TD3 – Langages de script

Abdallah Ammar

19 janvier 2026

1 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, placez-vous dans votre répertoire personnel
$ 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 :

```
#!/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`.

```
#!/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.

```
#!/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.

```
#!/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.

```
#!/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`).

```
#!/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 :)"
```

2 Tests et conditions simples

Les structures conditionnelles permettent d'exécuter du code en fonction d'une condition.

Travail à faire

1. Si aucun paramètre n'est fourni, affichez un message d'erreur.
2. Sinon, affichez le paramètre fourni.

Structure générale

```
if [ condition ]; then commandes fi
```

3 Tester l'existence d'un fichier

Travail à faire

Écrivez un script qui :
— prend un nom de fichier en paramètre ;
— affiche `"Fichier trouvé"` s'il existe ;
— affiche `"Fichier inexistant"` sinon.

Tests utiles

```
-f fichier   (teste si c'est un fichier) -d dossier   (teste si c'est un
répertoire)
```

4 Mini-script d'automatisation

Travail à faire

Écrivez un script qui :
— prend un répertoire en paramètre ;
— vérifie qu'il existe ;
— affiche le nombre de fichiers qu'il contient.

Indications

Vous pouvez utiliser les commandes suivantes :
— `ls`
— `wc -l`

Remarques

— Un script est simplement une suite de commandes déjà connues.
— Les erreurs sont normales lors de l'écriture d'un script.
— Lisez attentivement les messages d'erreur.

Pour aller plus loin – Exercice ludique (optionnel)

Écrivez un script bash affichant un message personnalisé à l'utilisateur, en utilisant une variable et un paramètre.

Par exemple :

— `./bonjour.sh Alice`

— affiche : `Bonjour Alice !`

5 Pourquoi utiliser une boucle ?

Lorsque l'on souhaite appliquer le même traitement à plusieurs éléments (fichiers, lignes, nombres, etc.), écrire une commande par élément n'est ni pratique ni robuste.

Question

Expliquez pourquoi il n'est pas raisonnable d'écrire une commande différente pour chaque fichier lorsque le nombre de fichiers peut varier.

6 Boucle `for` simple

Travail à faire

1. Écrivez un script affichant les nombres de 1 à 5.
2. Modifiez le script pour afficher les nombres de 1 à 10.

Structure générale

```
for i in 1 2 3 4 5; do echo $i done
```

7 Boucle sur des fichiers

Travail à faire

Dans un répertoire contenant plusieurs fichiers texte :

1. Affichez le nom de chaque fichier `.txt`.
2. Affichez le nombre de lignes de chaque fichier.

Indication

```
for f in *.txt; do echo $f wc -l $f done
```

8 Automatisation sur les fichiers du TD3

Dans le TD3, vous avez manipulé plusieurs fichiers texte dans un répertoire `data`.

Travail à faire

1. Placez-vous dans le répertoire `data`.
2. Parcourez tous les fichiers texte.
3. Pour chaque fichier, affichez :
   — son nom ;
   — son nombre de lignes.

9 Lire un fichier ligne par ligne

Il est possible de lire un fichier texte ligne par ligne à l'aide d'une boucle `while`.

Travail à faire

1. Écrivez un script qui lit le fichier `users.txt` ligne par ligne.
2. Affichez chaque ligne précédée du texte `"Utilisateur :"`.

Structure générale

```
while read line; do echo "Utilisateur : $line" done < users.txt
```

10 Mini-script d'automatisation

Travail à faire

Écrivez un script qui :
— prend un répertoire en paramètre ;
— vérifie que ce répertoire existe ;
— parcourt tous les fichiers `.txt` qu'il contient ;
— affiche pour chacun :
   — le nom du fichier ;
   — le nombre de lignes.

Indications

Vous pouvez utiliser :
— les paramètres du script (`$1`) ;
— le test `-d` pour vérifier l'existence d'un répertoire ;
— les commandes `ls` et `wc -l`.

Remarques

— Une boucle permet d'éviter la répétition de code.
— Un script doit rester simple et lisible.
— En cas d'erreur, lisez attentivement les messages affichés.

Pour aller plus loin – Exercice ludique (optionnel)

Écrivez un script utilisant une boucle `for` pour afficher plusieurs fois un message ou une suite de symboles à l'écran.

Par exemple :
— afficher une ligne de `*`
— afficher les nombres de 1 à 20

11 Contexte

Vous disposez de fichiers texte contenant des informations sur des utilisateurs et des événements (logs). Ces fichiers sont ceux utilisés lors du TD3.

L'objectif est d'écrire un script bash unique permettant d'analyser automatiquement ces fichiers.

12 Consignes générales

— Le script doit s'appeler `analyze.sh`.
— Il doit être exécutable.
— Il prend un répertoire en paramètre.
— Il doit vérifier que ce répertoire existe.
— Il doit produire une sortie lisible et structurée.
— L'utilisation d'éditeurs graphiques est interdite.

13 Étape 1 – Vérification des paramètres

Travail à faire

1. Si aucun paramètre n'est fourni, affichez un message d'erreur.
2. Si le paramètre fourni n'est pas un répertoire, affichez un message d'erreur.

Indications

Vous pouvez utiliser :
— les paramètres du script (`$1`) ;
— le test `-d`.

14 Étape 2 – Analyse des utilisateurs

À partir du fichier `users.txt` présent dans le répertoire fourni :

Travail à faire

1. Affichez le nombre total d'utilisateurs.
2. Affichez la liste des shells utilisés.
3. Indiquez le shell le plus utilisé.

Indications

Vous pouvez réutiliser les commandes vues en TD3 et TD5 :
— `cut`
— `sort`
— `uniq`
— `wc`

15 Étape 3 – Analyse des logs

À partir du fichier `logs.txt` présent dans le répertoire fourni :

Travail à faire

1. Affichez le nombre total d'événements.
2. Affichez le nombre d'erreurs.
3. Affichez la liste des utilisateurs ayant généré une erreur.

Indications

Vous pouvez utiliser :
— `grep`
— `cut`
— `sort`
— `uniq`

16 Étape 4 – Affichage structuré

Votre script doit afficher les résultats de manière claire, avec des titres explicites pour chaque partie de l'analyse.

Exemple indicatif de sortie

