Scripting Bash

1 - les variables

- **1.1** - Créez-vous un répertoire de travail (par exemple : `~/formationlinux/bash`) dans lequel vous créerez vos scripts. Écrivez un premier script `hello.sh` qui affiche "Hello world !" et "How are you today ?" et executez-le.
- **1.2** - 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
```

- 1.3 - 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 !
- 1.4 - En reprenant les codes couleurs étudiés en partie 8, définissez des variables de couleur comme `PURPLE` qui vaut `\033[35m` . Modifier le code de la question 1.1 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)

2 - paramétrabilité, interactivité

- 2.1 - 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`
- 2.2 - Écrivez un script `add.sh` qui **prends deux nombre en argument** en affiche le résultat de leur addition.
- 2.3 - Écrivez un script `age.sh` qui **demande interactivement** à l'utilisateur son année de naissance, puis calcule et affiche son âge.
- **2.4** - 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. Implémentez les étapes suivantes une par une, en les testant au fur et à mesure :
  - 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".
  - (Bonus) 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" ;
  - (Bonus) Même chose, mais avec le nombre de terminaux actuellement utilisés par l'utilisateur ;

3 - les conditions

- **3.1** - Reprendre le script `age.sh` de l'énoncé 2.2. 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 2020, 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).
- 3.2 - 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.
- 3.2 - 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

4 - les fonctions

- 4.1 - Comme en 1.4, 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**.
- 4.2 - Dupliquez le script `check_user.sh` de la question 3.2 (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`
- 4.3 - 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 !

5 - les boucles

- 5.1 - Écrivez à l'aide d'une boucle `for` un script qui affiche les carrés des 50 premiers entiers
- 5.2 - Même chose, mais à l'aide d'une boucle `while` .
- 5.3 - En reprenant le script `check_user.sh` , utilisez une boucle `for` pour lancer ce script sur tous les utilisateurs ayant `/bin/bash` comme shell.
- 5.4 - 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` .
- 5.5 - 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.
- 5.6 - 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. (Pour tester votre script, vous pouvez télécharger par exemple des images sur `pixabay.com` , ou demander un pack d'image au formateur)

6 - tâches automatiques avec `cron` et `at`

- 6.1 - 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/` .
- 6.2 - Sur votre serveur, dans un des dossiers `/etc/cron.*` , remarquez l'existence d'un script `logrotate` . Renseignez-vous sur l'utilité et la raison d'être de ce script/programme.
- 6.3 - À l'aide de `wget` , ajoutez un cron job qui télécharge dans votre HOME chaque minute une image de chaton différente depuis `https://placekitten.com/` (on pourra utiliser la variable `$RANDOM` pour obtenir un entier aléatoire)
- 6.4 - Sur votre serveur, téléchargez quelques images de chatons (ou autre) dans un nouveau dossier `img` dans `/var/www/mywebsite` . Créez un script (non automatisé pour le moment) qui permet de choisir une image aléatoire parmis toute celles disponible, et utilise cette image sur votre page d'acceuil. Pour ce faire, :
    - vous pouvez choisir un nombre aléatoirement avec la variable spéciale `$RANDOM` ,
    - mettre en place un fichier `template.html` qui contient le code de votre page web mais où le chemin de l'image est un mot clef comme `CHEMIN_IMAGE` . Votre script pourra alors utiliser `sed` pour généner `index.html` en remplacant `CHEMIN_IMAGE` par le vrai chemin.
- 6.5 - Une fois votre script validé, ajouter un job cron qui lancera ce script toutes les minutes pour choisir une nouvelle image aléatoirement

- 6.6 - Même principe, mais cette fois créez une page `monitor.html` qui contiendra des informations régulièrement mises à jour (e.g. toutes les quelques minutes) à propos du serveur : RAM utilisée / disponible, uptime, processus les plus gourmands, l'heure qu'il est, et un rapport des IPs récemment bannies par fail2ban ...