Mémoire technique

Utilitaire d'archivage

Justin Bossard Antoine Feuillette

Octobre 2024

Table des matières

1 Présentation du projet

Ce projet consiste à programmer un utilitaire d'archivage, qui télécharge une archive au format ZIP depuis un serveur Web. Cette archive contient un dump SQL, qui doit être à la racine et l'unique fichier SQL.

Cet utilitaire est prévu pour s'exécuter quotidiennement et déterminer si le dump SQL a subi des changements par rapport à la veille. Si c'est le cas, il faut archiver le dump sur un serveur distant via le protocle SFTP, dans une archive TGZ, au format `AAAADDMM.tgz`.

Il faut également établir un suivi par mail, et un historique des opérations dans un fichier de journalisation.

2 Justification des choix techniques

2.1 Choix du langage de script

Nous avons choisi d'écrire le script en bash, car il s'agit d'un langage de script clair et concis. De plus, il est parfaitement intégré à la plupart des distributions GNU/Linux (en tant que shell par défaut), s'intègre facilement à d'autres distributions Unix (MacOS, *BSD), et peut s'intégrer à Windows[1].

De plus, bash est plus léger que Python, qui est un langage multi-paradigme. Pour notre cas d'usage, on a uniquement besoin des capacités de scripting du langage, les possibilités de Python sont donc superflues.

Enfin, bash permet d'interagir directement avec les commandes du système, et ne nécessite pas de bibliothèque externe, comme c'est le cas sur Python.

Ainsi, le choix de bash est plus avantageux en termes :

- de lisiblité
- de limitations des dépendances
- d'adaptation au projet

2.2 Test pour des changements dans le dump SQL

Plusieurs manières sont envisageables pour détecter des changements dans un fichier. La manière la plus sûre consiste à conserver une copie du fichier et à comparer octet par octet avec un autre fichier. Néanmoins, cela pose deux problèmes principaux : tout d'abord, si les fichiers sont grands, cela peut prendre un temps non négligeable, ensuite, il est nécessaire de stocker constamment un fichier inutile autrement.

Nous avons donc décidé de procéder par calcul d'une somme de contrôle du dump, que l'on stocke dans le fichier `.prevChecksum`, et qui permet un suivi abstrait des modifications. Cela a plusieurs avantages : la comparaison prend toujours le même temps, l'espace de stockage nécessaire est négligeable, et cela permet d'avoir un historique en inscrivant la somme de contrôle dans les logs. De cette manière, s'il y a besoin d'utiliser des archives par la suite, on pourra vérifier l'intégrité de la sauvegarde avec sa somme de contrôle, en regardant dans l'historique des logs.

Nous utilisons la fonction `sha256sum`, de GNU coreutils, qui permet de calculer la somme de contrôle sur 256 bits d'un fichier.

[1] https://korben.info/installer-shell-bash-linux-windows-10.html

Il est amplement suffisant et d'usage d'utiliser 256 bits pour ce genre de situation. Pour limiter au maximum le risque de collision on pourrait la calculer sur 512 bits, néanmoins pour des fichiers de taille importante ce calcul prendrait plus de temps et de ressources.

