

SERVEUR D'ARCHIVE BASH

Le but de ce projet est de créer un serveur d'archive en langage Bash. Sur ce serveur, on retrouvera plusieurs archives sur lesquelles nous pourrions effectuer des opérations via l'interface client.

Une archive est composée de trois parties : une première partie ne contenant que deux nombres correspondant respectivement au numéro de ligne d'où commence la partie « header » et la partie « body ». La partie « header » contenant l'organisation des fichiers et dossiers dans l'archive ainsi que leurs différentes caractéristiques (droits, taille, début dans le body, taille dans le body). La partie « body » quant à elle contient tout le contenu des différents fichiers. Le fichier d'archive a pour extension .arch afin de le démarquer des autres fichiers pour ne pas faire d'erreur lors de la manipulation de ceux-ci.

CONNEXION AU SERVEUR

Le protocole de connexion utilisé est netcat.

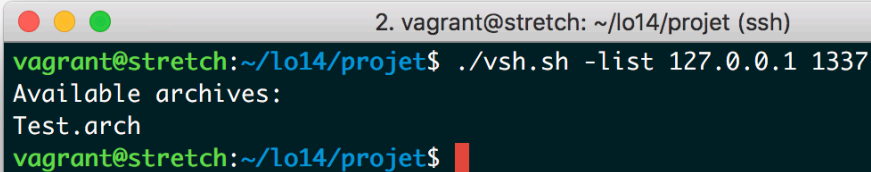
Pour faire fonctionner le serveur il faut lancer le script bash vsh-server.sh en indiquant tout d'abord un numéro de port, suivi du chemin vers le répertoire contenant les différentes archives disponibles.

Côté client, c'est depuis le script vsh que l'on se connecte au serveur. L'adresse IP/nom de domaine du serveur et le numéro sont demandés en paramètre et sont soumis à des vérifications de syntaxes. Le client ne stocke que peu de variable, à savoir l'archive sélectionnée, le répertoire courant dans celle-ci, l'adresse du serveur et le port.

Lorsque le client est lancé, il va tout d'abord vérifier si le serveur est allumé en envoyant une commande au serveur. S'il n'obtient pas de réponse, c'est donc que le serveur n'est pas activé, il affiche alors un message d'erreur. Si, en revanche, la réponse n'est pas vide, il va alors poursuivre ses vérifications en vérifiant que l'archive existe bien sur le serveur, et en exécutant le mode demandé.

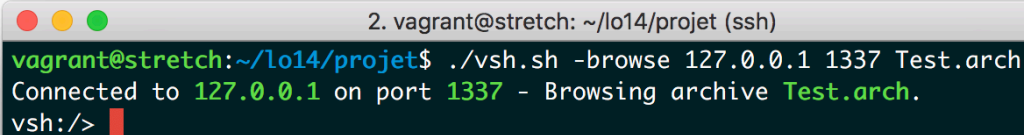
LE MODE LIST

Le mode list nécessite l'adresse IP/nom de domaine du serveur et un numéro de port pour fonctionner. Ce mode permet au client de se connecter au serveur et de lister toutes les archives présentes sur celui-ci (à savoir toutes celles qui se trouvent dans le chemin indiqué lors du lancement de vsh-server.sh). Pour se faire, la fonction liée à la commande list côté serveur va lister les fichiers et sélectionner ceux qui se terminent par .arch avec un grep.

A terminal window titled '2. vagrant@stretch: ~/lo14/projet (ssh)' showing the execution of the vsh.sh script with the -list flag. The output lists the available archives, which is 'Test.arch'.

```
2. vagrant@stretch: ~/lo14/projet (ssh)
vagrant@stretch:~/lo14/projet$ ./vsh.sh -list 127.0.0.1 1337
Available archives:
Test.arch
vagrant@stretch:~/lo14/projet$
```

LE MODE BROWSE

A terminal window titled '2. vagrant@stretch: ~/lo14/projet (ssh)' showing the execution of the vsh.sh script with the -browse flag. It shows a successful connection to the server and the start of browsing the 'Test.arch' file.

```
2. vagrant@stretch: ~/lo14/projet (ssh)
vagrant@stretch:~/lo14/projet$ ./vsh.sh -browse 127.0.0.1 1337 Test.arch
Connected to 127.0.0.1 on port 1337 - Browsing archive Test.arch.
vsh:/>
```

pwd

La commande est plutôt simple. Comme dit précédemment, le client stocke le répertoire courant de l'archive sélectionnée ou plus simplement ça position dans celle-ci. Lorsque l'on exécute la commande pwd, elle renvoie juste la position stockée chez le client.

cd

La fonction `cd` permet de naviguer dans l'arborescence de l'archive. Pour se faire elle a besoin de trois paramètres : l'archive sélectionnée, le chemin vers le répertoire que l'on souhaite accéder et le répertoire dans lequel nous sommes.

