
LO14

Rapport de fin de projet

Développement d'un serveur d'archives en *bash*

Préface

Dans le cadre de l'UE LO14 – *Administration des systèmes*, nous avons été amenés à concevoir un utilitaire de gestion d'archives orienté client / serveur, écrit en *bash* et conçu pour les systèmes UNIX.

Table des matières

I. Introduction	3
1. Présentation de l'utilitaire.....	3
2. Architecture du logiciel.....	3
2.1. Organisation du code source.....	3
2.2. Communication client-serveur.....	4
II. Manuel d'utilisation	5
1. Guide de démarrage.....	5
1.1. Téléchargement du code source.....	5
1.2. Création d'un alias.....	5
1.3. Dépendances.....	5
2. Usage en mode serveur.....	5
2.1. Mode <i>listen</i>	5
3. Usage en mode client.....	6
3.1. Mode <i>list</i>	6
3.2. Mode <i>create</i>	6
3.3. Mode <i>browse</i>	7
3.4. Mode <i>extract</i>	8
4. Messages d'erreur.....	9
4.1. Dépendances manquantes.....	9
4.2. Erreurs de connexion.....	9
4.3. Erreur d'accès aux archives.....	10
5. Scénario d'utilisation.....	10
III. Rétrospective sur le travail réalisé	13
1. Organisation.....	13
1.1. Phase de conception.....	13
1.2. Travail collaboratif.....	13
1.3. Progression.....	13
2. Difficultés rencontrées.....	14
2.1. Ecoute sur plusieurs ports.....	14

2.2. Accès aux autres scripts.....	15
2.3. Chemins utilisant des barres obliques inverses comme séparateurs.....	16
2.4. Contrôle du flux de communication avec le serveur.....	16
2.5. Scripts modifiant l'archive.....	17
2.6. Exécution du mode extract en tant que root.....	17
2.7. Problèmes de compatibilité.....	18
3. Auto-critique.....	19
3.1. Pas de script dédié pour <i>parser</i> les chemins.....	19
3.2. Mauvaises performances du mode browse.....	19
4. Bogues connus.....	19
4.1. Interruption du serveur par le signal d'interruption ^C.....	19
4.2. Présence d'un fichier contenant un octet nul.....	20
4.3. Présence d'un fichier contenant des signaux spéciaux.....	20
4.4. Modifications concurrentes d'une archive.....	20
IV. Conclusion	21
V. Annexes	22
1. Echantillon d'archive.....	22

I. Introduction

1. Présentation de l'utilitaire

vsh est un gestionnaire d'archives conçu pour les systèmes *UNIX* et présentant une interface en lignes de commandes.

Il permet à la fois de mettre en place un serveur d'archives et de s'y connecter.

2. Architecture du logiciel

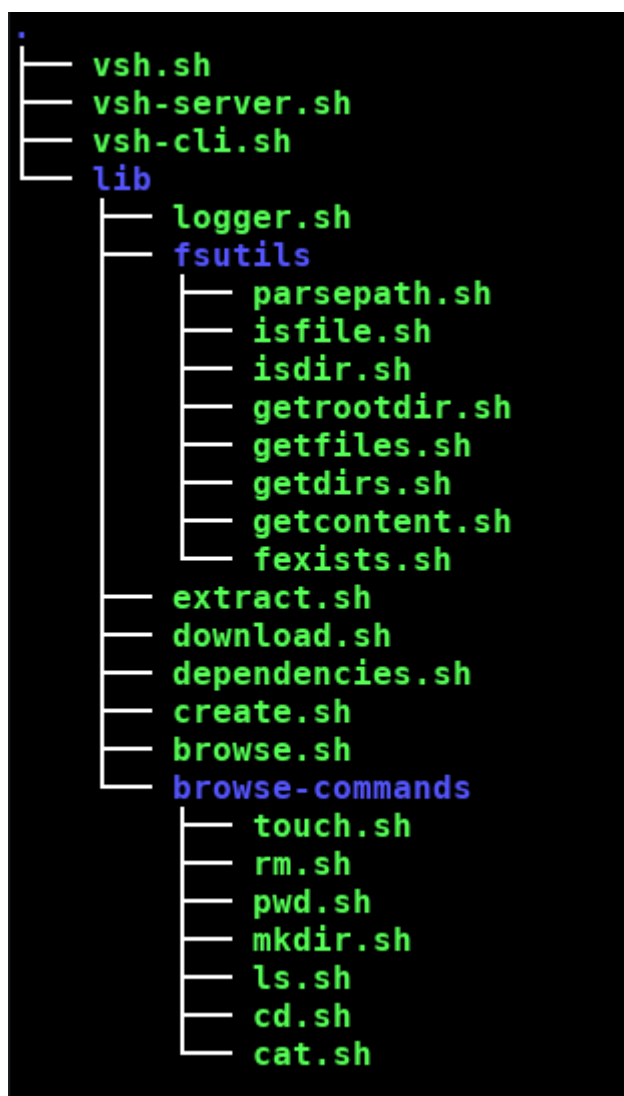
2.1. Organisation du code source

Le code source est organisé sous la forme d'un ensemble de scripts interprétables par le shell *bash*.

Les traitements sont pour la plupart réalisés par des scripts dédiés par souci de modularité et de factorisation du code.

Une classement en hiérarchie arborescente regroupe les scripts par niveaux.

L'arborescence du répertoire principal du logiciel se présente comme suit :



Les scripts "*terminaux*" sont situés au niveau principal.

vsh est un lanceur commun permettant d'accéder à l'ensemble des fonctionnalités.

C'est l'**exécutable principal** du logiciel.

vsh-server et *vsh-cli* sont les scripts permettant respectivement d'accéder aux fonctionnalités côté serveur et côté client.

Le répertoire *lib* contient des scripts utilitaires dédiés, appelés par les scripts *terminaux*.

- *logger*, responsable de l'affichage de messages spéciaux.

- *dependencies*, vérifiant la présence des dépendances logicielles.

- *download*, permettant au client et au serveur d'échanger des fichiers.

- Des scripts exécutables individuellement et correspondant chacun à une fonctionnalité.