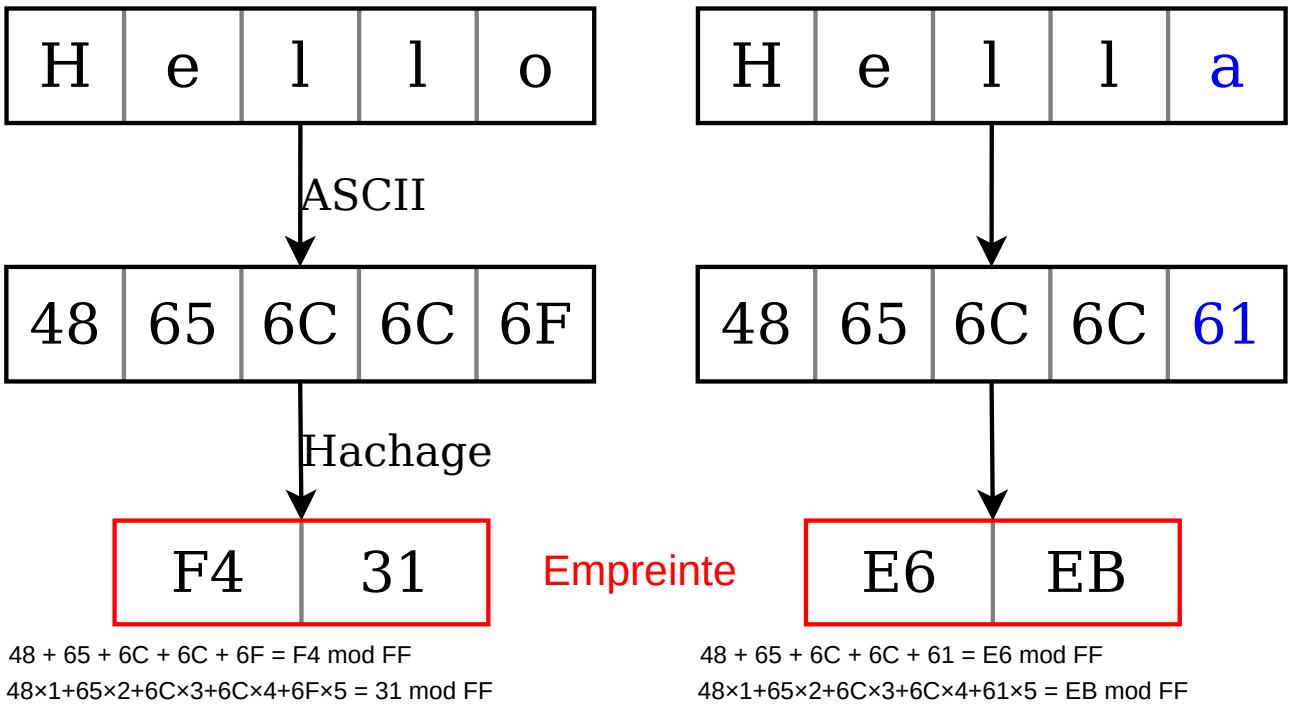

Figure 1: Exemple de calcul d'une checksum

2.3 Serveurs

Pour tester les différentes fonctionnalités, nous avons mis en place deux serveurs sur une Raspberry Pi 4.

2.3.1 Web

Pour pouvoir télécharger l'archive contenant le dump SQL à tout instant, nous avons mis en place un serveur Web avec Apache, accessible via HTTP (port 80) ou HTTPS (port 443).

Pour cela, il suffit d'installer Apache et de le démarrer automatiquement. Ensuite, on place l'archive d'intérêt dans `/var/www/html` et on y accède avec l'adresse IP (e.g. : `http(s)://176.190.35.242/sql_dump.zip`).

2.3.2 SSH

Le serveur d'archivage doit être accessible via SSH, nous avons donc installé OpenSSH sur le serveur. Il suffit ensuite de démarrer le démon SSH automatiquement puis d'accéder à la machine par le port 22 (par défaut).

Nous avons également mis en place un système de clés RSA, pour pouvoir se connecter au serveur sans avoir besoin du mot de passe :

- **Clé privée** Sur l'ordinateur client *uniquement*.
- **Clé publique** Sur l'ordinateur client et sur le serveur.

Voir la *Documentation utilisateur* pour générer et utiliser cette paire de clés.

2.3.2.1 Suppression automatique des archives trop anciennes

Pour supprimer automatiquement les archives antérieures à une durée paramétrable par l'utilisateur (voir plus loin), nous utilisons la commande `find`, de GNU findutils. La commande complète (depuis l'ordinateur client) est la suivante :

```
ssh "$usernameSSH@$adresseArchivage" "find $pathSSH -name *.tgz -type f -ctime
↪  +$dureeConservation -exec rm '{}' \; && exit"
```

Tout d'abord, on se connecte au serveur d'archivage, puis on cherche tous les fichiers (`-type f`) qui sont des archives `tgz` (`-name *.tgz`) dans le dossier des sauvegardes (`$pathSSH`), et qui ont été créée (`-ctime`) *au moins* il y a un certain nombre de jours, paramétrable (`+$dureeConservation`).

On supprime ensuite tous les fichiers trouvés (`-exec rm '{}' \;`).

Si l'opération est un succès, on ferme la connexion (`&& exit`).

