Automatiser l'administration Unix/Linux avec les scripts Shell

2 - Redirections, assemblages

- **2.1** - Créer un fichier `hello.txt` qui contient `"Hello!"` , à l'aide la commande `echo` et d'une redirection.
- **2.2** - À l'aide d'une deuxième commande `echo` et d'une autre redirection, ajoutez **à la suite** (sur une nouvelle ligne) le mot `World!` dans `hello.txt` .
- **2.3** - Stockez la sortie de `ls /usr/bin` dans un fichier, et observez ce fichier avec la commande `less` . Observez-vous une différence entre le format de ce fichier, et la sortie de `ls /usr/bin` lorsqu'elle est affichée directement dans le terminal ?
- **2.4** - Lancez le script `bash fibonnaci_forever.sh` en redirigeant sa sortie vers un fichier. La commande tournera jusqu'à ce qu'elle soit interrompue : attendez donc quelques secondes puis appuyez sur Ctrl+C pour l'interrompre. Inspectez le contenu du fichier ensuite.
- **2.4** - `bc` est un utilitaire permettant de faire de petit calculs. Testez `bc` en mode interactif pour faire quelques additions (Ctrl+D pour quitter). Mettez maintenant une suite de calcul dans un fichier (par exemple, `2+2` , `6*7` , `10/3`), que vous injecterez directement dans `bc` .
- **2.5** - Même question que précedemment, mais en injectant directement une chaine contenant un calcul dans `bc` (sans passer par un fichier).
- **2.6** - Écrivez **une seule ligne de commande** (qui comportera plusieurs sous-commandes séparées par `;` , mais sans utiliser `cd` !) qui :
  - créer le dossier `~/formation_linux/calculs`
  - créer un fichier `~/formation_linux/calculs/formule` contenant `6*7`
  - effectue le calcul contenu dans le fichier `formule` , et stocke le résultat dans `~/formation_linux/calculs/reponse`
- **2.7** - `curl` est un utilitaire qui permet de telecharger et le code source d'une page (par exemple, `fr.wikipedia.org`) dans la console.
  - Comment pouvez-vous faire pour sauvegarder cette page dans un fichier ?
  - Retentez l'expérience avec une adresse qui n'existe pas, et modifiez votre commande pour supprimer les erreurs mais afficher tout de même "Ca n'a pas marché" dans le cas où vous tentez de télécharger une page qui n'existe pas...
- 2.8 - (Avancé) Créez un fichier `/tmp/chat` dans lequel `padawan` et `r2d2` peuvent tous les deux écrire. Renseignez-vous sur l'option `-f` de la commande `tail` puis :
  - Lancez `tail -f /tmp/chat` en tâche de fond dans deux terminaux (l'un de `padawan` , l'autre de `r2d2`) ;
  - À l'aide de redirections, envoyez des messages dans le fichier `/tmp/chat` et observez les deux terminaux ;

- Améliorez le système en créant un alias `say` qui affiche un message préfixé de votre nom d'utilisateur (ex: `[r2d2] beep boop`).

3 - Pipes et boîte à outils

- 3.1 - Si ce n'est pas deja le cas, ajoutez un alias pour activer automatiquement `--color=auto` chaque fois que la commande `grep` est utilisée. Essayez quelques manipulations avec `grep` pour confirmer que les occurences trouvées sont bien mises en valeur.
- **3.2** - Lister les lignes de `/etc/passwd` qui correspondent aux utilisateurs ayant `/bin/bash` comme shell
- **3.3** - Même chose, mais cette fois en affichant uniquement le nom des utilisateurs
- **3.4** - Lister les utilisateurs (uniquement leur nom !) qui ont comme shell `nologin`
- **3.5** - Lister les utilisateurs (uniquement leur nom) qui ont un mot de passe non vide (rappel : historiquement les passwords se trouvaient dans `/etc/passwd` , mais ce n'est plus le cas de nos jours !)
- **3.6** - Écrivez un alias `estcequecestleweekend` qui vérifie si on est Samedi ou Dimanche en utilisant `date`
- 3.7 - Sachant que pour grep, `^` et `$` désignent un début et une fin de ligne, pouvez-vous affichez le contenu de `/etc/login.defs`
  - sans les commentaires (ce sont les lignes commençant par #)
  - puis sans les commentaires ni les lignes vides ? (Indice : une ligne vide est une ligne qui se commnence puis se termine tout de suite)
- 3.8 - Modifiez votre alias `testinternet` pour qu'il affiche "Connecté!" ou "Pas de connexion!" suivant si le ping fonctionne (en masquant la sortie initiale du ping).
- **3.9** - À l'aide des pages de man de `grep` , trouvez un moyen de lister toutes les occurences du mot `daemon` dans tous les fichiers à l'intérieur de `/etc/` (recursivement)
- **3.10** - À l'aide de `ps` , `sort` et `uniq` générer un bilan du nombre de processus actuellement en cours par utilisateur
- **3.11** - À l'aide de `sort` et `uniq` , analysez le fichier `loginattempts.log` (demander au formateur comment l'obtenir), et produisez un résumé du nombre de tentative de connections par ip

