TP 3 - Premiers scripts `bash`

À partir de cette séance, nous allons écrire des scripts `bash` que nous voudrons réutiliser. Pour pouvoir les utiliser n'importe où sur notre système, nous allons les placer dans un répertoire `~/bin` que nous rendrons accessible à tout moment grâce à la variable `PATH`. Pour ce faire, entrez les commandes suivantes:

```
$ mkdir ~/bin                                          # créer le répertoire
$ echo -e export PATH=$HOME/bin:\$PATH >> ~/.bashrc  # le \ est important!
$ source ~/.bashrc                                     # appliquer les changements
```

Une fois ces commandes exécutées, les scripts de ce répertoire seront accessibles partout. Un script nommé `exemple.sh` se lancera simplement en tapant `exemple.sh` (et **pas** `./exemple.sh` comme vu au cours).

Fonctions

À partir de cette séance, nous allons modifier des fichiers de configuration. En cas d'erreur, votre système pourrait ne plus fonctionner correctement. Pour nous faciliter les choses, nous allons créer une fonction de sauvegarde.

Exercice 1. Écrivez une fonction `sauvegarder`, qui prend en paramètre un chemin vers un fichier ou un répertoire et en crée une copie dont le nom se termine par `.BAK.` suivi de la date et de l'heure actuelle. Par exemple:

```
$ ls -l
total 4
-rw-r--r-- 1 anthony anthony    0 nov  7 01:39 fichier
drwxr-xr-x 2 anthony anthony 4096 nov  7 01:39 repertoire
$ sauvegarder fichier
$ sauvegarder repertoire
$ sauvegarder fichiermanquant
Erreur: le fichier ou répertoire fichiermanquant n'existe pas
$ ls -l
total 8
-rw-r--r-- 1 anthony anthony    0 nov  7 01:39 fichier
-rw-r--r-- 1 anthony anthony    0 nov  7 01:41
↪  fichier.BAK.2023-11-07T01:41:56+01:00
drwxr-xr-x 2 anthony anthony 4096 nov  7 01:39 repertoire
drwxr-xr-x 2 anthony anthony 4096 nov  7 01:39
↪  repertoire.BAK.2023-11-07T01:42:00+01:00
```

Scripts

Exercice 2. Pour gagner du temps dans l'écriture des nouveaux scripts, écrivez un script `nouveau_script.sh` créant un script prêt à l'emploi: un fichier vide et exécutable avec l'extension `.sh`, le bon entête, une description, le nom de l'auteur, et son adresse e-mail. Par exemple, la commande `./nouveau_script.sh essai.sh` créera le fichier exécutable `essai.sh` contenant:

```
#!/usr/bin/bash
#
# Description du script
#
# Utilisation:
#
# Copyright (C) 2022
#
# Auteurs: Anthony Labarre <anthony@debian>
```

Rendez votre script le plus générique possible à l'aide des variables d'environnement: ainsi, si vous envoyez `nouveau_script.sh` à un ami, le script utilisera automatiquement les coordonnées de votre ami. L'adresse e-mail sera au format "`nom d'utilisateur@nom de la machine`".

1. Une fois que votre script fonctionne, modifiez-le pour qu'il ne fasse rien et prévienne l'utilisateur d'une erreur si le nom du fichier à créer est vide. On peut vérifier si une chaîne est vide avec `if [[ -z $chaine ]]`.

2. Ensuite, modifiez-le pour qu'il rajoute un `.sh` à la fin du nom du fichier si l'utilisateur l'a oublié. On peut vérifier si `chaine` se termine par `suffixe` avec le test `if [[ chaine == *$suffixe ]]`.

À partir de l'exercice suivant, il sera parfois nécessaire de demander à l'utilisateur de rentrer des informations. On peut les stocker dans une variable avec la commande `read`. Par exemple:

```
$ read ma_variable  # l'utilisateur doit rentrer une valeur
bonjour
$ echo $ma_variable
bonjour
```

Il n'est pas nécessaire de déclarer `ma_variable` avant d'utiliser `read`.

Exercice 3 (Installation)**.** Écrivez un script `installer.sh` qui prend en paramètre une chaîne et installe automatiquement le paquet contenant le fichier correspondant. Par exemple:

```
$ ./installer.sh gentoopenguin.png
Recherche en cours, un instant...
Il y a exactement un paquet contenant ce nom: gnome-devel-docs
Voulez-vous l'installer? [o/n]
```

L'installation sera lancée si l'utilisateur tape `'o'`, ou annulée s'il tape `'n'`. Si le fichier est introuvable, ou s'il y a plus d'un paquet correspondant, on ne fait rien. Par exemple:

```
$ ./installer.sh jenexistepas
Recherche en cours, un instant...
Aucun résultat trouvé; utilisez sudo apt-file update si nécessaire.
$ ./installer.sh beamer.sty
Recherche en cours, un instant...
4 résultats trouvés, c'est trop pour moi.
```

Exercice 4 (Installation avec choix multiples)**.** Modifiez votre script de l'exercice 3 pour obtenir un script `installer_multi.sh` qui gère les cas où plusieurs paquets correspondent. Par exemple:

```
$ ./installer_multi.sh beamer.sty
Recherche en cours, un instant...
4 résultats trouvés:
    [0] texlive-latex-extra
    [1] texlive-pictures
    [2] texlive-publishers
    [3] texlive-xetex
Entrez le numéro du paquet à installer (ou n pour annuler):
```

On triera les paquets dans l'ordre alphabétique.

