

## Examen Scripting Infra M1 (durée 3h30)

Ce TP est à réaliser sur une machine **individuellement**.

Les recherches internet et les cours sont **autorisés**. La communication est **interdite**.

Le rendu prendra la forme d'une unique **archive zip** nommée

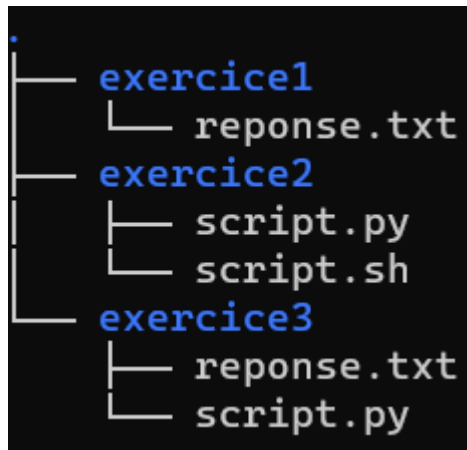
**nom\_prenom\_scripting.zip** qui contiendra tous les fichiers demandés. Elle sera déposée sur le moodle dans la section créée pour l'occasion.

**Pour chaque exercice**, il sera demandé de créer un **répertoire associé** comme dans l'exemple présenté ci-dessous.

Les réponses aux questions de chaque exercice seront écrites dans le dossier associé dans un fichier **reponse.txt**

Si un script est demandé en python ou en bash, il sera écrit dans le dossier associé

Exemple d'architecture de rendu :



```
├── exercice1
│   └── reponse.txt
├── exercice2
│   ├── script.py
│   └── script.sh
└── exercice3
    ├── reponse.txt
    └── script.py
```

**Exercice 1** : Quel usage pour Python & Bash ?

**Dans le fichier reponse.txt**

Expliquer les avantages et inconvénients de Bash et de Python pour le scripting.

Expliquer en quoi leur usage est complémentaire.

**Exercice 2** : Planification de tâches

Vous êtes sur un système Linux et vous souhaitez une fois par semaine faire une copie des fichiers de votre dossier Documents vers un autre disque pour ne pas perdre des données importantes.

**Dans le fichier reponse.txt**

- 1) Détailler spécifiquement la démarche pour mettre en place un système comme celui-ci
- 2) Comment faire en sorte de lancer ce script :
  - Les 4eme, 7eme et 23eme jours du mois à 16 heures
  - Une fois tous les deux jours à 9 heures
  - Tous les lundis à 13 heures
  - Au démarrage de la machine

### **Exercice 3 :** Expressions régulières

Lien : [http://www.3zsoftware.com/listes/liste\\_francais.zip](http://www.3zsoftware.com/listes/liste_francais.zip)

#### **Dans le fichier reponse.txt :**

- 1) Quelle commande bash permet de télécharger ce fichier zip dans un fichier appelé "exo.zip" (utiliser >)
- 2) Chercher une commande linux qui permet de dézipper le fichier (détailler le processus d'installation)

Un fichier liste\_francais.txt a été extrait, écrire des expressions régulières et utiliser grep -Eo pour récupérer les résultats des expressions sur le fichier.

- 3) A quoi sert le tag "-Eo" de la commande grep?
- 4) Quelle commande utiliser pour obtenir le nombre de lignes des résultats obtenus par la commande grep (utiliser un pipe)?

Ecrire les commandes grep et le nombre de lignes obtenues pour chacune des expressions suivantes :

- 5) celle qui extrait les mots qui commencent par b et qui finissent par e
- 6) celle qui extrait les mots qui commencent par ba, puis une consonne, puis une voyelle et qui terminent par e
- 7) celle qui extrait les mots qui commencent par p et qui possèdent un total de 6 lettres
- 8) celle qui extrait les mots qui alternent successivement des consonnes et des voyelles

### **Exercice 4:** Interagir avec une API avec un langage de script

#### **Dans un fichier requete\_bash.sh**

- 1) Créer le script bash qui prend **trois paramètres** en argument : le nom du meme, le texte du haut et le texte du bas, qui permet de générer une image grâce à cette API : <http://apimeme.com/?ref=apilist.fun>

