

Mon premier script Bash

Le shell est aussi un langage de programmation

🎯 Objectifs de ce chapitre

- Comprendre qu'un script est un fichier de commandes interprété par le shell.
- Écrire un script qui utilise des variables, une condition et une boucle.
- Rendre un script exécutable, puis l'exécuter.
- Réinvestir les chemins, les droits et les redirections vus dans les chapitres précédents.

Qu'est-ce qu'un script ?

Jusqu'ici, vous avez saisi des commandes une par une. Un **script** est un fichier texte qui contient une suite de commandes, que le shell exécute les unes après les autres. On l'écrit une fois, et on peut le relancer autant de fois qu'on veut : c'est la façon d'**automatiser** une tâche.

Le langage de Bash n'est pas qu'une liste de commandes : il a des **variables**, des **conditions** et des **boucles**, comme Python, avec une syntaxe différente.

Idée	Python	Bash
Afficher	<code>print("Bonjour")</code>	<code>echo "Bonjour"</code>
Donner une valeur à une variable	<code>nom = "Alice"</code>	<code>nom="Alice"</code> (sans espace autour de =)
Lire la valeur d'une variable	<code>nom</code>	<code>\$nom</code>
Condition	<code>if x > 0:</code> puis un bloc indenté	<code>if [condition]; then ... fi</code>
Boucle	<code>for i in range(3):</code> puis un bloc indenté	<code>for i in 1 2 3; do ... done</code>
Commentaire	<code># commentaire</code>	<code># commentaire</code>

En Bash, les blocs ne sont pas marqués par l'indentation, mais par des mots-clés : `then ... fi` pour une condition, `do ... done` pour une boucle.

💡 Pour aller plus loin

Les administrateurs de systèmes écrivent des scripts tous les jours : sauvegardes, installations, surveillance... Ce chapitre n'est pas évalué : c'est l'occasion de réutiliser tout ce que vous avez appris sur les chemins, les droits et les redirections.

I. Construire le script pas à pas

Vous allez écrire le fichier `mon_script.sh` étape par étape. À chaque étape, ajoutez les lignes à la suite de celles déjà écrites. La fiche récapitulative, à la fin du chapitre, montre le fichier que vous devez obtenir.

Étape 1 — Créer le fichier

Placez-vous dans `TP_OS`, puis ouvre l'éditeur de texte `nano` pour créer le fichier :

```
cd ~/TP_OS
nano mon_script.sh
```

Dans nano	Effet
<code>Ctrl+O</code> , puis Entrée	Enregistre le fichier (« écrire »).
<code>Ctrl+X</code>	Quitte nano.

L'extension `.sh` n'est qu'une habitude : elle indique que c'est un script du shell. Vous pouvez aussi utiliser un éditeur graphique si tu préfères.

Étape 2 — La première ligne : le shebang

```
#!/bin/bash
```

Cette ligne, le **shebang**, indique **quel programme doit interpréter** le fichier : ici `/bin/bash`. Elle doit toujours être la toute première ligne.

Étape 3 — Afficher du texte

```
echo "Bonjour, $USER !"
```

`$USER` est une **variable d'environnement** : le système la remplit avec votre nom d'utilisateur. `$HOME` (ton répertoire personnel) en est une autre.

Essaie, observe — dans le terminal, pas dans le fichier

1. Tape `echo $USER $HOME`. Que s'affiche-t-il ?
2. Compare `echo "Bonjour $USER"` et `echo 'Bonjour $USER'`. Quelle est la différence entre guillemets doubles et apostrophes ?

Étape 4 — Une variable

```
nom="Alice"
echo "Le nom est $nom"
```

Aucun espace autour du signe =

Écris `nom="Alice"` et non `nom = "Alice"`. Avec des espaces, Bash croit que `nom` est une commande et répond « commande introuvable ». Pour la valeur, on place un `$` devant le nom de la variable : `$nom`.

Pour rendre le script interactif, remplace la ligne `nom="Alice"` par une lecture au clavier :

```
read -p "Comment t'appelles-tu ? " nom
echo "Le nom est $nom"
```

Étape 5 — Une condition

```
if [ -d "$HOME/TP_SCRIPT" ]; then
    echo "Le dossier existe déjà."
else
    mkdir "$HOME/TP_SCRIPT"
    echo "Dossier créé."
fi
```

- `[ -d chemin ]` teste si le chemin désigne un **répertoire** (`-f` teste un fichier, `-e` teste si quelque chose existe).
- Les espaces à l'intérieur des crochets sont **obligatoires** : `[` est en réalité une commande (voir `man test`).
- `then`, `else` et `fi` délimitent les blocs. L'indentation n'est là que pour la lisibilité.
- `$HOME/TP_SCRIPT` est un **chemin absolu** : le script fonctionne quel que soit le répertoire d'où on le lance.

Étape 6 — Mini-défi : trois fichiers

Mini-défi

Complétez votre script pour qu'il crée, dans `TP_SCRIPT`, trois fichiers `doc1.txt`, `doc2.txt` et `doc3.txt`. Chacun doit contenir la **date du jour** et être lisible et modifiable par **toi seul**.

