TD1 – Langages de Script

Abdallah Ammar

13 janvier 2026

Le symbole `$` affiché au début des commandes correspond à l'invite du terminal. Il sert uniquement à indiquer que la commande doit être tapée, mais ne fait pas partie de la commande elle-même.

Il est fortement recommandé de taper les commandes à la main afin de se familiariser avec le terminal.

La commande `apt` est spécifique aux distributions Linux basées sur Debian (Ubuntu, WSL Ubuntu). Sur macOS, l'installation de logiciels se fait généralement avec `brew`. Dans ce TD, les commandes `apt` sont à exécuter uniquement sous Linux ou WSL.

Dans le terminal, il est possible de réutiliser et de modifier des commandes déjà tapées. Les flèches du clavier permettent de naviguer dans l'historique des commandes :

— flèche vers le haut : commande précédente ;
— flèche vers le bas : commande suivante.

Il est également possible d'utiliser la touche `TAB` pour compléter automatiquement les noms de commandes, de fichiers ou de répertoires. Cette fonctionnalité permet d'éviter les fautes de frappe et d'accélérer la saisie.

L'historique des commandes peut être affiché avec la commande suivante :

```
$ history
```

Les lignes commençant par le symbole `#` sont des commentaires. Elles servent à expliquer ou documenter des commandes, mais ne sont pas exécutées par le shell.

1 Premiers pas

1.1 Certaines commandes sont intégrées directement dans le shell (commandes internes ou *built-in*), tandis que d'autres sont des programmes externes.

La commande `type` permet de savoir de quel type est une commande. Par exemple :

```
$ type cd
```

Selon le système et la version du shell, certaines commandes peuvent exister à la fois comme commande interne et comme programme externe. Le résultat de `type` peut donc varier.

À l'aide de cette commande, déterminez si les commandes suivantes sont des commandes internes ou des commandes externes :

— `cd`
— `echo`
— `ls`
— `pwd`
— `type`
— `mkdir`
— `date`
— `sleep`
— `touch`
— `cat`
— `grep`

1.2 Savoir se documenter est une compétence essentielle sous Linux. La documentation permet de comprendre le fonctionnement d'une commande et de découvrir ses options.

La commande `man` fournit une page de manuel concise et standard pour les commandes externes, tandis que la commande `info` propose une documentation plus détaillée, organisée comme un ensemble de chapitres. La commande `help` affiche une aide intégrée au shell et s'applique principalement aux commandes internes.

1. Consultez la documentation de la commande `ls` avec `man`, puis avec `info`. Pour quitter ces interfaces, appuyez sur `q`.
2. Trouvez l'option permettant d'afficher les fichiers cachés, puis utilisez-la pour identifier les fichiers cachés présents dans votre répertoire personnel.
3. Expliquez pourquoi la commande `man cd` ne fonctionne pas. Consultez l'aide de la commande `cd`.

Commandes utiles

```
$ man ls
$ info ls
$ ls -a $HOME
$ ls -a ~  # commande équivalente à la commande précédente
$ help cd
```

1.3 Pour installer un nouveau programme sous Linux, il est recommandé de commencer par mettre à jour la liste des paquets disponibles avec `apt update`. Cela permet au système de connaître les versions les plus récentes des logiciels.

```
$ sudo apt update
```

La commande `sudo` permet d'exécuter une commande avec les droits d'administrateur. Elle est nécessaire pour installer des programmes ou modifier le système. Lors de son utilisation, un mot de passe peut être demandé.

> Utilisez `sudo` uniquement lorsque cela est nécessaire et vérifiez toujours la commande avant de l'exécuter.

Ensuite, on installe le programme souhaité à l'aide de la commande `apt install`.
Par exemple, installez puis lancez le programme `sl` :

```
$ # pour macOS :
$ brew install sl
$ # pour Linux :
$ sudo apt install sl
$ sl
```

Installez maintenant le programme `cowsay`, qui affiche un message sous forme de dessin ASCII :

```
$ # pour macOS :
$ brew install cowsay
$ # pour Linux :
$ sudo apt install cowsay
```

Puis lancez-le avec un message de votre choix :

```
$ cowsay "Bonjour Linux"
$ cowsay -f dragon "Je suis un dragon!"
$ cowsay -f tux "Je suis Linux!"
$ cowsay -f moose "Je suis un élan!"
$ cowsay -f koala "Je suis un koala!"
$ cowthink "Je pense donc je suis"
```

1.4 Dans cet exercice, vous allez créer une petite arborescence afin de comprendre l'effet des chemins relatifs et absolus. Après chaque changement de répertoire avec la commande `cd`, affichez systématiquement le répertoire courant à l'aide de la commande `pwd`.