```
=== Analyse des utilisateurs === Nombre total d'utilisateurs : 5 Shells utilisés :
/bin/bash (3) /bin/zsh (1) /usr/sbin/nologin (1)
=== Analyse des logs === Nombre total d'événements : 20 Nombre d'erreurs : 4
Utilisateurs ayant généré une erreur : alice bob
```

Cet exemple est fourni à titre indicatif. La présentation exacte peut varier.

17 Extensions (optionnelles)

Pour les étudiants les plus rapides :
— vérifier la présence des fichiers avant analyse ;
— sauvegarder les résultats dans un fichier ;
— ajouter une option `-h` affichant une aide ;
— analyser automatiquement tous les fichiers `.txt` du répertoire.

Remarques

— Ce TD ne demande aucun concept nouveau.
— Réutilisez les commandes et scripts déjà écrits précédemment.
— Testez votre script avec différents cas.
— Un script clair et simple est préférable à un script complexe.

Pour aller plus loin – Exercice ludique (optionnel)

Recherchez dans les fichiers de logs :
— les messages contenant `WARNING` ;
— les messages contenant `ERROR` ;
— les messages contenant votre prénom (s'il y en a).
Expliquez ce que vous observez.

Récupération des fichiers de logs

Ce TD a pour objectif d'apprendre à analyser des fichiers de logs similaires à ceux que l'on trouve sur un système Linux réel.

Créez un répertoire `data` et téléchargez les fichiers de logs :

```
$ mkdir data
$ cd data
$ wget https://raw.githubusercontent.com/AbdAmmar/LDS/main/TD/TD3/data/auth.log
$ wget https://raw.githubusercontent.com/AbdAmmar/LDS/main/TD/TD3/data/syslog
$ wget https://raw.githubusercontent.com/AbdAmmar/LDS/main/TD/TD3/data/kernel.log
```

18 Lecture et observation des logs

Travail à faire

1. Affichez le contenu de `auth.log`.
2. Combien de lignes contient ce fichier ?
3. Repérez les informations suivantes :
   — date et heure ;
   — service concerné ;
   — message.

19 Recherche d'événements

Travail à faire

À partir de `auth.log` :

1. Affichez les tentatives de connexion échouées.
2. Affichez les connexions réussies.
3. Affichez les lignes contenant `sudo`.

Commandes utiles

```
$ grep Failed auth.log
$ grep Accepted auth.log
$ grep sudo auth.log
```

20 Analyse par pipelines

Travail à faire

1. Comptez le nombre total de tentatives de connexion échouées.
2. Affichez la liste des utilisateurs concernés.

Exemples

```
$ grep Failed auth.log | wc -l
$ grep Failed auth.log | cut -d" " -f9 | sort | uniq
```

21 Analyse temporelle

Travail à faire

1. Recherchez les événements ayant eu lieu à une heure donnée (par exemple 10h).
2. Comptez le nombre d'événements sur cette plage horaire.

```
$ grep "Jan 10 10" auth.log
```

22 Script : génération d'un rapport de sécurité

Travail à faire

Écrivez un script `report.sh` qui :
— prend un répertoire de logs en paramètre ;
— analyse `auth.log` ;
— affiche :
  — le nombre de connexions réussies ;
  — le nombre de connexions échouées ;
  — la liste des utilisateurs impliqués ;
— produit un rapport lisible à l'écran.

Exemple indicatif de sortie

```
=== Rapport de sécurité === Connexions réussies : 3 Connexions échouées : 5
Utilisateurs concernés : alice bob
```

23 Extensions (optionnelles)

— analyser `syslog` ou `kernel.log` ;
— détecter des avertissements ;
— sauvegarder le rapport dans un fichier.

Pour aller plus loin – Exercice ludique (optionnel)

Une fois connecté à la machine distante via SSH, lancez le programme `neofetch`.
Comparez les informations affichées avec celles de votre machine locale.

24 Installation des outils

Ce TD a pour objectif de comprendre comment créer une commande personnalisée sous Linux, à la manière de la commande `fortune`.

Installez les programmes nécessaires :