2.4 Envoi de mails

Pour effectuer l'envoi des mails, on utilise `mutt`, un logiciel libre permettant de se connecter simplement à un serveur SMTP.

Pour tester cette fonctionnalité, et dans la mesure où la plupart des fournisseurs mails ont compliqué l'accès SMTP à des applications comme `mutt` (Google, Microsoft, Yahoo[2]...), nous n'avons trouvé que qu'un fournisseur permettant d'accéder facilement et *gratuitement* à leurs serveurs par SMTP : Zoho Mail.

Nous avons donc créer une adresse Zoho, et l'envoi de mails et de pièces a été concluant, en fonction de la configuration renseignée (pièce jointe systématique, en cas d'échec...).

2.5 Automatisation

Pour exécuter le script tous les jours à 4 h 00, avec un [cron] déjà configuré, après `crontab -e` (depuis un utilisateur qui a les droits nécessaires pour exécuter le script), ajouter à la fin du fichier ouvert :

```
0 4 * * * /path/to/archive.sh
```

Avec `/path/to/archive.sh` le chemin vers le script.

Attention, avec une implémentation suivant les spécifications par défaut de [cron], si l'ordinateur est éteint au moment voulu d'exécution, le script ne sera pas exécuté au redémarrage.

Pour avoir ce comportement, on peut utiliser fcron, avec le `crontab` suivant :

```
&bootrun(true) 0 4 * * * /path/to/archive.sh
```

3 Organisation

On a l'organisation de fichier suivante :

```
.
|-  archive.sh      # Script
```

[2] Fonctionne théoriquement, mais nous n'avons pas réussi, et certains posts laissent entendre que cette fonctionnalité est régulièrement désactivée.

```
|-  archive.conf    # Fichier de configuration
|-  archive.log     # Logs du script, possibilité de modifier l'emplacement
|-  .prevChecksum   # Somme de contrôle du précédent fichier
```

3.1 Script (`archive.sh`)

Pour simplifier la lecture et l'écriture du programme, nous avons écrit une fonction pour écrire les logs, une fonction pour envoyer un mail, et une fonction qui combine les deux.

3.1.1 Convention d'arguments des fonctions

Nous avons établi la convention d'arguments suivantes pour les fonctions nécessaires :

$1 Si l'opération est un succès, 0.
Si l'opération est un échec, 1.
$2 Si l'opération est un succès, checksum du fichier.
Si l'opération est un échec, motif de celui-ci.

3.1.2 Fonction d'écriture de logs

```
function ecrireLog() {
    if [[ $1 -eq 0 ]]; then
        echo "[ $(date +'%T - %d %b %Y') ] : Succès, checksum=$2" >>
        ↪  "$emplacementLog"
    else
        [[ $logStdout -eq 0 ]] && echo "$2"
        echo "[ $(date +'%T - %d %b %Y') ] : Échec, $2" >> "$emplacementLog"
    fi
}
```

On ajoute à la fin du fichier de logs la date et l'heure d'écriture, suivi des paramètres adéquats.

3.1.3 Fonction d'envoi de mails

```
function envoyerMail() {
    if [[ ${#mailDestinataires[*]} -ne 0 && (($1 -eq 1 && $envoyerMail -eq 1)
    ↪  || $envoyerMail -eq 2) ]]; then

        if [[ $muttrcUtilisateur -eq 0 ]]; then
            echo "$([[ $1 -eq 0 ]] && echo L\'opération de ce jour est un
            ↪  succès || echo $2)" | \
                mutt -x \
                     -s "$([[ $1 -eq 0 ]] && echo $objSucces || echo
                     ↪  $objEchec)" \
                     $([[ $joindreLog -eq 2 || ($1 -eq 1 && $joindreLog -eq 1)
                     ↪  ]] && echo "-a $emplacementLog --") \
                     "$(echo ${mailDestinataires[*]})"

        else
            echo "$([[ $1 -eq 0 ]] && echo L\'opération d\'archivage de ce
            ↪  jour est un succès. || echo $2)" | \
```

```
                mutt -nx \
                     -e "set from = \"$mailEnvoyeur\"" \
                     -e "set smtp_pass = \"$motDePasse\"" \
                     -e "set smtp_url =
                     ↪  \"smtps://$mailEnvoyeur@$serveurHote:$port\"" \
                     -e "set send_charset = \"utf-8\"" \
                     -s "$([[ $1 -eq 0 ]] && echo $objSucces || echo
                     ↪  $objEchec)" \
                     $([[ $joindreLog -eq 2 || ($1 -eq 1 && $joindreLog -eq 1)
                     ↪  ]] && echo "-a $emplacementLog --") \
                     "$(echo ${mailDestinataires[*]})"
        fi
    fi

    [[ $? -ne 0 ]] && ecrireLog 1 "erreur lors de l'envoi du mail."
}
```