Ensuite, la commande analyse le chemin du dossier vers lequel nous voulons nous rendre à travers un case.

Si le chemin ne se compose que d'un /, alors on retourne à la racine.

Si le chemin commence par un /, alors il va appeler la fonction `absolute_path` qui va vérifier la syntaxe du chemin à travers une regex et l'existence du dossier à travers une regex vérifiant l'existence d'une ligne «directory chemin» dans le dossier.

Si le chemin commence par .., la commande va appeler la fonction `relative_backward_path` qui va aussi vérifier l'existence et la bonne syntaxe du chemin. Elle commence par vérifier que l'on commence bien par .. Puis vérifie que le chemin n'est pas composé que de .. et de ../. Elle retire ensuite tous les .. et ../ du chemin pour vérifier que le dossier existe.

Si le chemin résultant est vide, cela veut dire que l'on débute de la racine, sinon on débute du dossier courant.

cat

La commande `cat` permet d'afficher le contenu d'un fichier.

Elle commence par identifier le type de chemin utilisé (absolue, relatif avant ou arrière). Puis que la syntaxe du chemin est correcte et que le dossier contenant le fichier existe dans l'archive. Elle vérifie ensuite que le fichier existe dans le contenu du dossier. Une fois cela fait, elle va chercher la taille du fichier et sa position dans le body. Elle affiche ensuite le contenu de l'archive.

ls

La commande `ls` utilise une fonction interne : `getContentList()`.

Cette fonction ne nécessite pas d'entrées mais utilise des variables locales de la commande `ls`. Elle récupère d'abord le nombre de ligne contenu dans la variable `header` (qui contient le header, mais qui va être modifié afin que de ne contenir qu'une partie spécifique du header). Ensuite, à l'aide d'une boucle `for`, elle va analyser les éléments du header ligne par ligne et appliquer des test dessus à savoir si le fichier est un dossier ou non.

Si oui, elle va ajouter à sa variable `content_list` le nom du dossier collé d'un /.

Si non, elle va ajouter à sa variable `content_list` le nom du fichier.

Elle affiche ensuite le contenu de `content_list`.

Mais avant, d'utiliser cette fonction interne, la commande `ls` fait passer une batterie de test à son entrée (qui contient le chemin vers le dossier dont on veut le contenu) pour savoir si cette entrée mène bien à un chemin existant vers le dossier souhaité que ce soit avec un chemin absolue, qu'avec un chemin relatif avant ou arrière.

rm

Pour la suppression des fichiers, nous avons opté pour la solution suivante : puisque pour accéder au contenu d'un fichier il faut parcourir le body à partir d'une position et pendant une certaine longueur indiquée dans le header, en supprimant la partie du header

contenant ces informations (donc la ligne du fichier concerné) , il est impossible d'accéder au dis fichier et à son contenu.

Cette solution a pour avantage de ne pas avoir à effectuer trop d'opérations sur l'archive quant à la mise à jour des informations du header. Par exemple, supprimer le contenu dans le body nécessite à la fois de supprimer la ligne du header du fichier supprimé et de modifier tous les indices de début de contenu dans le body dans les lignes de fichiers suivants.

La suppression à proprement parler s'effectue en capturant le contenu du header avant la ligne spécifique au fichier supprimé avec un head, en capturant le contenu après avec un tail et en réunissant les deux avec en une seule variable.

Pour la suppression de dossier, une fonction removeFolder va être appelée, cette fonction va alors chercher toutes les lignes correspondantes au dossier à supprimer ainsi qu'à son contenu et le supprimer en conséquence. De plus, si un autre dossier se trouve dans le dossier à supprimer, ce dernier sera lui aussi supprimé, ainsi que son contenu.

stop

La commande sert à éteindre le serveur. Elle atteint son but en tuant le processus lié au seueur via la commande kill.