5 - Variables

- **5.1** - Écrivez un premier script `hello.sh` qui affiche "Hello world !" et "How are you today ?" et executez-le.
- **5.2** - Dans des variables différentes, récupérez des informations comme :
  - le nom de la distribution (via `/etc/os-release`)
  - le nombre de processus actuellement lancés sur tout le système
  - la quantité de RAM totale et utilisée (voir `free -h`)
  - (bonus) l'uptime du système, l'IP locale, et l'IP globale du système

```
La distribution est Debian GNU/Linux 10 (buster)
Il y a 74 processus lancés actuellement
Il y a 340MB de RAM libre sur un total de 4GB
```

- 5.3 - Relancer le script avec `bash -x votrescript.sh` et analyser ce qui s'affiche : cette commande est utile pour débugger ce qu'il se passe en détail pendant l'exécution !
- 5.4 - (Bonus) En utilisant les codes couleurs de Bash, définissez des variables de couleur comme `PURPLE` qui vaut `\033[35m` . Modifier le code de la question 5.1 pour affichez "How are you today ?" avec les couleurs de l'arc-en-ciel ! (Attention : si il y a des couleurs, `echo` doit être appelé avec l'option `-e` pour interpréter les codes couleurs)

6 - Paramétrabilité, interactivité

- 6.1 - Écrivez un script `test_args.sh` qui affiche le nom du script, le nombre d'argument donnés, ainsi que le premier et deuxième argument. Testez et validez en lançant par exemple `test_args.sh titi grosminet` et `test_args.sh pikachu carapuce`
- 6.2 - Écrivez un script `add.sh` qui **prends deux nombre en argument** en affiche le résultat de leur addition.
- 6.3 - Écrivez un script `age.sh` qui **demande interactivement** à l'utilisateur son année de naissance, puis calcule et affiche son âge.
- **6.4** - Créez un script `check_user.sh` . Il est destiné à être lancé par `root` pour s'assurer qu'un utilisateur donné ne fait pas n'importe quoi.
  - Le script **prend en argument** un nom d'utilisateur ;
  - Calculer l'espace disque occupé par le repertoire personnel de l'utilisateur (aidez-vous de `du -hs <repertoire>`). Afficher un message comme "Son repertoire personnel pèse X Mo".
  - Compter le nombre de processus actuellement lancés par cet utilisateur (aidez-vous de `ps au -u <user>`) et afficher un message comme "L'utilisateur a X procesus en cours" ;

7 - Conditions

- **7.1** - Reprendre le script `age.sh` de l'énoncé 6.3. Vérifiez que l'année de naissance fait sens. Par exemple, si elle est inférieure à 1900, on pourra afficher "Hmm... T'es sur !?", et si l'année donnée est supérieure à 2025, on pourra afficher "Tu viens du futur !?". Dans les deux cas, on quittera le programme **en renvoyant un code d'erreur** (= code de retour différent de 0).
- 7.2 - Écrire un script `estcequecestbientotleweekend.sh` de sorte à ce que :
  - le script affiche "non :(" si le jour est lundi, mardi, mercredi ou jeudi
  - le script affiche "bientôt, mais il faut travailler encore un peu!" si on est vendredi mais qu'il est moins que 17h
  - le script affiche "oui c'est le weekend!" autrement