- Un répertoire *fsutils* contenant des scripts utilitaires indépendants.

- Un répertoire *browse-commands* contenant un script par commande du mode *browse*.

Ils sont exécutables individuellement, mais dépendant de variables d'environnement exportées par le script *browse*.

Figure I – 2.1.a : Organisation du code source du logiciel

2.2. Communication client-serveur

La connexion *TCP* entre la machine cliente et le serveur est réalisée au moyen de l'outil *netcat* et de tubes nommés temporaires, stockés dans le répertoire */tmp* et supprimés après utilisation.

Gestion de la connexion côté client :

Elle est réalisée automatiquement lors de l'exécution d'une commande par le client, sur la base du nom d'hôte et du port fournis en argument (voir II.3 – Manuel d'utilisation : *Usage en mode client*).

netcat est exécuté une seule fois par commande. Son entrée est redirigée depuis un tube nommé, et sa sortie vers un autre.

```
"$NETCAT" "$SERVER" "$PORT" < "$OUTGOING" > "$INCOMING" &
```

Figure I – 2.2.a : Connexion du client au serveur (script `\vsh-cli.sh`, fonction `\connect`)

Lorsqu'un client se déconnecte, le port qu'il utilisait redevient disponible pour un autre client.

La déconnexion est réalisée automatiquement à la fin d'une commande en terminant le processus *netcat*.

```
connection_is_active
if test $? -eq 0
then kill $NCPID
```

Figure I – 2.2.b : Fermeture d'une connexion par le client (script `\vsh-cli.sh`, fonction `\connect`)

Gestion de la connexion côté serveur :

Le serveur exécute une instance de *netcat* en tâche de fond pour chaque port sur lequel il écoute (voir II.2 – Manuel d'utilisation : *Usage en mode serveur*). Chacune des instances redirige sa sortie vers un unique tube nommé. Le contenu de ce tube est traité puis passé en entrée à *netcat*.

```
serve < "$lfifo" | "$NETCAT" -l -p $lport > "$lfifo"
```

Figure I – 2.2.c : Connexion du client au serveur (script `\vsh-server.sh`, fonction `\listen`). Cette instruction se répète en continu et en tâche de fond, pour chaque port.

La déconnexion est effectuée manuellement par l'opérateur du serveur.

```
[f ✖ ~]# vsh listen 9990-9991
Listening on 9990...
Listening on 9991...

Type in 'stop' to shut the server down.
stop
Shutting down the server...
[f ✖ ~]#
```

Figure I – 2.2.d : La commande `\stop` met fin à l'écoute par le serveur.

II. Manuel d'utilisation

1. Guide de démarrage

1.1. Téléchargement du code source

L'ensemble du code source est accessible publiquement et librement à partir de [ce dépôt en ligne](#).

1.2. Création d'un alias

Vous pouvez créer un alias de l'exécutable principal afin d'y accéder plus aisément.

```
alias vsh="/absolute/path/to/vsh.sh"
```

En ajoutant cette instruction à un fichier de configuration de *bash* comme suit,

```
echo "alias vsh='/absolute/path/to/vsh.sh'" >> ~/.bashrc
```

vous rendrez ce raccourci permanent.

Une autre solution consiste à créer un lien vers l'exécutable dans l'un des répertoires de votre variable **\$PATH** :

```
ln -s /absolute/path/to/vsh.sh /usr/bin/vsh
```

1.3. Dépendances

Pour fonctionner, l'utilitaire nécessite que les dépendances suivantes soient installées sur votre machine :

- *netcat* ou *nc*
- *gawk*

2. Usage en mode serveur

2.1. Mode *listen*

La commande d'écoute permet d'écouter et servir les connexions entrantes sur un ou plusieurs ports valides.

```
[f ✖ ~]# vsh -listen
USAGE : vsh.sh -listen <(port|port-range)...>
```

Figure II – 2.1.a : Usage du mode `-listen`

Le verbe *listen* peut indifféremment être précédé d'un tiret ou non.

```
[f ✖ ~]# vsh -listen 8880-8888 8889 80
!-- 80 : Reserved socket. Choose a minimum port number of 1024. --!
Listening on 8880...
Listening on 8881...
Listening on 8882...
Listening on 8883...
Listening on 8884...
Listening on 8885...
Listening on 8886...
Listening on 8887...
Listening on 8888...
Listening on 8889...
Type in 'stop' to shut the server down.
```

Figure II – 2.1.b : Exemple d'utilisation du mode `-listen` avec une plage de ports et plusieurs ports individuels.

L'écoute échouera pour des ports déjà utilisés par d'autre processus ou les ports réservés.

Le mode écoute peut être interrompu par l'opérateur du serveur au moyen de la commande **stop**.

3. Usage en mode client

Les commandes côté client établissant toutes une connexion avec un serveur d'archives, elles prennent systématiquement en argument un nom d'hôte et un numéro de port.

Le tiret avant le verbe (second argument) est facultatif.

3.1. Mode *list*

Le client peut lister les archives présentes sur un serveur exécutant *vsh* au moyen du verbe ``-list``.

```
USAGE : vsh.sh -list <server_name> <port>
```

Figure II – 3.1.a : Usage du mode ``-list``

```
[f * ~]$ vsh list localhost 4447
Archives of 93.10.66.136 :
-----
+ test          (already in use)
+ titi
[f * ~]$
```

Figure II – 3.1.b : Liste des archives présentes sur un serveur distant.

Une archive déjà utilisée ne sera accessible ni en mode ``-browse`` ni en mode ``-extract`` (voir II.4 - Manuel d'utilisation : Erreur d'accès aux archives).

3.2. Mode *create*

Le client peut créer une archive sur un serveur distant à partir de son répertoire de travail actuel au moyen du verbe ``-create``.

```
USAGE : vsh.sh -create <server_name> <port> <archive_name>
```

Figure II – 3.2.a : Usage du mode ``-list``

La racine de l'archive ainsi créée sera le chemin absolu du répertoire de travail du client.