Le booléen utilisé par le premier `if` permet d'envoyer un mail si la liste de destinataires du fichier de configuration n'est pas vide *et* qu'on est en situation d'envoyer un mail (échec et `envoyerMail=1`, ou `envoyerMail=2`).

On vérifie ensuite si l'utilisateur veut utiliser sa configuration de `mutt` ou non. Si ce n'est pas le cas, on précise que `mutt` ne doit pas utiliser le fichier de configuration de l'utilisateur (option `-n`), et on passe les paramètres nécessaires à l'envoi d'un mail en argument (option `-e "set option = \"valeur\""`) :

- **`from`** Permet de définir l'identité de l'envoyeur. On utilise l'adresse mail renseignée dans le fichier de configuration.
- **`smtp_pass`** Le mot de passe qui permet de se connecter au serveur SMTP.
- **`smtp_url`** Définit le serveur auquel se connecter et le protocole de connexion (`smtps` ici). On renseigne également le port de connexion définit dans la configuration.
- **`send_charset`** Définit l'encodage du corps du mail et des pièces jointes. On choisit `utf-8` pour pouvoir prendre en compte l'ensemble des caractères Unicode.

On définit le sujet du mail avec l'option `-s`, qui varie en fonction de l'aboutissement de l'opération (on utilise la syntaxe : `[[ succès ? ]] && oui || non`).

Ensuite, selon les préférences de l'utilisateur, on joint ou non le fichier de log. Pour cela, on vérifie s'il faut joindre systématiquement le log, ou s'il faut le joindre en cas d'échec et que c'en est un. Si la condition est vraie, on renvoie l'option `-a` avec le chemin de log suivi de `--`[3], sinon on ne renvoie rien.

Enfin, on affiche tous les éléments de la liste des destinataires.

3.1.4 Fonction de combinaison

```
function combo() {
    if [[ "$1" -eq 0 ]]; then
        ecrireLog 0 "$2"
```

[3]Voir le manuel de Mutt, en cas de pièce jointe il faut séparer le fichier des destinataires.

```
        envoyerMail 0
    else
        ecrireLog 1 "$2"
        envoyerMail 1 "L'opération d'archivage de ce jour a échoué pour le
        ↪  motif suivant : $2"
    fi
}
```

Cette fonction combine simplement les deux fonctions précédentes, et permet d'unifier les motifs d'arrêts de fonctions.

3.1.5 Gestion des erreurs

Pour gérer toutes les erreurs en bash, on utilise la syntaxe suivante :

```
if ! commande; then
    combo 1 "L'opération a échouée à cause de \`commande\`."
    exit 1
fi
```

Cela permet de gérer simplement toutes les erreurs.

3.1.6 Diagramme d'activité

Notre script a le diagramme d'activité suivant (voir le script complet) :