Les trois paramètres seront injectés dans l'url qui permet de générer l'image

Usage :

```
./requete_bash.sh "10-Guy" "bla1" "bla2"
```

Output : Une image .png du meme est enregistrée sur la machine

#### **Dans un scrape\_meme.sh**

- 2) Créer un script qui permet de récupérer tous les noms de meme de la page html <http://apimeme.com/?ref=apilist.fun> et les écrire dans un fichier **name\_memes.txt** (utiliser l'inspecteur d'éléments, ils sont cachés dans les balises <option></option>)
- 3) Dans le fichier **requete\_bash.sh**

Grâce aux commandes head -n, tail et la génération d'un nombre aléatoire, adapter le script pour récupérer un nom de meme tiré au hasard du fichier **name\_memes.txt** et l'injecter dans la requête

### **Exercice 5)** Automatisation & configuration d'un serveur web

On vous met à disposition un serveur linux vierge avec rien d'installé dessus (même pas python!)

**Dans un fichier configure\_server.sh**

1) Créer un script bash **configure\_and\_run\_server.sh** qui permet d'installer toutes les dépendances nécessaires permettant de lancer un serveur web avec python flask.

Voici l'architecture que le script doit créer et le contenu du fichier app\_flask.py :

```
├── configure_server.sh
├── web
│   ├── app_flask.py
│   └── blocked_ip.txt
```

```
1 from flask import Flask, request
2 app = Flask(__name__)
3 @app.route("/")
4 def check_ip():
5     return f"Connecté avec l'adresse {request.remote_addr}"
6 app.run()
```

2) Adapter le script **app.py** pour vérifier que l'adresse du client qui se connecte est présent dans la liste d'ip définies dans blocked\_ip.txt, si c'est le cas retourner le message "Accès refusé", sinon retourner le message normal "connecté avec l'adresse ...".

3) Créer un fichier blocked\_ip.json, comme l'exemple suivant :

```
1 {
2   "127.0.0.1": "Hello there localhost",
3   "191.74.9.76": "Hey James !",
4   "191.74.10.66": "Hey Sarah !"
5 }
```

**Dans le fichier app.py**, charger le fichier json grâce à la librairie json (chercher sur internet comment charger un fichier json avec cette librairie).

Un fichier json se transforme en dictionnaire en python.

Ensuite, si l'adresse client est présente dans le dictionnaire (par exemple 127.0.0.1), retourner le message personnalisé associé (par exemple Hello there localhost).

### **Exercice 6)** Gestion de processus et logging

Vous voulez empêcher des utilisateurs d'utiliser certains programmes sur votre machine

La librairie psutil est une librairie python permettant de monitorer les processus et les ressources systèmes.

Documentation : <https://psutil.readthedocs.io/en/latest/#>

Voici un code fonctionnel permettant d'afficher le nom des processus qui tournent sur la machine

```
1 import psutil
2 for proc in psutil.process_iter():
3     print(proc.name())
```

**Dans un fichier `exo_ps.py`:**

- 1) Ecrire un code qui parcourt tous les processus. Si l'un d'entre eux s'appelle "Teams.exe" tuer le processus avec la méthode `kill()`
- 2) Modifier ce script pour définir une **liste** de noms de programmes à tuer au lieu de l'unique programme "Teams.exe". Adapter le code en conséquence.
- 3) Modifier le script pour faire en sorte que cette vérification s'opère toutes les 5 secondes
- 4) Créer un fichier de log appelé `check_bad_programs.log` qui indique en niveau info qu'il n'y a rien à signaler lorsqu'aucun processus est tué, et qui signale en niveau warning le nom du processus qui a été tué s'il y en a un (utiliser la librairie `logging`)  
On veut veiller à tout moment qu'il reste suffisamment d'espace disque sur la machine.
- 5) Chercher une fonction dans la librairie `psutil` qui permet de récupérer le pourcentage d'espace disque occupé sur le disque principal.

**Toujours dans le même fichier python :** écrire le code qui gère cela :

- Lorsque l'espace disque est supérieur à 70% et inférieur ou égal à 80%, logger en warning que l'espace maximal est presque atteint
- Lorsque l'espace disque est strictement supérieur à 80%, logger en critical que l'espace maximal est atteint