Une archive temporaire est créée localement côté client, puis envoyée au serveur, qui la stocke dans son répertoire *archives*, avec le nom indiqué et l'extension `'.sos'`.

```
[f * ~]$ vsh -listen 9999
Listening on 9999...

Type in 'stop' to shut the server down.
stop
Shutting down the server...
[f * ~]$ cat "$vshpath/archives/mon-archive.sos"
3:44
directory \home\f\Bureau\test\

[f * test]$ pwd
/home/f/Bureau/test
[f * test]$ vsh create 93.10.66.136 9999 mon-archive
creating mon-archive
Archive 'mon-archive' has been created successfully.
[f * test]$
```

Figure II – 3.2.b : Creation d'une archive sur un serveur distant.

A droite, le client, qui archive l'arborescence `'/home/f/Bureau/test'`.

A gauche, le serveur, qui a téléchargé l'archive.

3.3. Mode *browse*

Ce mode permet de parcourir et modifier une archive présente sur le serveur.

USAGE : `vsh.sh -browse <server_name> <port> <archive_name>`

Figure II – 3.3.a : Usage du mode `-browse``

Une fois connecté au serveur en mode `-browse``, le client a accès à l'ensemble des commandes suivantes, se comportant de la même manière que les commandes *UNIX* homonymes.

Commandes du mode *browse* :

- `pwd` : Affiche le répertoire courant.
- `ls [path] [-option...]` : Liste le contenu du répertoire courant ou passé en argument / indique si un fichier existe.
Les noms de répertoires sont suivis de `'/'` et ceux des fichiers exécutables de `'*'`.
Options :
 - `-l` : affiche plus de détails sur les éléments listés
 - `-a` : affiche également les éléments cachés
 - `-la` : combine les deux premières options.
- `cd [path-to-dir]` : Change de répertoire de travail.
Sans arguments, retourne à la racine.
- `cat <path-to-file...>` : Affiche le contenu d'un ou plusieurs fichiers à la suite.
- `mkdir <path-to-new-dir...>` : Crée les répertoires passés en argument, s'ils n'existent pas et si leur répertoire parent existe.
Si le chemin est absolu, le nouveau répertoire est créé dans le dossier courant.
Option `-p` : crée récursivement les dossiers parents manquants.
- `touch <path-to-file...>` : Crée les fichiers vides ayant les chemins passés en argument.
Leur assigne les droits `rwxr-xr-x`.
- `rm <path...>` : Supprime tous les éléments dont le chemin est donné en argument.
- `stop` : Déconnecte le client.
Alias `^C`, `exit`, `disconnect` et `dc`.

Chemins dans le système de fichiers de l'archive :

- Le séparateur utilisé est la barre oblique inverse `'\'`.
- Un `'\'` en début de chemin désigne la racine de l'archive. Le chemin complet de la racine peut également être utilisé.
- Les chemins relatifs ne commencent pas par la racine.

- ‘..’ désigne le répertoire parent, ou la racine si l’utilisateur s’y trouve.
- ‘.’ désigne le répertoire courant.

```
[f ✖ ~]# vsh -browse localhost 9999 mon-archive
Browsing 'mon-archive'

Type 'stop' to disconnect.
# ls -al

Content of '\':
-----
drwxr-xr-x      4096    dossier
drwxr-xr-x      4096    dossier avec espaces
drwxr-xr-x      4096    .dossier_discret
-rwx-wx--x       31    droits73l
-rw-r--r--      111    .fichier_discret
drwxr-xr-x      4096    totos
-rw-r--r--       62    tutu.txt
-----
Total : 7 elements

# cat .fichier_discret
Mais comment vous m'avez trouvé !
J'étais pourtant si bien caché ...

Bon surtout, ne dites rien aux autres.

# cd dossier avec espaces

# pwd
\dossier avec espaces\

# cd ..

# pwd
\

# rm dossier
'\home\f\Bureau\test\dossier\sous-dossier\fichier' has been removed successfully.
'\home\f\Bureau\test\dossier\sous-dossier\' has been removed successfully.
'\home\f\Bureau\test\dossier\' has been removed successfully.

# exit
```

Figure II – 3.3.b : Exemple d’utilisation du mode ``-browse``.

3.4. Mode *extract*

Ce mode permet au client d’extraire sur sa machine une archive stockée sur le serveur. L’arborescence contenue par l’archive est reconstituée dans le répertoire de travail du client.

USAGE : `vsh.sh -extract <server_name> <port> <archive_name>`

Figure II – 3.3.c : Usage du mode ``-extract``.

Point d’extraction

Soit une archive *test* de racine “*/home/toto*” :

Si le client extrait l’archive dans “*/home/titi/Documents/*”, un répertoire “*/home/titi/Documents/home/toto*” sera créé.

Si le client extrait l’archive dans “*/*”, le contenu de l’archive sera restitué dans “*/home/toto*”.

Si l'utilisateur ne dispose pas des droits d'écriture dans son répertoire de travail, il doit appeler le script en tant que *root* – le répertoire extrait lui appartiendra quand même.

```
[f * /]# sudo "/home/f/Bureau/L014/Projet/vsh-to14/scripts/vsh.sh" -extract 93.10.66.136 9999 mon-archive
Extracting archive mon-archive at '/' ...

Script ran as root !
Ownership of /home/f/Bureau/test
Granted to :
  User    →    f
  Group   →    f
[f * /]# ls -l /home/f/Bureau/test
total 20
drwxr-xr-x 3 f f 4096 31 déc. 16:18 dossier
drwxr-xr-x 2 f f 4096 31 déc. 16:14 'dossier avec espaces'
```

Figure II – 3.3.d : Extraction d'une archive à la racine en tant que root.

4. Messages d'erreur

4.1. Dépendances manquantes

netcat

La connexion client-serveur étant réalisée via *netcat*, les scripts client comme serveur échoueront si **ni *netcat*, ni *nc*** ne sont présents dans un répertoire de sa variable **\$PATH**.

```
[f * ~]# vsh listen 9999
!-- Could not find a netcat executable on your system. --!
```

Figure II – 4.1.a : La dépendance à *netcat* ou *nc* n'a pas pu être satisfaite.

gawk

Les scripts *mkdir*, *touch* et *rm* nécessitent que *gawk* soit installé sur le système local. (voir III.2.7 - Problèmes de compatibilité).