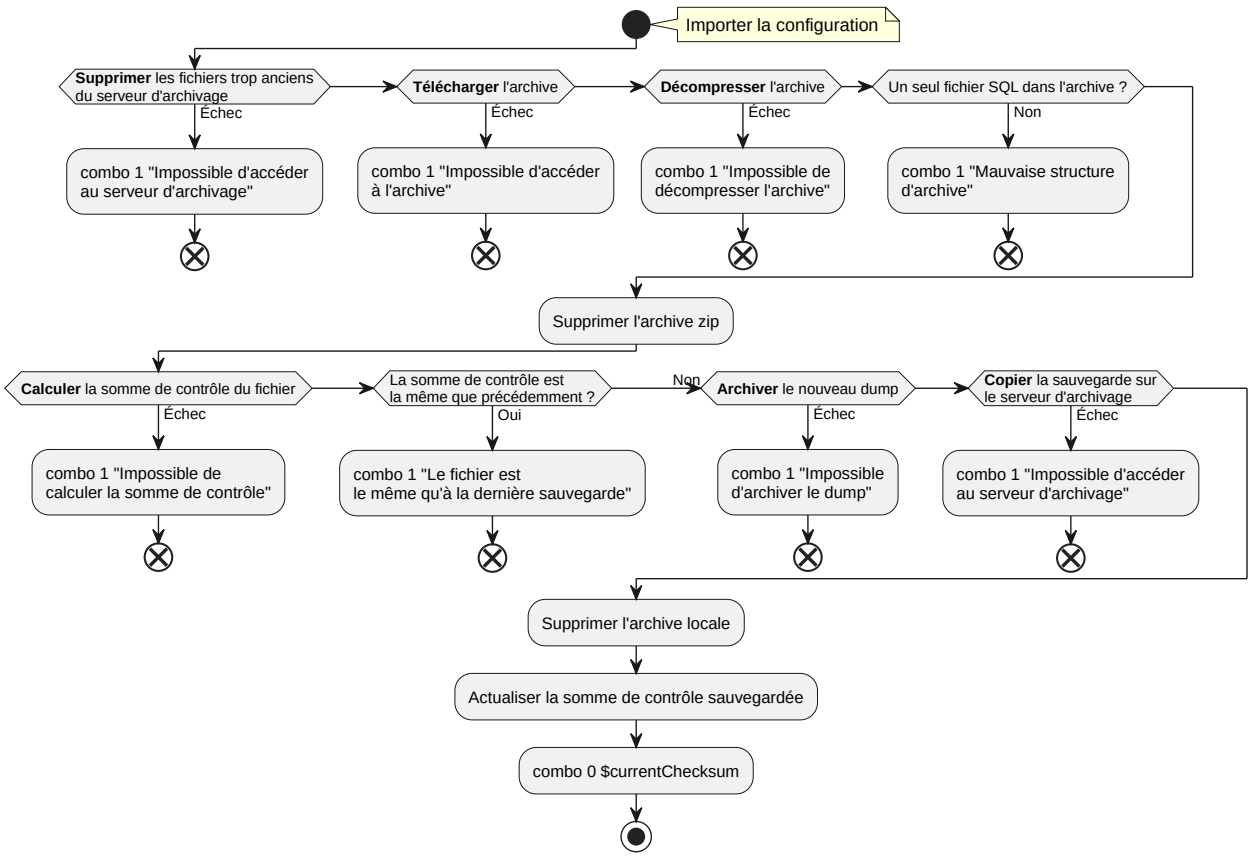

Figure 2: Diagramme d'activité

3.2 Fichier de configuration (`archive.conf`)

Le fichier de configuration est organisé en plusieurs sections :

1. Configuration générale
2. **Configuration du serveur d'archivage** Permet de configurer la connexion à un serveur SFTP via SSH.
3. **Configuration des mails** Permet de configurer les infos générales des mails : quand en envoyer, objets, destinataires...
   - Configuration serveur SMTP Permet de définir comment se connecter au serveur d'envoi SMTP spécialement pour ce script, si nécessaire.

3.3 Fichier de journaux (`archive.log`)

Le fichier de journalisation a la syntaxe suivante pour une ligne (en utilisant une version édulcorée de la syntaxe bash pour succès ou échec) :

```
[ heure - date ] : [[ Succès, checksum || Échec, cause de l'échec ]]
```

Une nouvelle entrée par ligne, en ordre décroissant d'ancienneté.

3.4 Fichier de somme de contrôle (`.prevChecksum`)

Le fichier `.prevChecksum` n'a pas vocation à être consulté par l'utilisateur, d'où le fait qu'il soit caché. Il permet de stocker la somme de contrôle en 256 bits du dump SQL précédent, pour pouvoir déterminer si des modifications sont advenues.

4 Utilisation des variables

4.1 Variables du fichier de configuration

Le fichier de configuration `archive.conf` comprend les variables suivantes :

4.1.1 Configuration générale

`emplacementLog` Définit où les logs sont enregistrés. Attention, l'utilisateur qui exécute le script doit avoir les droits d'écriture dans le dossier parent.

