborescence, le symbole « slash » `/` y indique une bifurcation. Le chemin `/home/etudiants/alanturing/machine.py` mène vers un script *Python* nommé `machine.py` rangé dans le répertoire personnel de l'étudiant dont le login est `alanturing` dans le répertoire `etudiants` dans le répertoire `home`. On se déplace dans l'arborescence grâce à la commande `cd` suivie du chemin vers le répertoire destination. Exemple : `cd /home/etudiants` déplace l'utilisateur dans le répertoire des étudiants.

Si un chemin débute par `/`, il s'agit d'un **chemin absolu** qui part nécessairement de la racine, sinon c'est un **chemin relatif** qui débute à la position de l'utilisateur. Le répertoire où l'on se trouve est codé par un simple point `.`, le répertoire « au-dessus » par un double point `..` et le répertoire de travail par le tilde `~`. On "remonte" donc dans le répertoire supérieur de l'arborescence avec la commande `cd ..` et dans son répertoire de travail avec `cd ~` (ou `cd` sans aucun paramètre).

Gestion des droits et des utilisateurs

Les **permissions** accordées à chaque fichier sont codées par une chaîne de 10 = 1 + 3 + 3 + 3 caractères visibles via la commande `ls -l nomdufichier`. Le 1er indique la nature du fichier (lien, fichier simple, répertoire, etc.), les 3 blocs de 3 caractères suivant codent les permissions du **propriétaire** du fichier, des utilisateurs du **groupe** auquel il appartient et des **autres** utilisateurs :

`r` pour la lecture, `w` pour l'écriture et `x` pour l'exécution.

Exemple : `-rwxr-xr--` indique qu'il s'agit d'un simple fichier (symbole -, ce serait le symbole `d` pour un répertoire), que son propriétaire peut le lire, le modifier et l'exécuter (`rwx`), que les utilisateurs du même groupe peuvent le lire, ne peuvent pas y écrire mais peuvent l'exécuter (`r-x`), et que les autres utilisateurs peuvent le lire mais ni y écrire ni l'exécuter (`r--`).

Commandes UNIX (sans les détails)

On saisit des **commandes** UNIX dans un **terminal**. Elles sont exécutées par un programme appelé **interprète de commandes** ou **shell** et qui doivent respecter la syntaxe du langage shell en question. Les différents shell sont de véritables langages de programmation spécialisés pour ce système.

Pour ouvrir un **terminal**, on appuie *simultanément* sur les trois touches `alt`, `ctrl` et `t`, ou on clique sur le bouton droit de la souris faisant apparaître un menu qui propose de le faire. L'**invite de commande** est un symbole dans le terminal après lequel il faut taper sa commande puis la valider avec la touche `Entrée`. Il s'agit généralement du symbole `$`, précédé ou non selon la configuration, d'informations sur sa position dans l'arborescence de la machine, entre autres. Une commande peut attendre zéro, un ou plusieurs paramètres, séparés par des espaces. Exemple `ls -l nomdufichier` fait apparaître les 2 paramètres `-l` et `nomdufichier` à la commande ls.

NB. L'invite de commande du *Python*, qui est lui aussi un langage de commande est `>>>`.

Deux types de commandes sont possibles, **internes** et **externes**. Les commandes internes sont incluses dans le shell et peuvent être exécutées directement alors que les commandes externes, comme l'interprète *Python* par exemple, sont rangées dans l'arborescence UNIX. Il faut en général indiquer au shell leur chemin pour les exécuter.

NB. Le shell configuré par défaut sur les machines de TP est le **bash** (*bourne again shell*).

!!	Rappelle la dernière commande sur le terminal
↑/↓	Monte/Descend dans l'historique des commandes exécutées dans la session
↹	Complète le nom du fichier en cours de saisie dans le terminal
<ctrl z> + bg	Interrompt le programme actif lancé dans le terminal puis le détache
<ctrl d>	Ferme le terminal ou envoie le caractère <EOF> de fin de fichier.
<ctrl c>	Tue le processus en cours d'exécution dans le terminal
cat / more / less	Affiche le contenu du fichier en paramètre
cd	Déplacement à la position en paramètre (`..` pour remonter)
chmod	Change le mode de lecture/écriture/exécution d'un fichier en paramètre
chown	Change le propriétaire d'un fichier en paramètre
clear	Efface le contenu du terminal
cp / ln	Copie/Crée un lien d'un fichier en paramètre
df / du	Affiche l'utilisation de l'espace disque / du fichier en paramètre
diff	Affiche les différences de contenus entre deux fichiers en paramètres
echo	Affiche la chaîne de caractère en paramètre sur la sortie standard
find	Cherche et/ou agit sur un fichier/répertoire en paramètre
grep	Recherche les lignes de fichiers satisfaisant des contraintes en paramètres
head / tail	Affiche la tête / la queue d'un fichier
history	Affiche les commande précédentes
info / man	Affiche une aide sur une commande
jobs / ps / top	Affiche les processus en cours d'exécution
kill / xkill	Tue un processus (en cliquant sur sa fenêtre pour xkill)
locate	Recherche rapide de la position du fichier en paramètre
ls	Liste le contenu du dossier actif ou celui dont le nom est en paramètre
make	Utilitaire de gestion de programmes. Configuration dans un fichier Makefile
mkdir / rmdir	Crée/Détruit le dossier dont le nom est en paramètre
mv	Déplace un fichier/dossier en paramètre vers une destination en paramètre
rm	Élimine les fichiers/dossiers en paramètre
sed	Modifie des chaînes de caractères d'un fichier en paramètre
sort	Trie le contenu du fichier en paramètre
ssh / sftp	Connexion/Transfert de fichier vers la machine distante en paramètre
tar	Commande de création d'archives des fichiers en paramètre
uname	Affiche les informations détaillées du système