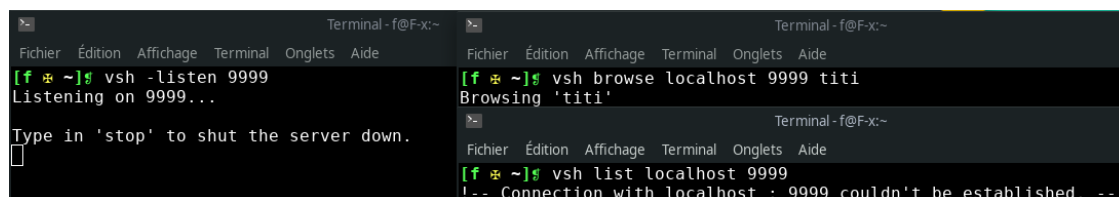
Les scripts terminaux de *vsh* vérifieront que cette dépendance peut être satisfaite.

```
[f * ~]# vsh
!-- Missing dependency : gawk --!
```

Figure II – 4.1.b : La dépendance à *gawk* n'a pas pu être satisfaite.

4.2. Erreurs de connexion

Tentative de connexion client sur un port déjà pris



```
Terminal - f@F-x:~
Fichier  Édition  Affichage  Terminal  Onglets  Aide
[f * ~]# vsh -listen 9999
Listening on 9999...

Type in 'stop' to shut the server down.
□

Terminal - f@F-x:~
Fichier  Édition  Affichage  Terminal  Onglets  Aide
[f * ~]# vsh browse localhost 9999 titi
Browsing 'titi'

Terminal - f@F-x:~
Fichier  Édition  Affichage  Terminal  Onglets  Aide
[f * ~]# vsh list localhost 9999
!-- Connection with localhost : 9999 couldn't be established. --!
```

Figure II – 4.2.a : La connexion échoue si un autre client est déjà connecté sur le port demandé

Nom d'hôte introuvable ou port fermé

```
[f * ~]# vsh list loocaalhoost 9999
!-- Connection with loocaalhoost : 9999 couldn't be established. --!
```

Figure II – 4.2.b : Connexion à un hôte introuvable

Connexion perdue

```
[f * ~]$ vsh listen 9999
Listening on 9999...

Type in 'stop' to shut the server down.
█

[f * ~]$ vsh browse localhost 9999 titi
Browsing 'titi'

Type 'stop' to disconnect.
B █
```

Figure II – 4.2.c : Le serveur (à gauche) et le client (à droite) sont connectés.

```
[f * ~]$ vsh listen 9999
Listening on 9999...

Type in 'stop' to shut the server down.
stop
Shutting down the server...
[f * ~]$ █

[f * ~]$ vsh browse localhost 9999 titi
Browsing 'titi'

Type 'stop' to disconnect.
B !-- Connection with localhost : 9999 was lost. --!
[f * ~]$ █
```

Figure II – 4.2.d : L'opérateur du serveur interrompt l'écoute.
Le client perd sa connexion au serveur.

4.3. Erreur d'accès aux archives

Une archive en train d'être parcourue par un client connecté en mode *browse* est rendue inaccessible par le serveur au moyen d'un verrou d'exclusion mutuelle.

5. Scénario d'utilisation

Démonstration de l'utilisation de vsh avec deux clients et un serveur :

```
[f * scn]$ vsh -listen 9990-9999
Listening on 9990...
Listening on 9991...
Listening on 9992...
Listening on 9993...
Listening on 9994...
Listening on 9995...
Listening on 9996...
Listening on 9997...
Listening on 9998...
Listening on 9999...

Type in 'stop' to shut the server down.
```

Figure II – 5.S1 : L'opérateur du serveur lance l'écoute sur la plage de ports allant de 9990 à 9999.

```
zhangxz@zhangxz-TM1613:~/Bureau/vraitoto$ vsh -list 93.10.66.136 9995
Archives of 93.10.66.136 :
-----
!-- No archives were found on this server --!
```

Figure II – 5.C2 : Un client distant souhaite afficher la liste des archives présentes sur le serveur. Le serveur lui répond qu'il n'y en a pour le moment aucune.

```
zhangxz@zhangxz-TM1613:~/Bureau/vraitoto$ vsh -create 93.10.66.136 9995 uarchive
creating uarchive
Archive 'uarchive' has been created successfully.
zhangxz@zhangxz-TM1613:~/Bureau/vraitoto$ vsh -list 93.10.66.136 9995
Archives of 93.10.66.136 :
-----
+ uarchive
zhangxz@zhangxz-TM1613:~/Bureau/vraitoto$ █
```

Figure II – 5.C2 : Il crée sur le serveur une archive *uarchive* à partir de son répertoire de travail.

```

zhangxz@zhangxz-TM1613:~/Bureau/vraitoto$ pwd
/home/zhangxz/Bureau/vraitoto
zhangxz@zhangxz-TM1613:~/Bureau/vraitoto$ ls
totoA  totoB
zhangxz@zhangxz-TM1613:~/Bureau/vraitoto$ cat totoA totoB
Salut, c'est moi, le vrai toto. totoA.
Bonne année 2022 !
Ne croyez pas l'autre, ce n'est qu'un usurpateur!
Salut, c'est moi, le vrai toto. totoB.
Bonne année 2022 !
Ne croyez pas l'autre, ce n'est qu'un usurpateur!
zhangxz@zhangxz-TM1613:~/Bureau/vraitoto$

```

Figure II – 5.C3 :

Voici comment se présente son répertoire de travail, à partir duquel il a créé l'archive.

```

[f ✖ archives]$ cat uarchive.sos |lolcat -F 1
3:7

directory \home\zhangxz\Bureau\vraitoto\
totoA  -rw-rw-r-- 110 7 3
totoB  -rw-rw-r-- 110 10 3
@
Salut, c'est moi, le vrai toto. totoA.
Bonne année 2022 !
Ne croyez pas l'autre, ce n'est qu'un usurpateur!
Salut, c'est moi, le vrai toto. totoB.
Bonne année 2022 !
Ne croyez pas l'autre, ce n'est qu'un usurpateur!
[f ✖ archives]$

