TD2 – Langages de script

Abdallah Ammar

17 janvier 2026

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

Avant de regarder les solutions proposées dans ce TD, prenez le temps de réfléchir et de tenter de résoudre les questions par vous-même. L'erreur fait partie du processus d'apprentissage.

1 Globbing

Le *globbing* est un mécanisme du shell permettant de faire correspondre des motifs (*patterns*) à des noms de fichiers.

Le globbing est effectué par le shell avant l'exécution de la commande. Les caractères `*`, `?` et `[...]` ont une signification spéciale et peuvent être désactivés à l'aide de guillemets ou d'un antislash.

— Créez les fichiers suivants dans un répertoire de test :

```
$ touch a.txt b.txt c.txt aa.txt bb.txt cc.txt
$ touch a.java b.java c.java
$ touch ba.txt ab.txt ca.txt ac.txt
$ touch note.pdf image.png
```

— Exécutez les commandes suivantes :

```
$ ls
$ ls *.txt
$ ls *.java
$ ls *.png
$ ls *.pdf
$ ls *.*
$ ls *.TXT
```

Que remarquez-vous ?

Le caractère `*` correspond à n'importe quelle suite de caractères (dans un nom de fichier).

— Exécutez les commandes suivantes :

```
$ ls ?.txt
$ ls a?.txt
```

```
$ ls A?.txt
$ ls ?c.txt
$ ls ??.txt
$ ls ???.txt
```

Le caractère `?` correspond exactement à un seul caractère.

— Exécutez les commandes suivantes :

```
$ ls [ab].txt
$ ls [ba].txt
$ ls [bc].txt
$ ls [ac].txt
$ ls a[bc].txt
$ ls [bc]a.txt
$ ls [abc][abc].txt
```

Les crochets `[ ]` permettent de rechercher un caractère parmi plusieurs possibilités. Par exemple, `[ab]` correspond à soit 'a' soit 'b'.

— Comparez les commandes suivantes :

```
$ ls *.txt
$ ls "*.txt"
```

Lorsque le motif est entouré de guillemets, le globbing n'est pas interprété par le shell.

— Exécutez les commandes suivantes :

```
$ ls *
$ ls .*
```

Par défaut, le globbing ne correspond pas aux fichiers cachés (commençant par un point).

2 Premier script Bash

L'objectif de cette question est de découvrir la création et l'exécution d'un script Bash simple.

— Créez un répertoire nommé `TD2` dans votre répertoire personnel (`$HOME`).

```
$ # d'abord, on se place dans le répertoire HOME
$ cd $HOME
$
$ # ensuite, on crée le répertoire TD2
$ mkdir TD2
$
$ # optionnel :
$ pwd
$ ls
```

— Placez-vous dans le répertoire `TD2` et créez un sous-répertoire nommé `Q1`.

```
$ # pour aller dans TD2
$ cd TD2
$ # ou bien :
$ cd $HOME/TD2
$
$ mkdir Q1
$
$ # optionnel :
$ pwd
$ ls
```

— Placez-vous dans le répertoire `Q1` et créez un fichier nommé `hello.sh`.

```
$ # pour aller dans Q1
$ cd Q1
$ # ou bien :
$ cd $HOME/TD2/Q1
$
$ # pour créer le fichier :
$ touch hello.sh
$
$ # optionnel :
$ pwd
$ ls
```

Ajouter une extension aux fichiers, comme `.sh` dans `hello.sh`, est optionnel sous Linux. Cependant, c'est une bonne pratique : cela facilite la lecture et permet aux éditeurs de texte de colorer la syntaxe.

— Éditez le fichier `hello.sh` afin qu'il affiche `Hello World` à l'écran, en y ajoutant les lignes suivantes :

```
echo "Hello World";
echo "Ceci est mon premier script, mais certainement pas le dernier ;)";
```

Le point-virgule `;` est optionnel ici car chaque commande est sur une ligne distincte, mais il permet de séparer plusieurs commandes sur une même ligne.

Rappel : On peut lancer un éditeur de texte (`gedit`, `nano`, etc.) avec :

```
$ # pour utiliser gedit :
$ gedit hello.sh &
$ # pour utiliser nano :
$ nano hello.sh &
```

L'utilisation de `&` permet de lancer le programme en arrière-plan et de conserver le contrôle du terminal. Le shell crée alors un processus fils pour exécuter la commande.