Pistes : une boucle `for i in 1 2 3; do ... done`, une redirection `>`, la commande `date`, `chmod`. Pour insérer la valeur de `i` dans un nom de fichier, écris `doc${i}.txt`.

Étape 7 — Une boucle sur les fichiers .txt

```
for f in "$HOME/TP_SCRIPT"/*.txt; do
    echo "Fichier : $f"
    wc -l "$f"
done
```

Le joker `*.txt` est remplacé par la liste des fichiers du répertoire qui se terminent par `.txt`, et la boucle traite chacun d'eux dans la variable `f`. Cette boucle doit donc être écrite **après** la création des fichiers : s'il n'y en a aucun, la boucle reçoit le texte `*.txt` tel quel.

Étape 8 — Rendre le script exécutable et le lancer

Essaie, observe — dans le terminal

1. Enregistrez votre fichier et quitte nano. Tape `ls -l mon_script.sh` : quels sont les droits ?
2. Essayez de le lancer avec `./mon_script.sh`. Que se passe-t-il ? Pourquoi ?
3. Ajoutez le droit d'exécution avec `chmod +x mon_script.sh`, vérifiez avec `ls -l`, puis lancez à nouveau `./mon_script.sh`.
4. Lancez-le une deuxième fois. Qu'est-ce qui a changé dans les messages ? Pourquoi ?
5. Tapez `ls -l ~/TP_SCRIPT` et `cat ~/TP_SCRIPT/doc1.txt`. Les droits et le contenu sont-ils ceux attendus ?

💡 Pourquoi écrire `./mon_script.sh` ?

Le `./` est un **chemin relatif** : `.` désigne le répertoire courant. Si vous tapez seulement `mon_script.sh`, le shell cherche la commande dans les répertoires listés dans la variable `PATH` (tape `echo $PATH`) et ne regarde pas dans le répertoire courant. Avec `./`, vous lui indiquez précisément où se trouve le fichier.

Erreurs fréquentes

Message	Cause probable	Correction
Permission non accordée	Le fichier n'est pas exécutable.	<code>chmod +x mon_script.sh</code>
commande introuvable	Espaces autour du <code>=</code> , ou oubli du <code>./</code> .	<code>nom="Alice"</code> , et <code>./mon_script.sh</code>
erreur de syntaxe	Un <code>then</code> , <code>fi</code> , <code>do</code> ou <code>done</code> manque.	Chaque <code>if</code> a son <code>fi</code> , chaque <code>for</code> son <code>done</code> .
Aucun fichier ou dossier de ce type	Mauvais chemin ou mauvais nom.	Vérifie avec <code>pwd</code> et <code>ls</code> .

À retenir

📖 L'essentiel

- Un **script** est un fichier texte de commandes ; Bash est aussi un langage avec variables, conditions et boucles.
- La première ligne, le **shebang** `#!/bin/bash`, indique l'interpréteur.
- `nom="Alice"` (sans espaces) pour affecter, `$nom` pour lire ; `if [ ... ]; then ... fi ; for ... do ... done`.
- Pour lancer un script : `chmod +x mon_script.sh` (droit d'exécution), puis `./mon_script.sh` (chemin relatif).

Fiche récapitulative : le script à obtenir

Voici le contenu complet du fichier `mon_script.sh`. Les commentaires commencent par `#`.

```
#!/bin/bash

# Affichage d'une variable d'environnement
echo "Bonjour, $USER !"

# Une variable saisie par l'utilisateur
read -p "Comment t'appelles-tu ? " nom
echo "Le nom est $nom"

# Vérifier / créer un répertoire
if [ -d "$HOME/TP_SCRIPT" ]; then
    echo "Le dossier existe déjà."
else
    mkdir "$HOME/TP_SCRIPT"
    echo "Dossier créé."
fi

# Création de trois fichiers contenant la date
for i in 1 2 3; do
    date > "$HOME/TP_SCRIPT/doc${i}.txt"
    chmod 600 "$HOME/TP_SCRIPT/doc${i}.txt"
done

# Parcours des fichiers .txt du répertoire
for f in "$HOME/TP_SCRIPT"/*.txt; do
    echo "Fichier : $f"
    wc -l "$f"
done
```

⚠ Les deux commandes ci-dessous ne font pas partie du script

Elles se tapent dans le **terminal**, après avoir enregistré le fichier :

```
chmod +x mon_script.sh
./mon_script.sh
```

Pour aller plus loin (bonus)

- Demandez à l'utilisateur **combien de fichiers** créer, avec `read`, et utilisez `for i in $(seq 1 "$n")` pour les créer.
- Écrivez dans chaque fichier le nom saisi **et** la date : `echo "$nom : $(date)" > ....`
- Affichez un message d'erreur si l'utilisateur n'a rien saisi (cherche le test `[ -z "$nom" ]`).