Rapport projet LO14:

Création d'un réseau de machines virtuelles

GAMBET Arthur
PORCHER Hugo
SRT2

Table des matières

Introduction

Ce projet a pour objectif de créer un réseau virtuel de machine linux. Pour ce faire, nous devions créer une nouvelle commande shell, nommée rvsh. Afin de créer cette nouvelle commande, il nous a fallu la coder dans un script BASH, en effet, bien que le shell ne soit qu'un interpréteur de commandes, c'est également un langage de programmation en soi qui permet d'écrire des scripts. Ces derniers sont des fichiers contenant un ensemble de commandes BASH.

La commande rvsh fonctionne selon les deux modes suivants:

- 1er mode: c'est le mode connect. Ce mode s'invoque par la commande suivante
  ./rvsh.bash –connect nom_machine nom_utilisateur
  Cette commande permet de se connecter à une machine virtuelle avec le nom d'un utilisateur.
  Une fois que la connection est lancée, un prompt **utilisateur@machine>** est affiché.

- 2ème mode: c'est le mode admin. Ce mode s'invoque par la commande suivante:
  ./rvsh.bash –admin
  Cette commande permet à l'administrateur de gérer la liste des machines connectées au réseau virtuel et la liste des utilisateurs.
  Une fois que la connection est lancée, un prompt **rvsh>** est affiché.

Nous avons décidé de séparer notre rapport en 2 parties. Dans une première partie, nous tâcherons de décrire le projet et l'architecture que nous avons choisi. Dans la deuxième partie, nous détaillerons toutes les commandes de base, aussi bien du mode connect que du mode administrateur. Nous terminerons ce rapport sur une conclusion qui aura pour but de présenter les difficultés auxquelles nous avons été confronté et ce que nous a appris ce projet.

I - Description du projet

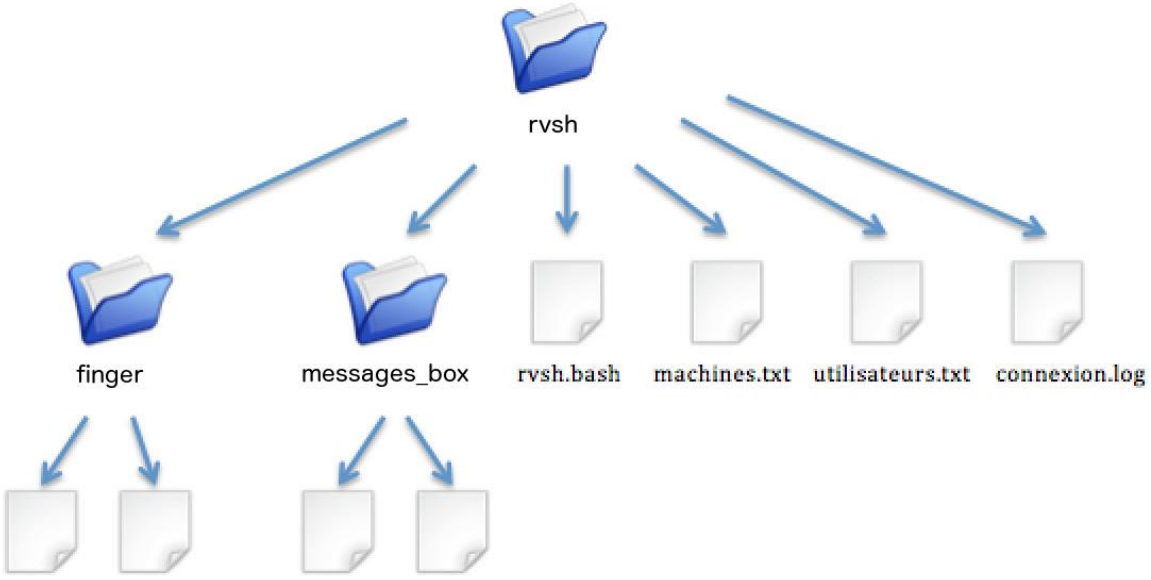
Dans cette partie, nous allons décrire comment nous avons procédé pour faire ce projet puis détailler l'architecture logique de notre commande

a) Contenu du package rvsh

Notre package est constitué du script rvsh (rvsh.bash) ainsi que des fichiers textes représentant les machines et utilisateurs. On compte également un fichier log qui a pour but de faire un journal des connexions au réseau virtuel, un dossier finger contenant les fichiers finger (commentaires) relatifs aux utilisateurs et un dossier messages_box contenant les messages que les utilisateurs peuvent s'échanger à l'aide de la commande *write* (voir 2ème partie sur les fonctionnalités de base).