— Exécutez le script en utilisant la commande `bash`.

```
$ bash hello.sh
```

— Modifiez le fichier `hello.sh` pour ajouter une ligne de *shebang* en début de fichier :

```
#!/usr/bin/env bash

echo "Hello World"
echo "Ceci est mon premier script, mais certainement pas le dernier ;)"
```

Le shebang `#!/usr/bin/env bash` indique au système quel interpréteur doit être utilisé pour exécuter le script.
Sans cette ligne, lorsque l'on tente d'exécuter le fichier avec `./hello.sh`, le système essaie de l'exécuter comme un programme binaire, ce qui échoue car il s'agit d'un script texte.
Il est également possible d'utiliser `#!/bin/bash`. Cependant, avec `#!/usr/bin/env bash`, le système utilise la commande `env` pour rechercher automatiquement l'interpréteur `bash` dans les chemins définis par la variable `PATH`, ce qui rend le script plus portable d'un système à un autre.

— Si on inspecte les droits du fichier avec :

```
$ ls -l hello.sh
```

on constate qu'il n'est pas exécutable par défaut (`-rw-rw-r-`). Pour le rendre exécutable (`-rwxrwxr-x`), on utilise :

```
$ chmod +x hello.sh
```

Les droits d'un fichier sont définis par :
- `r` : droit de lecture (read)
- `w` : droit d'écriture (write)
- `x` : droit d'exécution (execute)

Ces droits sont définis pour le propriétaire du fichier, le groupe et les autres utilisateurs.

— Exécutez le script directement, sans utiliser la commande `bash`.

```
$ # avec un chemin relatif :
$ ./hello.sh
$ # avec un chemin absolu :
$ $HOME/TD2/Q1/hello.sh
```

— Essayez de l'exécuter sans préciser de chemin :

```
$ hello.sh
```

Cette commande ne fonctionne pas car le répertoire courant n'est pas inclus dans la variable d'environnement `PATH`. Contrairement à des commandes comme `ls` ou `pwd`, le shell ne sait pas où chercher `hello.sh`.

— Modifiez le script pour définir une variable nommée `prenom` contenant votre prénom, et affichez le message :

```bash
#!/usr/bin/env bash

prenom="Toto"

echo "Coucou ${prenom}"
```

Il est recommandé d'utiliser `{}` autour des variables afin d'éviter toute ambiguïté lors de l'expansion.

— Modifiez le script afin de lire le prénom lors de l'exécution à l'aide de la commande `read`.

```bash
#!/usr/bin/env bash

echo "Entrez votre prénom :"
read prenom

echo "Coucou ${prenom}"
```

— Modifiez le script afin que le prénom soit fourni comme argument lors de l'exécution du script.

```bash
#!/usr/bin/env bash

prenom=$1

echo "Coucou ${prenom}"
```

Lors de l'exécution d'un script Bash, certaines variables spéciales sont définies automatiquement.
`$0` contient le nom du script tel qu'il a été utilisé pour le lancer.
`$1`, `$2`, ... correspondent aux arguments passés au script : `$1` est le premier argument, `$2` le second, etc.

— Modifiez le script afin de demander également l'âge de l'utilisateur, puis affichez un message prenant en compte cet âge.

```bash
#!/usr/bin/env bash

prenom=$1
age=$2

echo "Coucou ${prenom}"
echo "J'espère que ton ${age}e anniversaire s'est bien passé !"
```

— Modifiez le script afin d'afficher une phrase indiquant que le prochain anniversaire sera encore meilleur, en utilisant un calcul sur l'âge.

```bash
#!/usr/bin/env bash

prenom=$1
age=$2

echo "Coucou ${prenom}"
echo "J'espère que ton ${age}e anniversaire s'est bien passé !"

prochain_age=$((age + 1))
echo "Le ${prochain_age}e sera encore meilleur :)"
```

— Modifiez le script afin de réaliser l'affichage à l'aide d'un stégosaure (en utilisant `cowsay`).

```bash
#!/usr/bin/env bash

prenom=$1
age=$2

cowsay -f stegosaurus "Coucou ${prenom}"
cowsay -f stegosaurus "J'espère que ton ${age}e anniversaire s'est bien
↪  passé !"

prochain_age=$((age + 1))
cowsay -f stegosaurus "Le ${prochain_age}e sera encore meilleur :)"
```

3 Récupération des fichiers de données

Dans ce TD, vous allez travailler sur des fichiers texte fournis par l'enseignant. Ces fichiers contiennent des données fictives destinées à l'apprentissage des commandes de filtrage et d'analyse de texte.