Internal Field Separator (IFS)

Cheatsheet 1: `IFS`

`IFS` (pour ***i**nternal **f**ield **s**eparator*) est une variable d'environnement de `bash` précisant les caractères à utiliser pour délimiter les mots. Par défaut, cette variable correspond aux espaces, tabulations et retours à la ligne. Si l'on veut découper du texte ligne par ligne, en gardant les espaces, il faut modifier cette variable en précisant que seuls les retours à la ligne sont des séparateurs. Il suffit de réaliser l'affectation `IFS=$'\n'`. Ainsi, pour mettre toutes les lignes d'un fichier dans un tableau, on écrira:

```
$ IFS=$'\n'
$ readarray lignes < fichier.txt
```

Pour éviter les problèmes dans le terminal, on annule nos changements à l'aide de `unset`, qui restaurera la valeur par défaut d'`IFS`:

```
$ IFS=$'\n'
$ # ... du code ...
$ unset IFS
```

Pour illustrer le genre de problèmes que l'IFS peut créer, plaçons-nous dans un répertoire vide, créons-y le fichier `mon fichier.txt`, et comparons l'affichage direct du contenu du répertoire avec `ls`, ou en mettant le résultat de `ls` dans un tableau:

```
$ ls
'mon fichier.txt'
$ tableau=($(ls))
$ for ligne in ${tableau[@]}; do echo $ligne; done
mon         # = tableau[0]
fichier.txt # = tableau[1]
```

En indiquant que l'IFS est un retour à la ligne, on obtient le résultat correct: chaque élément du résultat de `ls` est un élément du tableau:

```
$ IFS=$'\n' # à faire AVANT de lancer la commande ls!
$ tableau=($(ls))
$ for ligne in ${tableau[@]}; do echo $ligne; done
mon fichier.txt # = tableau[0]
```

Pour les exercices suivants, vous aurez besoin de télécharger un fichier supplémentaire:

```
$ wget http://igm.univ-mlv.fr/~alabarre/teaching/shnu/linux/dates
```

Ce fichier `dates` contient une liste d'événements au format "MM-JJ: description", où MM représente un mois sur deux chiffres et JJ un jour sur deux chiffres. Attention, les données du fichier `dates` ne sont pas nécessairement triées.

Exercice 5 (IFS: échauffement)**.** Écrivez un script qui enregistre dans un tableau toutes les lignes commençant par une date donnée en paramètre, au format MM-JJ, et qui ensuite les affiche. Par exemple:

```
$ ./dates_ifs.sh 11-11
Il y a 2 événements pour le 11-11:
        11-11: Armistice première guerre mondiale
        11-11: Fête des célibataires (Chine)
```

Exercice 6. Écrivez un script qui affiche les événements du jour à l'utilisateur sur base du fichier `dates`. On veut afficher tous les événements qui correspondent à la date d'aujourd'hui. Par exemple:

```
$ grep 10-01 dates
10-01: Fête nationale chinoise
$ ./evenements.sh
La date d'aujourd'hui est ven 1 oct 2021 12:40:06 CEST

Voici les événements du jour:

* Fête nationale chinoise
```

Exercice 7. Écrivez un script `sous_reps_vides.sh` qui renvoie les chemins complets vers les sous-répertoires vides du répertoire donné en paramètre. La commande `${chaine::-1}` peut vous être utile: elle renvoie le contenu de `chaine` sans son dernier caractère (comme `chaine[:-1]` en Python). Pour tester votre script, vous pouvez créer un répertoire `essai` contenant des répertoires vides. Par exemple:

```
$ mkdir -p essai/pas_vide/vide_dans_pas_vide essai/vide "essai/vide avec espaces"
```

Le résultat sera alors:

```
$ ./sous_reps_vides.sh essai
essai/pas_vide/vide_dans_pas_vide
essai/vide
essai/vide avec espaces
```

Fonctions avec tableaux

Exercice 8 (Tous les couples différents)**.** Écrivez une fonction `tous_couples_differents` qui prend en paramètre un tableau de chaînes et affiche tous les couples différents de valeurs de ce tableau. Par exemple:

```
$ mon_tableau=("bonjour" "gnu" "linux")
$ tous_couples_differents "${mon_tableau[@]}"
bonjour gnu
bonjour linux
gnu linux
```

On ne peut obtenir de doublon dans le résultat que si le tableau contient plusieurs fois le même élément.
Attention: si vous décidez d'affecter un nom à votre variable dans la fonction pour que le code soit plus lisible, il faut utiliser la syntaxe vue au cours pour récupérer tous les éléments du tableau:

```
ma_variable=("$@")  # dans la fonction, pour utiliser ma_variable au lieu de $1
```