```

Figure II – 5.S2 :

L'archive a bien été créée sur le serveur.

```

zhangxz@zhangxz-TM1613:~/Bureau/vraitoto$ vsh -browse 93.10.66.136 9995 uarchive
Browsing 'uarchive'

-----

Type 'stop' to disconnect.
B ls
totoA  totoB

B cat totoA totoB
Salut, c'est moi, le vrai toto. totoA.
Bonne année 2022 !
Ne croyez pas l'autre, ce n'est qu'un usurpateur!
Salut, c'est moi, le vrai toto. totoB.
Bonne année 2022 !
Ne croyez pas l'autre, ce n'est qu'un usurpateur!

B rm totoB
'\home\zhangxz\Bureau\vraitoto\totoB' has been removed successfully.

B touch totoC
'totoC' has been created successfully in '\home\zhangxz\Bureau\vraitoto\'

B stop
zhangxz@zhangxz-TM1613:~/Bureau/vraitoto$

```

Figure II – 5.C4 : Le client décide de parcourir cette archive pour vérifier qu'elle est bien identique à l'arborescence du répertoire d'origine.
Il supprime au passage le fichier *totoB* et crée un fichier vide *totoC*.
Puis, il se déconnecte.

```
[f ✖ archives]$ cat uarchive.sos |lolcat -F 1
3:7

directory \home\zhangxz\Bureau\vraitoto\
totoA -rw-rw-r-- 110 7 3
totoC -rwxr-xr-x 0 9 0
@
Salut, c'est moi, le vrai toto. totoA.
Bonne année 2022 !
Ne croyez pas l'autre, ce n'est qu'un usurpateur!
```

Figure II – 5.S3 :
L'archive a été
modifiée en
conséquence sur le
serveur.

```
$ vsh -extract 93.10.66.136 9992 uarchive
Extracting archive uarchive at '/home/f/Documents' ...
$ tree
├── home
│   └── zhangxz
│       └── Bureau
│           └── vraitoto
│               ├── totoA
│               └── totoC
└── ...

4 directories, 2 files
$ cd home/zhangxz/Bureau/vraitoto/
$ ls -l
total 4
-rw-rw-r-- 1 f f 109  5 janv. 22:59 totoA
-rwxr-xr-x 1 f f  0  5 janv. 22:59 totoC
$ cat totoA
Salut, c'est moi, le vrai toto. totoA.
Bonne année 2022 !
Ne croyez pas l'autre, ce n'est qu'un usurpateur!
$
```

Figure II – 5.K1 :
Un autre client
se connecte au
serveur et
extraît
l'archive dans
son répertoire
de travail.
Le contenu de
l'arborescence
est bien
conforme à
l'archive
stockée sur le
serveur.

```
Type in 'stop' to shut the server down.
stop
Shutting down the server...
```

Figure II – 5.S4 : L'opérateur du serveur peut arrêter le mode écoute après une dure journée à regarder des archives multicolores.

III. Rétrospective sur le travail réalisé

1. Organisation

1.1. Phase de conception

Préalablement à la phase de développement du projet, nous avons mené une phase de conception préliminaire visant à définir les contours du logiciel et à réfléchir aux technologies à utiliser pour le réaliser, à son architecture, aux différents obstacles que nous pourrions être amenés à rencontrer, et aux algorithmes à implémenter pour chaque fonctionnalité.

Ce processus de réflexion nous a notamment permis de mettre en commun nos idées pour nous mettre d'accord sur une vision commune du projet.

Il nous a également fait gagner beaucoup de temps en nous permettant d'organiser efficacement la répartition des tâches au sein de notre binôme.

En définissant à l'avance les relations d'interdépendance entre les différents modules, et en distinguant les interfaces client / serveur des traitements, nous avons été en mesure de nous répartir le travail de manière à pouvoir progresser séparément chacun à notre rythme, puis de mettre en commun nos scripts pour finaliser le projet.

1.2. Travail collaboratif

Nous avons utilisé l'outil de gestion de configuration *git* afin de collaborer via un dépôt distant *github* maintenu à jour avec les dernières versions stables de notre code source. Ayant planifié préalablement les modules que chacun avait à produire, nous n'avons pas eu recours à ses fonctionnalités avancées de gestion de version telles que les branches.

Etant voisins, nous avons pu nous réunir régulièrement pour faire le point sur l'avancement de notre projet, nous aider mutuellement et nous mettre d'accord sur des changements de stratégie.

Nous avons également communiqué via l'application de messagerie instantanée *Discord* pendant les vacances.

Pour la présentation orale, nous avons configuré un routeur pour qu'il redirige les connexions entrantes sur une certaine plage de ports vers l'un de nos ordinateurs, créant ainsi un serveur accessible publiquement, afin de faire chacun la démonstration de fonctionnalités différentes sur une même archive.

1.3. Progression

Nous avons **commencé par créer les scripts terminaux** (voir I.2.1 – Organisation du code source) en implémentant seulement, dans un premier temps, la connexion client-serveur. Une fonction affichant un simple message temporaire a été définie pour

chaque commande.

La première étape a été d'implémenter, côté serveur, **un mode d'écoute sur un seul port**, que nous avons **ensuite transformé en un mode d'écoute multiports**.

Nous avons ensuite rédigé le **script *create***, permettant de créer une archive, après quoi nous avons intégré cette fonctionnalité aux scripts client et serveur. Il nous a fallu pour cela définir un signal de fin de transmission et un **script de téléchargement** de fichiers.

Cela fait, nous avons intégré le **mode *list*** directement dans les scripts client et serveur.

Il nous a ensuite fallu rédiger le **script *browse* et ses sous commandes**. Nous avons d'abord réalisé les commandes **de consultation**, nous paraissant plus simples (*pwd*, *cd*, *ls*, *cat*), après quoi, forts de l'expérience ainsi acquise, nous nous sommes attelés aux commandes **modifiant l'archive** (*mkdir*, *touch*, *rm*). Au fur et à mesure de la réalisation de ces scripts, nous avons créé des utilitaires permettant par exemple de vérifier l'existence d'un fichier ou d'obtenir le contenu d'un répertoire.