```
$ sudo apt update
$ sudo apt install cowsay
```

25 Création des citations

Créez un fichier contenant des citations, blagues ou proverbes en français.

Travail à faire

1. Créez un fichier nommé `citations.txt`.
2. Ajoutez au moins 5 phrases en français (une par ligne).

Exemple

```
$ nano citations.txt
```

Contenu possible :

```
Linux est puissant.
Tout est fichier.
Ça marche chez moi.
Il n'y a pas de bug, seulement des fonctionnalités.
RTFM.
```

26 Afficher une citation aléatoire

Linux fournit la commande `shuf` permettant de choisir une ligne aléatoire.

Travail à faire

Affichez une citation aléatoire depuis le fichier `citations.txt`.

```
$ shuf -n 1 citations.txt
```

27 Création d'une commande personnalisée

Nous allons maintenant créer une commande appelée `fortune-fr`.

Travail à faire

1. Créez un fichier nommé `fortune-fr`.
2. Ajoutez la ligne `#!/bin/bash`.
3. Écrivez un script qui affiche une citation aléatoire.

Exemple de script

```
#!/bin/bash shuf -n 1 $HOME/citations.txt
```

Rendez le script exécutable :

```
$ chmod +x fortune-fr
```

Testez-le :

```
$ ./fortune-fr
```

28 Ajouter la commande au PATH

Pour pouvoir utiliser votre commande depuis n'importe quel répertoire, elle doit se trouver dans un répertoire du `PATH`.

Travail à faire

1. Créez le répertoire `$HOME/bin` s'il n'existe pas.
2. Déplacez votre script dans ce répertoire.
3. Vérifiez que `$HOME/bin` est bien dans le `PATH`.

```
$ mkdir -p $HOME/bin
$ mv fortune-fr $HOME/bin
$ echo $PATH
```

Si nécessaire, ajoutez cette ligne à votre fichier `~/.bashrc` :

```
export PATH=$PATH:$HOME/bin
```

Rechargez la configuration :

```
$ source ~/.bashrc
```

29 Combiner avec cowsay

Utilisez maintenant votre commande personnalisée avec `cowsay`.

```
$ fortune-fr | cowsay
```

Essayez différentes exécutions.

30 Extensions (optionnelles)

Pour aller plus loin :
— ajouter une option `-h` affichant une aide ;
— gérer plusieurs fichiers de citations ;
— afficher le nombre total de citations ;
— utiliser une vache différente avec `cowsay -l`.

31 Pourquoi structurer un script ?

Ce TD a pour objectif d'apprendre à écrire des scripts bash plus lisibles, plus structurés et plus faciles à maintenir.

Jusqu'à présent, vous avez écrit des scripts contenant une suite de commandes. Lorsque les scripts deviennent plus longs, il devient difficile :
— de comprendre ce qu'ils font ;
— de les modifier ;
— de corriger des erreurs.

Structurer un script permet de le rendre plus lisible et plus robuste.

32 Découper un script en fonctions

Une fonction permet de regrouper des commandes ayant un rôle précis.

Travail à faire

Reprenez un script écrit lors d'un TD précédent (TD6 ou TD7 par exemple) et identifiez les différentes parties du script.

Par exemple :
— vérification des paramètres ;
— analyse d'un fichier ;
— affichage des résultats.

Créez une fonction pour chacune de ces parties.

33 Définir et utiliser des fonctions

Structure générale

```
nom_fonction() { commandes }
```

Une fonction peut ensuite être appelée simplement par son nom.

Travail à faire

1. Créez une fonction `check_args` qui vérifie les paramètres.
2. Créez une fonction `analyze` qui effectue le traitement principal.
3. Créez une fonction `print_result` qui affiche les résultats.
4. Appelez ces fonctions dans la partie principale du script.

34 Variables et portée

Les variables définies dans un script sont accessibles dans les fonctions.

Travail à faire

1. Définissez une variable au début du script.
2. Modifiez cette variable dans une fonction.
3. Affichez la variable après l'appel de la fonction.

Question

Que constatez-vous ? Expliquez pourquoi la valeur de la variable a changé.

35 Ajouter une aide au script

Un script bien écrit doit expliquer comment il s'utilise.

Travail à faire

Ajoutez une option `-h` ou `-help` à votre script. Lorsque cette option est fournie, le script doit afficher :
— une description du script ;
— les paramètres attendus ;
— un exemple d'utilisation.

Indication

Vous pouvez utiliser une structure conditionnelle du type :

```
if [ "$1" = "-h" ]; then echo "Aide du script" exit 0 fi
```

36 Organisation finale du script

Votre script final doit être organisé de la manière suivante :
— commentaires en début de fichier ;
— définition des variables ;
— définition des fonctions ;
— partie principale (appels des fonctions).

Travail à faire

Réorganisez votre script pour qu'il respecte cette structure.

Remarques

— Un script lisible est plus important qu'un script court.
— Les fonctions permettent d'éviter les répétitions.
— Ce TD prépare l'écriture de scripts plus complexes.