Par défaut sur `./archive.log`.

`logStdout` En cas d'échec, redirige le motif à la sortie standard (`0`) ou pas (`1`).

Par défaut sur `0`.

`archiveURL` Définit l'emplacement de l'archive via une URL, accessible depuis l'ordinateur client.

4.1.2 Serveur d'archivage

`adresseArchivage` Adresse IP du serveur d'archivage, qui doit être accessible via SSH. Si le port 22 est utilisé, pas besoin de le préciser.

`usernameSSH` Nom d'utilisateur à utiliser sur le serveur d'archivage.

pathSSH Chemin sur lequel enregistrer les archives sur le serveur. Le chemin doit déjà exister et être accessible en lecture et écriture pour l'utilisateur renseigné à `usernameSSH`.

dureeConservation Durée à partir de laquelle les anciennes archives seront supprimées, en jours.

Par défaut sur `30`.

4.1.3 Envoi de mails

envoyerMail Dans quel cas envoyer un mail, jamais (`0`), en cas d'échec de l'exécution (`1`) ou toujours (`2`).

Par défaut sur `1`.

mailDestinataires=(dest1@mail.org dest2@mail.org) Destinataires du mail, séparés par des espaces. Si cette liste est vide, et peu importe la valeur de `envoyerMail`, le programme quittera sans envoyer de mail et sans erreur.

objSucces Objet du mail à envoyer en cas de succès.

Par défaut sur `Archivage du $(date +'%d %B %Y') réussi`.

objEchec Objet du mail à envoyer en cas d'échec.

Par défaut sur `Archivage du $(date +'%d %B %Y') échoué`.

joindreLog Dans quelle situation joindre le fichier de logs complet, jamais (`0`), en cas d'échec (`1`) ou toujours (`2`).
Attention, il s'agit du fichier de log entier. Le motif d'échec, si c'est le cas, est toujours indiqué dans le corps du message.

Par défaut sur `1`.

muttrcUtilisateur Utiliser le `~/.muttrc` de l'utilisateur (`0`) ou non (`1`).

Par défaut sur `1`.

4.1.3.1 Serveur SMTP

Ces options n'auront aucune incidence si `muttrcUtilisateur=0`.

serveurHote Serveur SMTP qui gère l'envoi de mails.

port Port sur lequel contacter le serveur. En général, on a :

- `25` : sans chiffrement
- `465` : chiffrement implicite (TLS/SSL)
- `587` : chiffrement explicite (STARTTLS)

mailEnvoyeur Mail envoyeur des informations de l'utilitaire, enregistré sur le serveur renseigné dans `serveurHote`.

motDePasse Mot de passe associé au mail pour s'identifier sur `serveurHote`.

4.2 Variable utilisée dans le script

Toutes les variables du fichier de configuration sont utilisées dans le script. On ne définit qu'une variable dans le script, comme tel :

```
currentChecksum=$(sha256sum ./*.sql | cut -d ' ' -f1)
```

Cela permet de stocker la somme de contrôle du dump SQL en cours de traitement pour pouvoir la comparer avec la somme de contrôle sauvegardée.

---

5 Conclusion

Le script produit est donc fonctionnel, et considère toutes les possiblités d'erreurs.

On peut cependant envisager une amélioration de sécurité au niveau du stockage du mot de passe du compte mail (`$motDePasse`), qui est stocké en clair. Pour cela, on peut utiliser une implémentation d'OpenPGP, comme GnuPG.
Cela nécessiterait néanmoins un travail de documentation supplémentaire.