Commencez par créer un répertoire `data`, puis téléchargez les fichiers à l'aide des commandes suivantes :

```
$ mkdir data
$ cd data
$ wget https://raw.githubusercontent.com/AbdAmmar/LDS/main/TD/TD2/data/users.txt
$ wget https://raw.githubusercontent.com/AbdAmmar/LDS/main/TD/TD2/data/logs.txt
$ wget https://raw.githubusercontent.com/AbdAmmar/LDS/main/TD/TD2/data/notes.txt
```

Vérifiez que les fichiers ont bien été téléchargés. Si la commande `wget` n'est pas disponible, installez-la avec :

```
$ sudo apt update
$ sudo apt install wget
```

4 Lire et compter

Travail à faire

1. Affichez le contenu du fichier `users.txt`.

2. Comptez le nombre de lignes du fichier.
3. Affichez le contenu du fichier `logs.txt`.
4. Comptez le nombre de lignes du fichier `notes.txt`.

Commandes utiles

```
$ cat users.txt
$ wc -l users.txt
```

5 Introduction aux pipes

Un *pipe* permet d'envoyer la sortie d'une commande en entrée d'une autre.

Travail à faire

1. Comptez le nombre de lignes de `users.txt` sans afficher le contenu du fichier.
2. Expliquez le rôle du symbole `|`.

Commande clé

```
$ cat users.txt | wc -l
```

6 Rechercher des informations avec `grep`

Travail à faire

À partir du fichier `users.txt` :

1. Affichez les lignes contenant `bash`.
2. Comptez le nombre d'utilisateurs utilisant `bash`.
3. Affichez les lignes ne contenant pas `bash`.

Commandes utiles

```
$ grep bash users.txt
$ grep -c bash users.txt
$ grep -v bash users.txt
```

7 Extraire des informations

Le fichier `users.txt` est structuré : les champs sont séparés par le caractère `:`.

Travail à faire

1. Affichez uniquement la colonne des noms d'utilisateurs.
2. Affichez uniquement la colonne des shells.

Commandes utiles

```
$ cut -d: -f1 users.txt
$ cut -d: -f2 users.txt
```

8 Trier et compter

Travail à faire

1. Affichez la liste des shells utilisés.
2. Triez cette liste.
3. Comptez combien de fois chaque shell apparaît.

Pipeline attendu

```
$ cut -d: -f2 users.txt | sort | uniq -c
```

9 Analyse de fichiers de logs

Travail à faire

À partir du fichier `logs.txt` :

1. Affichez uniquement les lignes contenant `ERROR`.
2. Comptez le nombre d'erreurs.
3. Affichez les noms des utilisateurs ayant généré une erreur.

Commandes utiles

```
$ grep ERROR logs.txt
$ grep -c ERROR logs.txt
$ grep ERROR logs.txt | cut -d= -f2
```

10 Mini-analyse

Travail à faire

Sans utiliser d'éditeur de texte :
— Quel est le shell le plus utilisé ?
— Quel utilisateur apparaît le plus souvent dans les logs ?
Justifiez vos réponses à l'aide de commandes Linux.

11 Les flux standards

Ce TD a pour objectif d'introduire la gestion des sorties et des erreurs sous Linux. Sous Linux, un programme communique avec l'extérieur à l'aide de flux :
— l'entrée standard (stdin) ;
— la sortie standard (stdout) ;
— la sortie d'erreur (stderr).
Par défaut, la sortie standard et la sortie d'erreur sont affichées à l'écran.

12 Rediriger la sortie standard

Travail à faire

1. Listez le contenu de votre répertoire personnel.
2. Redirigez cette sortie dans un fichier `liste.txt`.
3. Vérifiez le contenu du fichier.

Commandes

```
$ ls
$ ls > liste.txt
$ cat liste.txt
```

13 Générer et observer une erreur

Travail à faire

1. Essayez d'afficher un fichier qui n'existe pas.
2. Observez le message affiché.

```
$ cat fichier_inexistant.txt
```

Expliquez pourquoi le message n'est pas redirigé avec `>`.

14 Rediriger la sortie d'erreur

La sortie d'erreur peut être redirigée séparément.

Travail à faire

1. Redirigez l'erreur précédente dans un fichier `erreur.txt`.
2. Vérifiez le contenu du fichier.

```
$ cat fichier_inexistant.txt 2> erreur.txt
$ cat erreur.txt
```

15 Rediriger sortie et erreur

Il est possible de rediriger à la fois la sortie standard et la sortie d'erreur.

Travail à faire

1. Lancez une commande produisant à la fois une sortie et une erreur.
2. Redirigez les deux flux dans un même fichier.

```
$ ls fichier_inexistant > sortie.txt 2> erreur.txt
```

Ou en une seule commande :

```
$ ls fichier_inexistant > tout.txt 2>&1
```