Enfin, nous avons implémenté le mode *browse* côté client et côté serveur. Ce ne fut pas une tâche aisée, car il nous fallut modifier notre système de connexion afin que celle-ci soit maintenue même lorsque l'entrée de la commande *netcat* se retrouvait vide, le serveur se mettant en attente d'une saisie du client pour une durée indéfinie.

Cette partie nous a à tous les deux pris plusieurs jours à réaliser, en faisant ainsi la plus longue phase du développement, d'autant plus que nous avons rencontré des problèmes de compatibilité (voir III.2.7 – *Problèmes de compatibilité*) en travaillant chacun sur des distributions Linux différentes.

S'en est suivie une phase d'améliorations, qui furent apportées par chacun de nous pendant les vacances.

Nous avons notamment apporté des **améliorations au système de connexion** et à l'interface de nos différents scripts, soumis l'accès de chacune des archives à un verrou d'exclusion mutuelle, et créé une fonction pour vérifier l'existence et l'accessibilité d'une archive avant de la parcourir ou de l'extraire.

A l'issue des vacances, nous avons implémenté, puis intégré aux scripts client et serveur, le mode *extract*, sans rencontrer de difficultés particulières pour ce faire.

Pour finir, au cours de la dernière semaine avant la présentation, nous avons principalement testé le logiciel et corrigé des bogues.

2. Difficultés rencontrées

2.1. Ecoute sur plusieurs ports

Lors de la phase de conception du logiciel, nous n'avions pas envisagé l'écoute sur plusieurs ports par le serveur – nous avons seulement compris par la suite que cette fonctionnalité était demandée dans le sujet.

Cet aspect du projet a présenté pour nous un défi conséquent, mais n'a heureusement pas impacté notre stratégie concernant le reste du logiciel.

Pour le réaliser, nous avons mis en place une solution d'écoute où plusieurs instances d'une fonction *listen* réalisant la connexion via *netcat* s'exécutent continuellement et en tâche de fond.

Chaque **port valide** est associé à un **tube nommé** et à un **processus d'écoute**, la conservation de ces éléments étant requise pour une interruption propre du serveur.

$$\overbrace{\{ \{ PORT 3 \} \{ FIFO 3 \} \{ LISTENPID 3 \} \}}^3$$

Figure III – 2.1.a : Exemple d'association d'un port, d'un tube nommé et d'un processus d'écoute.

Un indice entier lie les membres de cette association.

```
echo "Listening on $(eval echo "\${PORT$k}")"'....'
listen $k &
((listening++))
export LISTENPID$k="$!"
```

Figure III – 2.1.b : Appel de la fonction *listen* en tâche de fond avec un indice d'association en argument.

vsh-server.sh, fonction `start`

```
listen () {
    z=0
    while true;
    do
        lport="$(eval echo "\${PORT$1}")"
        lfifo="$(eval echo "\${FIFO$1}")"
        serve < "$lfifo" | "$NETCAT" -l -p $lport > "$lfifo"
    done
}
```

Figure III – 2.1.c : La fonction `listen $k` redirige les données entrantes sur le port `\${PORT$k}` vers le tube `\${FIFO$k}` et envoie au client la sortie de la fonction *serve*.

2.2. Accès aux autres scripts

Pour importer script à partir de son chemin relatif, il nous a fallu situer l'emplacement de l'appelant dans le système de fichiers.

Nous avons d'abord pensé qu'il fallait utiliser `./`, mais ce chemin désigne le répertoire de travail de l'utilisateur.

Le problème a été résolu en utilisant la séquence d'instructions suivante :

```
SCRIPTDIR="$( cd -- "$( dirname -- "${BASH_SOURCE[0]}" )" &>
/dev/null && pwd )/"
```

Figure III – 2.2 : Cette instruction permet à un script d'obtenir le chemin absolu du répertoire dans lequel il se trouve.

2.3. Chemins utilisant des barres obliques inverses comme séparateurs

L'utilisation de barres obliques inverses '`\`' comme séparateur de chemins dans le système de fichiers des archives a grandement compliqué les traitements appliqués sur l'archive.

En effet, des outils tels que *sed*, *grep* ou *awk*, utiles pour extraire des éléments d'une chaîne de caractères ou les traiter, considèrent '`\`' comme un caractère d'échappement.

Nous l'avons donc, avant chaque traitement de chemin, **soit doublé** en remplaçant chaque '`\`' par la séquence '`\\`' au moyen par exemple d'*awk.gsub* :

```
awk 'gsub(/\\/, "\\")'
```

Soit **transformé en barre oblique** '`/`' en utilisant `tr "\\\" /" 2> /dev/null`,

Mais il devient alors nécessaire de retransformer la chaîne avant de l'afficher.

2.4. Contrôle du flux de communication avec le serveur

Il nous a fallu beaucoup de temps avant de parvenir à comprendre comment contrôler le flux de communication entre le client et le serveur.

En effet, la connexion du client, telle qu'elle était initialement mise en place, devait être interrompue manuellement.

Nous avons ensuite eu besoin d'utiliser un *FIFO* vers lequel rediriger la sortie de *netcat* pour traiter la transmission du serveur sans nécessairement l'afficher au client. Cette solution posait le problème inverse : la connexion s'interrompait dès qu'il était vide.

Or, lorsque le client se connectait au serveur en mode *browse*, ce dernier devait pouvoir se mettre en attente d'une entrée pour une durée indéfinie.

Pour remédier à ces problèmes, nous avons donc mis en place plusieurs solutions :

- Un **signal de fin de transmission** envoyé par le serveur pour déconnecter le client à la fin d'un traitement.

```
EOT_SIGNAL="\000\001"
```

Figure III – 2.4.a : Le signal de fin de transmission est composé de caractères rares afin d'éviter un malentendu entre le client et le serveur.

- Un **signal d'invite de saisie**, transmis par le serveur pour indiquer au client qu'une entrée est attendue.

```
PROMPT_SIGNAL="\002\003\004\005\006\007"
```

Figure III – 2.4.b : Le signal d'invite de saisie, transmis par le serveur au client lorsqu'il se met en attente d'une entrée.