> Il est fortement recommandé d'éviter les espaces dans les noms de fichiers et de répertoires sous Linux. Les espaces compliquent l'utilisation des commandes dans le terminal. Préférez l'utilisation du caractère `-` (tiret) ou `_` (underscore) pour séparer les mots.

Rappel :
— `.` désigne le répertoire courant ;
— `..` désigne le répertoire parent.

1. Allez dans votre répertoire personnel.

   ```
   $ # ces commandes sont équivalentes
   $ cd
   $ cd $HOME
   $ cd $HOME/
   $ cd $HOME/.
   $ cd ~
   $ cd ~/
   $ cd ~/.
   ```

2. Créez un répertoire nommé `TD1`.

   ```
   $ mkdir TD1
   ```

3. Entrez dans `TD1` et vérifiez votre position.

```
$ cd TD1
$ pwd
```

4. Créez un répertoire `TD1-bis` et placez-vous dedans.
5. Créez les répertoires suivants :
   — `TD1-bis-1`
   — `TD1-bis-2`
   — `TD1-bis-3`
6. Entrez dans `TD1-bis-1` et créez un répertoire nommé `test`.
7. Revenez en arrière puis entrez dans `TD1-bis-3`.

```
$ # revenir en arrière
$ cd ..
$ # ces deux commandes sont optionnelles
$ pwd
$ ls
$ cd TD1-bis-3
```

8. Depuis `TD1-bis-3`, créez un répertoire nommé `test` à l'intérieur de `TD1-bis-2`, sans vous déplacer dans ce dernier.

```
$ mkdir ../TD1-bis-2/test
```

9. Revenez dans votre répertoire personnel et affichez l'arborescence complète du répertoire `TD1`.

```
$ cd
$ tree TD1
```

   Si la commande `tree` n'est pas disponible, installez-la :

```
$ # pour macOS :
$ brew install tree
$ # pour Linux :
$ sudo apt update
$ sudo apt install tree
```

10. Supprimez ensuite tous les répertoires dont le nom commence par `TD1-bis-` en une ou plusieurs commandes.

```
$ rm -r TD1/TD1-bis/TD1-bis-1
$ rm -r TD1/TD1-bis/TD1-bis-2
$ rm -r TD1/TD1-bis/TD1-bis-3
$ # ou bien
$ rm -r TD1/TD1-bis/TD1-bis-*
```

> La commande `rm` supprime définitivement des fichiers et des répertoires. Utilisez-la avec précaution et vérifiez toujours le chemin avant de l'exécuter.

1.5 Dans cet exercice, vous allez créer un fichier texte et y écrire plusieurs lignes.

1. Placez-vous dans le répertoire `TD1` et créez un fichier nommé `notes.txt`.

```
$ cd $HOME/TD1
$ touch notes.txt
```

2. Écrivez une première ligne dans le fichier.

```
$ echo "Le bonheur est parfois caché dans l'invisible." > notes.txt
```

3. Ajoutez deux lignes supplémentaires sans effacer les précédentes.

```
$ echo "Rire, c'est vivre deux fois." » notes.txt
$ echo "Après la pluie, le beau temps." » notes.txt
```

4. Affichez le contenu du fichier.

```
$ cat notes.txt
```

5. Ouvrez maintenant le fichier `notes.txt` avec un éditeur de texte (par exemple `gedit`, `nano`, `code`, etc.).

```
$ gedit notes.txt
```

6. Dans la fenêtre de l'éditeur, ajoutez les lignes suivantes à la fin du fichier :

   La nuit porte conseil.
   Petit à petit, l'oiseau fait son nid.

   Enregistrez le fichier puis fermez l'éditeur.

7. Affichez le contenu du fichier pour vérifier que toutes les lignes ont bien été ajoutées.

```
$ cat notes.txt
```

8. Affichez le nombre de lignes du fichier `notes.txt`.

```
$ wc -l notes.txt
```

2 Informations système

Linux permet d'obtenir rapidement des informations sur la machine (processeur, mémoire, etc.) ainsi que sur le système.