    - pour ce faire : on pourra récupérer le jour et l'heure actuelle à partir de `date`, et stocker ces valeurs dans deux variables `JOUR` et `HEURE`, puis utiliser des comparaisons.
- 7.2 - En reprenant `check_user.sh` :
    - ajouter un test que l'utilisateur existe en vérifiant qu'il y a bien une ligne correspondante dans `/etc/passwd` (ou un autre moyen de votre choix), sinon afficher un message d'erreur (dans stderr !) et arrêter le script avec `exit` en renvoyant le code de retour `1` ;
    - adaptez le comportement du script de sorte à ce que `./check_user.sh --help` (ou `-h`) affichera (au lieu du fonctionnement normal de la fonction) une courte description de ce que fait le script et de comment l'utiliser.
    - ajoutez des tests de sorte que le script râle si il se trouve que le répertoire personnel dépasse une certaine taille
    - (bonus) ajoutez des tests de sorte que le script râle si il se trouve que l'utilisateur a lancé plus que 50 processus
    - (bonus) ajoutez un test qui vérifie que l'user est bien propriétaire de son dossier personnel

8 - Fonctions

- 8.1 - Comme en 5.4, créer un script `utils.sh` qui défini des variables correspondant aux couleurs. À l'aide de celles-ci, implémentez et testez au fur à mesure les fonctions suivantes :
    - `success` qui prends un message en argument et affiche `"[ OK ] Le message"` avec le mot `OK` en vert ;
    - `info` qui prends un message en argument et affiche `"[INFO] Le message"` avec le mot `INFO` en bleu ;
    - `warning` qui prends un message en argument et affiche *dans la sortie d'erreur* `"[WARN] Le message"` avec le mot `WARN` en orange ;
    - `error` qui prends un message en argument et affiche *dans la sortie d'erreur* `"[FAIL] Le message"` avec le mot `FAIL` en rouge ;
    - `critical` qui prends un message en argument et affiche *dans la sortie d'erreur* `"[CRIT] Le message"` avec le mot `CRIT` en rouge, **et termine directement le script avec un code de retour de 1**.
- 8.2 - Dupliquez le script `check_user.sh` de la question 7.2 (la copie peut s'appeler `check_user_v2.sh`), puis transformez les différentes parties du script en fonctions. Ce genre de travail s'appelle *(re)factoriser du code*. **Testez vos modifications au fur et à mesure !**
    - au début (après le `#!/bin/bash`), ajoutez `source utils.sh` pour charger les fonctions définies dans `utils.sh` dans ce script.
    - introduisez une fonction `assert_user_exists` qui affichera une erreur et arrêtera le script (via `critical`) si l'utilisateur n'existe pas ;
    - introduisez une fonction `usage` qui sera appelée pour décrire le fonctionnement du script si appelé avec `--help` ou `-h` ;

    - introduisez une fonction `check_home_usage` qui vérifiera que la taille du home de l'utilisateur ne dépasse pas un certain seuil. Si le nombre n'est pas trop grand, affichez le nombre avec `info` - ou bien avec `warn` sinon.
    - de façon similaire, une fonction `check_terminals`
    - de façom similaire, une fonction `check_processes`
- 8.3 - Finalement, adaptez le script pour qu'il affiche à la fin que tout va bien avec `success` si aucun problème n'a été détecté, ou `fail` pour signifier que l'utilisateur a dépassé les bornes des limites !

9 - Boucles

- 9.1 - Écrivez à l'aide d'une boucle `for` un script qui affiche la table de multiplication d'un entier `N` passé en argument du script:

```
Table du 7
----------
 1 x 7 = 7
 2 x 7 = 14
 [...]
 10 x 7 = 70
```