wc	Compte le nombre de caractères, mots, lignes du fichier en paramètre
which	Renvoie l'emplacement du programme lié à la commande en paramètre
who	Affiche les utilisateurs connectés sur la machine
whoami	Affiche le login de l'utilisateur sur la machine du terminal
zip / unzip	Compresse/Décompresse le/les fichiers en paramètre(s)

IDLE PYTHON, AVANTAGES/INCONVÉNIENTS

IDLE[1] *Python* (Integrated Development and Learning Environment) est un environnement de développement intégré pour le langage *Python*. Il anime plusieurs logiciels :

1. Un **éditeur** de texte. Modifie et affiche le contenu littéral d'un fichier.
2. Un **terminal** de commandes, c'est l'interface entre l'utilisateur et la machine.
3. Un **interprète** de commandes, c'est le programme qui interprète un script en langage *Python*.
4. Un **débogueur** qui permet de débusquer les erreurs dans le script.

Avantages : on retrouve le même environnement sur des architectures et des systèmes différents (Windows, UNIX, MacOS, etc.) et il regroupe les outils essentiels pour écrire des scripts.
Inconvénients : il masque les interactions entre les différents logiciels qu'il anime et qui doivent être connues par des étudiant.e.s en informatique. En cas de plantage à l'exécution, l'utilisateur peut perdre tout ou partie de son travail. C'est très fréquent.

NB. Il ne faut pas confondre un **éditeur de texte** avec un **traitement de texte.** La mise en page d'un document à l'aide d'un traitement de texte génère des commandes d'un langage de formatage qui sont conservées dans le document **en sus** du texte. Elles sont interprétées par le logiciel mais invisibles pour l'utilisateur, alors qu'un éditeur de texte manipule le contenu exact du fichier.

PYTHON DANS UN SYSTÈME D'EXPLOITATION DE TYPE UNIX

Paramétrage éventuel

Le programme de l'interprète *Python* version 3 est nommé `python3` et se trouve généralement dans le répertoire `/usr/bin`. Pour l'exécuter il faut indiquer son chemin complet, autrement dit il faut taper la commande `/usr/bin/python3`. Il est possible que l'administrateur du système ait créé un alias nommé `python` ou `python3` pour vous éviter de taper le chemin. Si ce n'est pas le cas, vous pouvez créer vous même cet alias grâce à la commande suivante sur le terminal

```
$ alias python='/usr/bin/python3'
```

Pour éviter de taper cette commande à chaque nouvelle session de TP, un fichier de configuration personnalisé du shell nommé `.bashrc` est rangé à la racine de votre répertoire de travail et il est exécuté (s'il existe) **à chaque ouverture** d'un terminal. Il permet, entre autres, de créer des alias, déclarer des variables locales, des variables d'environnement, etc. Il suffit d'éditer ce fichier en exécutant la commande suivante :

```
$ gedit ~/.bashrc
```

Cette commande lance l'éditeur de texte *gedit* qui ouvre le fichier `.bashrc` dans lequel il faut écrire la commande indiquée plus haut sur une ligne vierge et sauvegarder le fichier :

```
alias python = '/usr/bin/python3'
```

[1] en hommage à Eric Idle, membre des *Monty Python*.

Modes d'utilisation de *Python* :

1. `$ python` Dans ce cas, *bash* lance un interprète *python* qui prend la main sur le terminal, ce qui est matérialisé par l'invite de commande du Python, les trois chevrons `>>>`.
2. `$ python monscript.py` Dans ce cas, *bash* lance un interprète *Python* qui va exécuter séquentiellement les commandes du script intitulé `monscript.py` (rédigé au préalable avec un éditeur de texte comme *gedit* par exemple) puis rendre la main au shell une fois les instructions du script exécutées.

NB. On peut placer en tâche de fond un processus lancé sur un terminal pour que l'interprète shell redevienne actif avant la fin de l'exécution du processus. Il suffit de rajouter le symbole `&` à la fin de la commande. Exemple : `$ gedit monfichier.py &` lance l'éditeur *gedit* sur le fichier intitulé monfichier.py et le détache du terminal. Sans `&`, le terminal serait bloqué tant que l'exécution de *gedit* ne serait pas achevée.