2.1 Déterminez les informations suivantes à l'aide de commandes Linux :
- Nom de la machine
- Nom de l'utilisateur courant
- Système d'exploitation (distribution Linux le cas échéant)
- Version du noyau Linux
- Architecture du processeur (ex. `x86_64`, `aarch64`)

Commandes utiles

```
$ hostname
$ whoami
$ uname -a
$ uname -m
$ uname -s
$ lsb_release -a # Linux uniquement
```

2.2 Le processeur (CPU) est l'un des éléments centraux d'une machine.

1. Combien de processeurs logiques possède votre machine ?
2. Quel est le modèle du processeur ?
3. Comparez vos résultats avec un autre étudiant.

Commandes utiles

```
$ lscpu # pour macOS : sysctl -a | grep machdep.cpu
$ nproc
```

2.3 La mémoire vive est utilisée pour exécuter les programmes en cours.

1. Quelle est la quantité totale de mémoire vive ?
2. Quelle quantité est actuellement utilisée ?

Commandes utiles

```
$ free -h # pour macOS : vm_stat
```

2.4 Le programme `neofetch` permet d'afficher un résumé visuel des informations système. Installez-le à l'aide des commandes suivantes :

```
$ # pour macOS :
$ brew install neofetch
$ # pour Linux :
$ sudo apt update
$ sudo apt install neofetch
```

Puis lancez-le :

```
$ neofetch
```

et comparez les informations affichées avec celles que vous avez obtenues à l'aide des commandes précédentes.

2.5 Linux organise les disques de stockage à l'aide de points de montage.

1. Quelle est la taille de l'espace disque contenant votre répertoire personnel ?
2. Combien d'espace est actuellement disponible ?
3. Quelle est la taille de votre répertoire personnel ?

Commandes utiles

```
$ df -h ~
$ du -h --max-depth=1 ~
$ # ou bien :
$ du -h -d 1 ~
```

3 Variables en Bash

Dans cette section, nous allons introduire la notion de variable en Bash et apprendre à manipuler des données simples à l'aide du terminal.

En Bash, les espaces peuvent être significatifs selon le contexte. Par exemple, il ne doit pas y avoir d'espace autour du symbole = lors de la définition d'une variable.

Le shell Bash est sensible aux espaces et à la casse (majuscules/minuscules). Les variables `prenom`, `Prenom` et `PRENOM` sont considérées comme différentes.

En Bash, une variable ne possède pas de type. C'est le contexte (arithmétique ou textuel) qui détermine la façon dont sa valeur est interprétée.

3.1 Manipulation de chaînes de caractères

— Définissez une variable `prenom` contenant votre prénom.

— Comparez les commandes suivantes et expliquez la différence :

```
$ echo prenom
$ echo $prenom
$ echo $Prenom
```

— Définissez une variable `nom` contenant votre nom de famille.

— Affichez une phrase contenant votre prénom et votre nom sur une seule ligne.

```
$ echo "Je m'appelle $prenom $nom"
```

— Comparez les résultats des commandes suivantes :

```
$ echo "Je suis $prenom"
$ echo 'Je suis $prenom'
```

3.2 Manipulation de variables entières

En Bash, les opérateurs `-gt`, `-lt`, `-eq` permettent de comparer des entiers. Les symboles `>`, `<` et `=` ont une signification différente et ne doivent pas être utilisés pour les comparaisons numériques.

— Définissez une variable `a` contenant la valeur 5 et une variable `b` contenant la valeur 3.
— Affichez le contenu des variables `a` et `b`.
— Essayez d'afficher la somme de `a` et `b` avec la commande suivante :

```
$ echo $a + $b
```

— Calculez la somme de `a` et `b` à l'aide de la syntaxe arithmétique de Bash et stockez le résultat dans une variable `somme`.

```
$ somme=$((a + b))
```

— Affichez la valeur de la variable `somme`.

```
$ echo "somme = $somme"
```

— Modifiez la valeur de la variable `a` en lui ajoutant 1.

```
$ a=$((a + 1))
```

— Calculez le produit de `a` et `b` et stockez le résultat dans une variable `produit`.
— Les commandes de test `[ ... ]` ne produisent pas de sortie. Elles indiquent le résultat d'un test uniquement à l'aide de leur code de retour, accessible via la variable `$?`. Comparez les comportements obtenus et expliquez la différence.

```
$ [ $a < $b ]
$ echo $?
```

```
$ [ $a -lt $b ]
$ echo $?
```