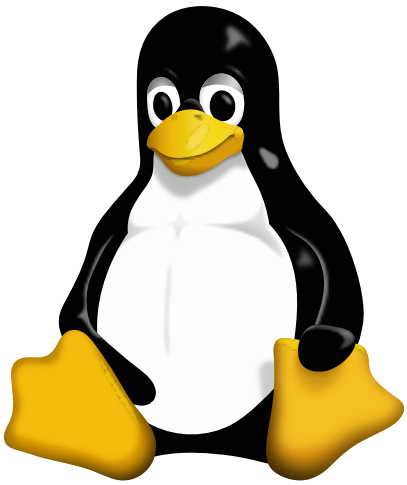

Figure 3: Tux

Annexes

Fichier de configuration (`archive.conf`)

```
################################################
## CONFIGURATION DE L'UTILITAIRE D'ARCHIVAGE ##
################################################

## Emplacement des logs de l'utilitaire
emplacementLog="archive.log"

## En cas d'erreur, renvoyer le motif à la sortie standard en plus des logs
# 0 : vrai
# 1 : faux
logStdout="0"

## Adresse de l'archive ZIP contenant un fichier SQL
archiveURL="http://176.190.35.242:80/sqldump.zip"


#########################
## SERVEUR D'ARCHIVAGE ##
################################################

## Adresse du serveur sur lequel on veut archiver les dumps
adresseArchivage="176.190.35.242"

## Utilisateur qui va recevoir les sauvegardes, sur le serveur d'archivage
usernameSSH="john-doe"

## Emplacement des sauvegardes, en chemin absolu (possiblité d'utiliser `~`)
pathSSH="~/Sauvegardes"

## Durée de conservation des archives sur le serveur, en jours
dureeConservation="30"


##########
## MAIL ##
################################################

## Dans quel cas envoyer un mail
# 0 : Jamais
# 1 : En cas d'échec
# 2 : Toujours
envoyerMail="1"

## Adresses mail sur lesquels on veut envoyer un état de la dernière exécution (succès / échec). Séparer chaque adresse par des espaces.
# Pour envoyer des mails internes, on peut utiliser les noms d'utilisateurs (e.g. : root)
```

```
mailDestinataires=(dest1@mail.com)

## Objet des mails en cas de succès
objSucces="Archivage du $(date +'%d %B %Y') réussi"
## Objet des mails en cas d'échec
objEchec="Archivage du $(date +'%d %B %Y') échoué"

## Joindre le log
# 0 : Jamais
# 1 : En cas d'échec
# 2 : Toujours
joindreLog="1"

## Utiliser le `.muttrc` de l'utilisateur. Si oui, la modifications des options d'après n'aura aucune incidence.
# 0 : vrai
# 1 : faux
muttrcUtilisateur="1"

###################
## SERVEUR SMTP ##
###################

## Authentification sur serveur de messagerie, si nécessaire.
serveurHote="smtp.mail.org"

## Généralement :
# 25 (sans chiffrement)
# 465 (chiffrement implicite)
# 587 (chiffrement explicite)
port="465"

mailEnvoyeur="envoyeur@mail.org"
motDePasse="mdp"
```