- 9.2 - Même chose, mais à l'aide d'une boucle `while` .
- 9.3 - En reprenant le script `check_user.sh` , utilisez une boucle `for` pour lancer ce script sur tous les utilisateurs ayant `/bin/bash` comme shell.
- 9.4 - Écrivez un script `confirm.sh` qui, à l'aide d'une boucle `while` , demande à l'utilisateur `Est-ce que tu es sur ? [oui/non]` tant qu'il n'a pas répondu `oui` ou `non` .
- 9.5 - Revenant d'un mariage dans votre famille, vous décidez de partager vos 100 photos avec le reste de votre famille. Sachant que vos appareil photo dernier-cri prends des photos en moulte-méga-pixels qui pèsent 10 Mo chacune, vous décidez qu'il faut mieux redimensionner les images avant de les envoyer pour ne pas surcharger les boîtes mails de tout le monde avec 1Go de photos. Pour ce faire, renseignez-vous sur la commande `convert` (du paquet ImageMagick) qui permet de redimensionner (ou *rescaler*) et éventuellement d'adapter le niveau de qualité. Sachant maintenant manipuler cette commande, écrivez une boucle `for` pour redimensionner d'un seul coup toutes les images `jpg` présentent dans le dossier courant. Tester ce script sur le pack d'image fourni par le formateur.
- 9.6 - Ecrire en bash un jeu qui consiste à demander à l'utilisateur de deviner un nombre choisi aléatoirement par le programme. L'utilisateur entrera un nombre, et le programme dira à l'utilisateur si le nombre à trouver est plus petit ou plus grand, jusqu'à ce que le bon nombre soit trouvé.

Jeu des allumettes

Avec toutes les notions de variables, fonctions et boucles désormais découvertes, il est possible de créer un petit jeu dans un script Bash.

Le jeu des allumettes est un jeu pour deux joueurs, où n allumettes sont disposées, et chaque joueur peut prendre à tour de rôle 1, 2 ou 3 allumettes. Le perdant est celui qui se retrouve obligé de prendre la dernière allumette.

- Écrire une fonction `afficher_allumettes` capable d'afficher un nombre donné d'allumettes (donné en argument), par exemple avec le caractère `|` :

```
Il reste 27 allumettes.
|||||||||||||||||||||||||||
```

- Écrire une fonction `choisir_nombre` qui demande à l'utilisateur combien d'allumette il veut prendre. Cette fonction vérifiera que le choix est valide (soit 1, 2 ou 3).

```
Combien prendre d'allumettes ?  [Attendre et enregister la réponse de l'utilisateur]
```

- Écrire la fonction principale `main` qui :
  - Initialise le nombre d'allumette sur la table
  - Afficher des allumettes avec `afficher_allumettes`
  - Demander à l'utilisateur combien il veut prendre d'allumettes avec `choisir_nombre`
  - Propager ce choix sur le nombre d'allumette actuellement sur la table
  - Afficher le nouvel état avec `afficher_allumettes`
  - etc... *tant que* il reste au moins une allumette
- Modifier le programme précédent pour gérer deux joueurs (1 et 2) et les faire jouer à tour de rôle jusqu'à ce qu'une condition de victoire soit détectée (= il reste moins d'une allumette).

11 - Expressions régulières

- Rendez-vous sur `regex101.com` . Rentrez les exemples fournis par le formateur et tentez de construire une regex qui match seulement les bonnes occurences.

12 - Tâches automatiques avec `cron` et `at`

- 12.1 - Utiliser `at` (via un script) pour écrire, au bout de deux minutes, les mots "C'est automagique !" dans un fichier `magic.txt` . Dans les secondes avant que la tâche ne se déclenche, regardez le contenu du dossier `/var/spool/atjobs/` .
- 12.2 - Ajoutez à l'aide de `date` et `crontab -e` un cron "personnel" qui met toutes les 5 secondes la date qu'il est dans un fichier "date" dans votre répertoire personnel.
- 12.3 - À l'aide de `wget` ajoutez un cron job dans `/etc/cron.d/` qui télécharge dans votre répertoire personnel chaque minute une image de chat différente

depuis `https://cataas.com/cat`. Vous pouvez utiliser l'option `-O <nom_de_fichier>` de `wget` qui permet de choisir le nom de fichier de destination.

- 12.4 - Installez le paquet "logrotate" à l'aide de `apt`, puis regardez des les dossiers `/etc/cron.*` pour remarquez l'existence d'un script `logrotate`. Renseignez-vous sur l'utilité et la raison d'être de ce script/programme.