- Une boucle, exécutée en tâche de fond par le client, vérifiant à intervalles réguliers que le serveur est toujours joignable, et interrompant l'exécution en cas de rupture de connexion.


```

while true
do
    connection_is_active
    if test $? -ne 0
    then
        disconnect
        break
    else
        if test $established -eq 0
        then
            established=1
        fi
    fi
    sleep 1.25
done &

```

Figure III – 2.4.c : Côté client, une boucle exécutée en tâche de fond permet d’interrompre la connexion en cas de “timeout”.
Script ``vsh-cli.sh``, fonction ``connect``.

``connection_is_active`` vérifie l’état du processus netcat, dont le PID a été retenu.

``disconnect`` interrompt proprement l’exécution et supprime les fichiers temporaires créés par le client.

- Pour l’échange de fichiers, un script de téléchargement (`lib/download.sh`) communiquant un nombre de lignes à télécharger, puis les envoyant à la suite. Ce script est utilisé par le serveur pour télécharger une archive créée par le client, mais aussi par le client pour télécharger une archive à extraire.

2.5. Scripts modifiant l’archive

Les commandes du mode *browse* dont le rôle est de modifier les archives nous ont donné du fil à retordre.

En effet, l’ajout ou le retrait d’une ligne dans l’en-tête de l’archive doit systématiquement être suivi de la mise à jour des pointeurs de fichiers et de la longueur de l’en-tête de l’archive.

De même, en cas de suppression d’un fichier de contenu non-vide, le pointeur de début de ligne de chaque fichier commençant après l’élément supprimé doit être mis à jour.

Pour réaliser ces traitements avec les meilleures performances possibles, nous avons défini une commande *awk* complexe, prenant en entrée l’archive à modifier et écrasant son contenu une fois terminée.

```

awkcommand='( NR>=$HEADERSTART' && NR<=$HEADEREND' && ($0 ~ /\^.* +[^\d][rwx\~]{9} [0-9]+ [0-9]+ [0-9]+)/ ) {for
( NR>=$HEADERSTART' && NR<=$HEADEREND' && ($0 !~ /\^.* +[^\d][rwx\~]{9} [0-9]+ [0-9]+ [0-9]+)/ ) {pr
( ! (NR>=$HEADERSTART' && NR<=$HEADEREND') ) {print}'
echo "$(gawk -F ' ' "$awkcommand" "$ARCHIVE")" > "$ARCHIVE" #adding 3 to each file's line start

```

Figure III – 2.5 : Ajoute 3 à chaque pointeur de fichier de l’en-tête. L’exécution est (relativement) efficace en terme de performances.

Cette solution nous a cependant confronté à un problème de compatibilité (voir *III.2.7 - Problèmes de compatibilité*).

2.6. Exécution du mode extract en tant que root

L’extraction dans un dossier dans lequel l’utilisateur courant ne possède pas les droits d’écriture requiert l’exécution de *vsh* en tant que *root*.

Cependant, les archives extraites de cette façon appartenait ensuite à *root*. Nous avons donc cherché à transférer leur propriété à l'utilisateur appelant *sudo*.

```
if test "$USER" == "root"
then
    owner="$(logname)" #files should be owned by the user calling the script if they run it using sudo
    ownergrp="$(groups $owner |awk '{print $NF}')"
fi

chown -R "$owner:$ownergrp" "$ROOTDIR"
```

Figure III – 2.6.a : Cette séquence d'instructions permet de rendre un utilisateur appelant *sudo* propriétaire de l'archive extraite.

Si ``logname`` vaut *root*, l'arborescence issue de l'extraction appartiendra tout de même à *root*.

```
[f * /]# sudo '/home/f/Bureau/L014/Projet/vsh-lo14/scripts/vsh.sh' -extract 93.10.66.136 9999 mon-archive
Extracting archive mon-archive at '/' ...

Script ran as root !
Ownership of /home/f/Bureau/test
Granted to :
    User      →      f
    Group     →      f
[f * /]# ls -l /home/f/Bureau/test
total 20
drwxr-xr-x 3 f f 4096 31 déc. 16:18 dossier
```

Figure III – 2.6.b : Exemple d'extraction d'une archive par un utilisateur *f* exécutant *sudo*.

2.7. Problèmes de compatibilité

Travaillant sur des distributions de linux différentes (*Linux Manjaro Pahvo* et *Ubuntu Focal Fossa*) avons également été confrontés à des problèmes de compatibilité.

Versions d'awk

La solution utilisée par les commandes du mode *browse* pour mettre à jour les pointeurs de fichiers d'une archive (voir III.2.5 - *Scripts modifiant l'archive*) fonctionnait avec la version d'awk distribuée par *Linux Manjaro Pahvo*, mais pas avec celle qui était installée sur *Ubuntu Focal Fossa*, en raison d'une différence de traitement des expressions régulières.

Les pointeurs de fichier n'étaient donc pas mis à jour par un serveur d'archives exécuté sous *Ubuntu*.

Une fois la cause du problème mise en lumière, il nous a donc fallu remplacer *awk* par *gawk* et ajouter ce programme à la liste des dépendances du logiciel. (voir II.1.3 - *Dépendances*).

Tube nommée bloqué

Suite à l'ajout d'une fonction de vérification de l'existence et de l'accessibilité d'une archive, nous avons constaté qu'un client utilisant *Ubuntu* ne recevait plus aucune communication en provenance du serveur après avoir obtenu le droit d'utiliser une archive.

Nous avons mis beaucoup de temps pour trouver l'origine du problème, mais nous finîmes par comprendre qu'il était causé par une différence dans le traitement des

tubes nommés par les deux distributions.

Nous avons résolu le problème en faisant une tentative d'accès non autorisée au *FIFO* recevant les transmissions du serveur, ce qui a eu pour effet de le débloquent en écriture.

```
exec 33< "$INCOMING" 2> /dev/null
```

Figure III – 2.7 : Le *FIFO* est déjà utilisé en écriture au moment de l'exécution de cette instruction, qui, ne se voyant pas attribuer les droits d'accès, échoue.

Elle a néanmoins l'effet d'empêcher le tube nommé de se fermer en écriture plus tard dans l'exécution du script sous *Ubuntu*.