Script (`archive.sh`)

```
#!/bin/bash

# Importer le fichier de configuration
source "./archive.conf"

################################################
## Fonction d'écriture de log
# $1 : succès (0) / échec (1) de l'opération
# $2 : si échec, description
#      si succès, checksum du fichier concerné
function ecrireLog() {
    if [[ $1 -eq 0 ]]; then
```

```
13          echo "[ $(date +'%T - %d %b %Y') ] : Succès, checksum=$2" >>
            ↪ "$emplacementLog"
14      else
15          [[ $logStdout -eq 0 ]] && echo "$2"
16          echo "[ $(date +'%T - %d %b %Y') ] : Échec, $2" >> "$emplacementLog"
17      fi
18  }
19
20  ## Fonction d'envoi de mail
21  # $1 : succès (0) / échec (1)
22  # $2 : si échec, corps du message
23  function envoyerMail() {
24      if [[ ${#mailDestinataires[*]} -ne 0 && (($1 -eq 1 && $envoyerMail -eq 1)
        ↪ || $envoyerMail -eq 2) ]]; then
25
26          if [[ $muttrcUtilisateur -eq 0 ]]; then
27              echo "$([[ $1 -eq 0 ]] && echo L\'opération de ce jour est un
                ↪ succès || echo $2)" | \
28                  mutt -x \
29                       -s "$([[ $1 -eq 0 ]] && echo $objSucces || echo
                         ↪ $objEchec)" \
30                       $([[ $joindreLog -eq 2 || ($1 -eq 1 && $joindreLog -eq 1)
                         ↪ ]] && echo "-a $emplacementLog --") \
31                       "$(echo ${mailDestinataires[*]})"
32
33          else
34              echo "$([[ $1 -eq 0 ]] && echo L\'opération d\'archivage de ce
                ↪ jour est un succès. || echo $2)" | \
35                  mutt -nx \
36                       -e "set from = \"$mailEnvoyeur\"" \
37                       -e "set smtp_pass = \"$motDePasse\"" \
38                       -e "set smtp_url =
                         ↪ \"smtps://$mailEnvoyeur@$serveurHote:$port\"" \
39                       -e "set send_charset = \"utf-8\"" \
40                       -s "$([[ $1 -eq 0 ]] && echo $objSucces || echo
                         ↪ $objEchec)" \
41                       $([[ $joindreLog -eq 2 || ($1 -eq 1 && $joindreLog -eq 1)
                         ↪ ]] && echo "-a $emplacementLog --") \
42                       "$(echo ${mailDestinataires[*]})"
43          fi
44      fi
45
46      [[ $? -ne 0 ]] && ecrireLog 1 "erreur lors de l'envoi du mail."
47  }
48
49  ## Fonction combo
50  # $1 : succès (0) / échec (1)
51  # $2 : si échec, description
52  #      si succès, checksum du fichier concerné
```

```
53 function combo() {
54     if [[ "$1" -eq 0 ]]; then
55         ecrireLog 0 "$2"
56         envoyerMail 0
57     else
58         ecrireLog 1 "$2"
59         envoyerMail 1 "L'opération d'archivage de ce jour a échoué pour le
       ↪  motif suivant : $2"
60     fi
61 }
62
63 ################################################
64 # Ménage sur le serveur d'archivage
65 if ! ssh "$usernameSSH@$adresseArchivage" "find $pathSSH -name *.tgz -type f
   ↪  -ctime +$dureeConservation -exec rm '{}' \; && exit"; then
66     combo 1 "impossible d'accéder au serveur d'archivage renseigné via SSH,
       ↪  vérifier la configuration."
67     exit 1
68 fi
69
70 # Vérifier si l'URL existe
71 if ! wget -q "$archiveURL"; then
72     combo 1 "URL renseignée inaccessible, ou \`wget\` n'est pas installé."
73     exit 1
74 fi
75
76 # Décompresser l'archive
77 if ! unzip -q ./*.zip; then
78     combo 1 "erreur lors de l'invocation de \`unzip\`."
79     exit 1
80 fi
81
82 # Supprimer l'archive
83 rm -f ./*.zip
84
85 # Vérifier si il y a un seul fichier SQL
86 if [[ ! $(ls ./*.sql | wc -l) -eq 1 ]]; then
87     combo 1 "structure d'archive à modifier, plusieurs fichiers SQL ou
       ↪  organisation inadéquate."
88     exit 1
89 fi
90
91 # Stocker la somme de contrôle du nouveau dump
92 if ! currentChecksum=$(sha256sum ./*.sql | cut -d ' ' -f1); then
93     combo 1 "fichier SQL non trouvé, ou \`sha256sum\` inaccessible."
94     exit 1
95 fi
96
97 # Si la sauvegarde de somme de contrôle n'existe pas, la créer
```

```
98  touch .prevChecksum
99
100 # Vérifier les changements
101 if [[ "$(cat .prevChecksum)" == "$currentChecksum" ]]; then
102     rm -f ./*.sql
103     combo 1 "somme de contrôle identique à la dernière enregistrée."
104     exit 1
105 fi
106
107 # Archiver le nouveau dump
108 if ! tar -czf "$(date +'%Y%d%m')".tgz ./*.sql; then
109     combo 1 "erreur lors de la compression en tgz, vérifier que \`tar\` est
        ↪  installé et accessible."
110     exit 1
111 fi
112
113 # Supprimer le dump SQL
114 rm -f ./*.sql
115
116 # Pousser sur le serveur, avec SSH (SFTP)
117 if ! scp -q ./*.tgz "$usernameSSH@$adresseArchivage:$pathSSH"; then
118     combo 1 "erreur lors de la connexion SSH au serveur d'archivage, vérifier
        ↪  les informations de connexion."
119     exit 1
120 fi
121
122 # Supprimer l'archive
123 rm -f ./*.tgz
124
125 # Actualiser la somme de contrôle
126 echo "$currentChecksum" > .prevChecksum
127
128 combo 0 "$currentChecksum"
129 exit 0
```