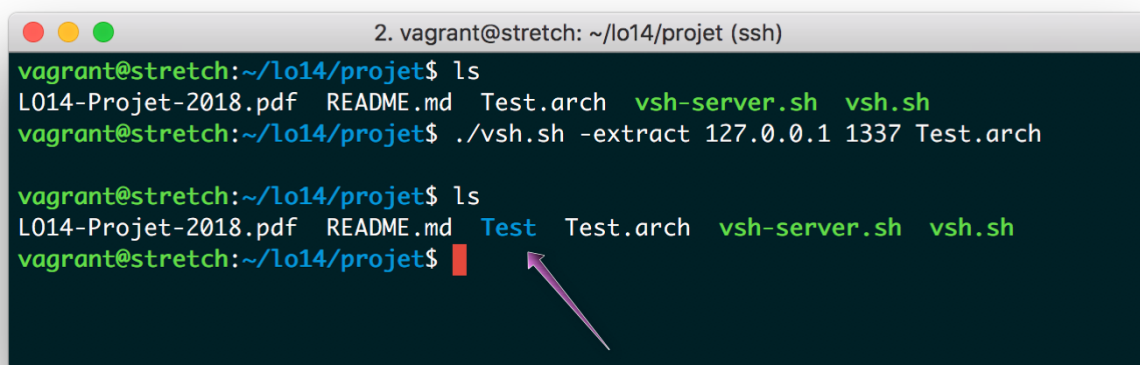
quit

La commande quit est relativement simple. Elle permet de sortir de la boucle d'écoute (l'attente de commandes) de la fonction browse. Se faisant, elle endort le processus pendant une seconde, puis efface l'affichage dans le terminal, et enfin termine normalement le processus avec un exit.

help

La commande help affiche les différentes commande possibles ainsi que leur différents paramètres pour fonctionner et leur utilités.

LE MODE EXTRACT



```
2. vagrant@stretch: ~/lo14/projet (ssh)
vagrant@stretch:~/lo14/projet$ ls
L014-Projet-2018.pdf  README.md  Test.arch  vsh-server.sh  vsh.sh
vagrant@stretch:~/lo14/projet$ ./vsh.sh -extract 127.0.0.1 1337 Test.arch

vagrant@stretch:~/lo14/projet$ ls
L014-Projet-2018.pdf  README.md  Test  Test.arch  vsh-server.sh  vsh.sh
vagrant@stretch:~/lo14/projet$
```

A screenshot of a terminal window titled "2. vagrant@stretch: ~/lo14/projet (ssh)". The terminal shows the execution of the 'ls' command, listing files: 'L014-Projet-2018.pdf', 'README.md', 'Test.arch', 'vsh-server.sh', and 'vsh.sh'. Then, the command './vsh.sh -extract 127.0.0.1 1337 Test.arch' is run. After another 'ls' command, the file 'Test' has been extracted and is now listed alongside 'Test.arch'. A pink arrow points to the newly created 'Test' file.

Le mode extract permet d'extraire le contenu d'une archive.

Pour se faire, le script scinde d'abord le script en deux parties : header et body.

Une fois ces deux parties stockés dans des fichiers (temporaires créés juste pour l'extraction), le script va parcourir le fichier contenant le header ligne par ligne, en ignorant les lignes commençant par « directory ». On se retrouve donc avec des lignes du type <nom> <type><droit> <taille> <début dans le body> <taille dans le body> à traiter pour le script.

Si la partie <type>, extrait avec un ensemble de cut et de sed, est un « d », alors le script va créer un dossier <nom> dans le répertoire courant.

Si la partie <type> est un « - », alors le script va créer un fichier <nom> dans le répertoire courant. Ensuite, le script va analyser la partie <droit> pour convertir les groupes de trois caractères en chiffres, pouvant être un, deux ou quatre, afin d'avoir un nombre à trois chiffres. Que ce soit un fichier ou un dossier, le script va appliquer les droits de l'archive avec la commande « sudo chmod ».

Seulement pour les fichiers, le script va ensuite extraire le contenu de chacun dans body avec un jeu de « tail » et « head » et les parties <début dans le body> <taille dans le body>. Le script va ensuite placer le contenu extrait dans le fichier créé.

Une fois tous le contenu créé (dossiers et fichiers), le script va revenir sur le header et isoler le contenu compris entre une ligne commençant par directory et une ligne commençant par @. Elle va aussi isolé le chemin indiqué sur la ligne directory. Une fois cela fais, le script va simplement bouger les fichiers dans les dossiers auxquels ils appartiennent avec la commande mv.

Une fois finie, l'archive est complètement extraite.