3. Auto-critique

Forts de l'expérience que nous avons acquise en réalisant ce projet, nous pouvons désormais porter un regard critique sur le code que nous avons produit.

3.1. Pas de script dédié pour *parser* les chemins

Nous n'avons pas créé d'utilitaire dédié pour réaliser les opérations de lecture et de transformation des chemins pour le mode *browse*.

Ces opérations sont donc réalisées directement dans les commandes *cd*, *ls*...

Les mêmes instructions sont donc répétées un nombre important de fois, là où le code aurait pu être factorisé pour permettre notamment une modification simultanée du traitement des chemins par toutes les commandes.

De plus, si les commandes du mode *browse* peuvent traiter les chemins '..' et '.', elles ne sont pas en mesure de traiter des chemins complexes composés de ces symboles, tels que '\dossier\.\sous-dossier\..\toto.txt'.

3.2. Mauvaises performances du mode *browse*

Certaines commandes du mode *browse* font appel à des utilitaires permettant de faire l'assertion de l'existence d'un élément, de son type, ou encore de retourner le contenu d'un répertoire.

Les commandes récursives telles que *mkdir -p 'A\B\C\D'* réalisent de nombreux appels successifs à ces utilitaires, impactant ainsi négativement les performances des opérations de modification des archives.

Il aurait convenu d'éviter les vérifications redondantes : par exemple, si le répertoire *A* a pu être créé, il n'est pas nécessaire de vérifier qu'il existe, que c'est bien un répertoire et qu'il ne contient pas déjà d'élément nommé *B*.

4. Bogues connus

4.1. Interruption du serveur

La mise en place d'un mode d'écoute sur plusieurs port, utilisant plusieurs tubes nommés temporaires et processus lancés en tâche de fond, a grandement complexifié

le nettoyage de l'environnement lors de l'interruption du serveur.

Pour arrêter le serveur convenablement, en supprimant ses fichiers temporaires et en terminant tous les processus lancés en tâche de fond, **il est nécessaire d'utiliser l'instruction *stop* ou le signal *SIGINT* d'interruption (*Ctrl+C*)**.

Arrêter le processus en utilisant *SIGKILL* ne permettra pas de l'interrompre proprement.

4.2. Présence d'un fichier contenant un octet nul

En testant la création d'archives à partir d'arborescences contenant des fichiers au contenu complexe, telles que des fichiers *pdf* ou des fichiers *son*, nous avons constaté que la plupart des fonctionnalités du logiciel se comportaient anormalement en raison d'un octet nul dans l'archive, perturbant ainsi les commandes telles que *read* ou *cat* avec une entrée inattendue et formant ainsi des boucles de lecture infinies.

4.3. Présence d'un fichier contenant des signaux spéciaux

Des signaux spéciaux ont été définis afin de contrôler les flux de communication client-serveur (voir *III.2.4 - Contrôle du flux de communication avec le serveur*).

La présence du signal de fin de transmission 適適麵 dans le nom ou le contenu d'un élément de l'archive peut perturber le mode *browse* en provoquant la déconnexion immédiate du client dès qu'il est rencontré.

La présence du signal d'invite de saisie dans l'archive (ㄗ ㄗㄣ ㄗㄣㄗ ㄗ ㄗ) pose elle aussi problème, car lorsque ce symbole est transmis au client (par exemple, dans le cas où le client désire afficher le nom ou le contenu d'un fichier comportant cette séquence de caractères), il invite le client à saisir une nouvelle commande.

Ces problèmes pourraient être résolus en cryptant les archives.

4.4. Modifications concurrentes d'une archive

Nous avons mis en place un verrou d'exclusion mutuelle interdisant à un client l'accès à une archive déjà en cours d'utilisation par un autre client.

Cependant, il reste possible à l'opérateur du serveur de modifier une archive alors même qu'elle est utilisée par l'un de ses clients, provoquant ainsi des états incohérents ou des plantages.

IV. Conclusion

En conclusion, il convient de noter que la réalisation de ce projet a été l'occasion de perfectionner notre pratique de la programmation *bash* ainsi que notre connaissance théorique des environnements *UNIX*.

Nous avons pu, à travers l'usage d'une grande diversité de commandes, approfondir notre maîtrise des outils vus en cours et en découvrir de nouveaux.

Ce projet a également été riche en enseignements portant sur les notions d'administration des systèmes d'exploitation, en nous confrontant à des problèmes de partage de ressources, d'attribution de droits et de gestion de processus.

La nécessité de mettre en place un système de connexion et la configuration d'un serveur accessible publiquement a aussi constitué un apprentissage de valeur dans le contexte de notre formation d'ingénieurs en Réseaux et Télécommunications.

Le processus de développement du projet s'est dans l'ensemble bien déroulé, et bien que nous ayons rencontré plusieurs écueils majeurs, ceux-ci ont toujours constitué un apprentissage nous permettant de monter en compétence.

La phase de conception du logiciel nous a beaucoup servi, notamment d'un point de vue technique, en nous ayant permis de partager une vue d'ensemble commune de l'architecture du logiciel, mais également d'un point de vue organisationnel, en nous permettant une répartition et une planification efficaces des différentes tâches à réaliser. Si nous avons dû, à plusieurs reprises, revenir sur des choix formulés initialement, notre progression n'a pas été fortement impactée par ces imprévus et notre stratégie globale est restée inchangée.

Dans son état actuel, l'utilitaire répond à l'ensemble des exigences spécifiées dans le sujet, même s'il n'est pas exempt de bogues et pourrait être amélioré à bien des égards.

Parmi les différentes perspectives d'évolution envisagées pour ce logiciel, nous retenons notamment la possibilité d'analyser des chemins plus complexes, la correction des problèmes dus à la présence d'octets nuls dans certains fichiers, ou encore la possibilité d'utiliser le signal d'interruption pour mettre fin à l'écoute par le serveur.

V. Annexes

1. Echantillon d'archive

Vous pouvez télécharger [un échantillon d'arborescence](#) ainsi qu'une [archive](#) créée à partir de cette arborescence afin de disposer d'un jeu de données pour tester l'utilitaire.