Les machines sont donc purement virtuelles dans le sens où elles sont définies uniquement par leur nom, ces noms sont stockés dans le fichier *machines.txt*. Concernant les utilisateurs, ces derniers sont également définis uniquement par leur nom, par un mot de passe (qui peut être changé plus tard) et par les machines auxquelles ils peuvent accéder. Toutes ces informations sont stockées dans le fichier utilisateurs.txt. Le fichier *connexion.log* lui, est constitué de plusieurs informations, il y a évidemment le nom de l'utilisateur qui s'est connecté et la machine à laquelle y s'est connecté et également la date et heure à laquelle il s'est connecté. Comme nous l'avons dit plus tôt, le dossier finger contient un fichier texte pour chaque utilisateur existant, ainsi, si l'administrateur décide d'ajouter un nouvvel utilisateur, on crée automatiquement un fichier finger lié à ce dernier. Enfin, le dossier messages_box, comme le dossier finger, contient un fichier texte pour chaque utilisateur. Néanmoins, celui-ci n'est crée qu'à partir du moment où il reçoit un message.

b) Architecture logique de rvsh



II - Fonctionnalités de base

Dans cette partie nous allons détailler une par une les fonctions de notre projet. Pour ce faire, nous avons décidé de partager cette analyse en 3 parties distinctes. Nous verrons d'abord les fonctions principales de rvsh (le main et toutes les fonctions intermédiaires permettant d'accéder au prompt), puis les commandes utilisateurs et enfin les commandes administrateurs.

a) Fonctions principales de rvsh

Nous verrons ici qu'un aperçu du cheminement effectué par le programme, le script rvsh.bash étant très commenté.

Avant d'accéder au prompt, il est évident que l'utilisateur doit vérifier certaines contraintes. Afin que le programme soit le plus optimal possible et puisse faire face aux erreurs humaines, nous avons implémenté une fonction *erreur* ayant pour but de regrouper toutes les erreurs possibles et de les afficher en fonction de son argument passé en entrée. Ainsi, nous avons mis en place plusieurs fonctions permettant de vérifier ce que l'utilisateur écrit et de le corriger si nécessaire.

La première d'entre elles est celle qui vérifie le nombre d'arguments en entrée de la commande, ainsi si l'utilisateur rentre un nombre trop faible ou trop important d'arguments, ce dernier se retrouve face à une erreur lui expliquant sa faute. Il en est de même si l'utilisateur entre une machine ou un utilisateur qui n'existe pas. Ensuite, si l'utilisateur a entré ces derniers paramètres correctement, le programme vérifie si l'utilisateur a bien accès à la machine donnée, si ce n'est pas le cas, un message d'erreur indique encore une fois à l'utilisateur le problème. Enfin, si toutes ces étapes ont été passées, le programme vérifie l'authenticité de l'utilisateur en lui demandant son mot de passe. Pour des raisons de sécurité, nous avons décidé de cacher le mot de passe entré afin que si d'éventuelles personnes regardent par dessus son épaule, ils sont dans l'incapacité de voir son mot de passe. Egalement, afin d'éviter que des malins utilisent un brute force pour venir à bout du mot de passe, nous avons limité le nombre d'essais à 3. Une fois cette étape d'authentification réussie, l'utilisateur peut enfin accéder au prompt.

b) Commandes utilisateurs

Une fois que l'utilisateur a accès au prompt, il peut entrer les commandes principales du programme.

Celles-ci sont au nombre de 8, dans cette partie nous allons les détailler une par une.

- who : permet de voir qui est connecté sur une machine. La commande renvoie le nom de chaque utilisateur, l'heure et la date de sa connection. Dans notre cas, il s'agit simplement de lire le fichier log en supprimant le nom de la machine avec la commande *sed* vu que nous supprimons le log de chaque utilisateur qui se déconnecte, celui-ci contient uniquement les utilisateurs connectés au réseau virtuel.

- rusers : permet de voir la liste des utilisateurs connectés sur le réseau. Elle renvoie le nom de chaque utilisateur et le nom de la machine où il est connecté, ainsi que l'heure et la date de sa connexion. Dans notre cas, il s'agit simplement de lire le fichier connexion.log.

- rhost : renvoie la liste des machines connectées au réseau virtuel. Dans notre cas, il suffit simplement de lire le fichier machines.txt à l'aide d'une boucle *while*.

- connect : permet à l'utilisateur de se connecter à une autre machine du réseau. Dans notre cas, il suffit de faire appel à la commande connect qui vérifie que la machine existe et que l'utilisateur en question peut y accéder. Si la commande s'est bien déroulée, on écrit dans le fichier connexion.log la nouvelle connexion.

- su : permet de changer d'utilisateur. Pour cela, on fait appel à la commande su qui vérifie d'abord si l'utilisateur en question existe bien, puis l'authentifie à l'aide d'un mot de passe. Comme pour connect, une ligne de log est ajoutée si la commande se déroule avec succès.

- passwd : permet à l'utilisateur connecté de changer son mot de passe sur l'ensemble du réseau virtuel. Dans notre cas, on demande le nouveau mot de pase à l'utilisateur puis on remplace l'ancien par celui-ci dans le fichier utilisateurs.txt.

- finger : permet de renvoyer des éléments complémentaires sur l'utilisateur. Toutes ces informations n'existent pas par défaut et sont rentrées par l'administrateur grâce à la commande *afinger* (voir plus bas). La commande finger permet également à l'utilisateur de voir ses messages reçus.

- write : permet d'envoyer un message à un utilisateur connecté sur le réseau. Le message est envoyé avec la syntaxe suivante *write nom_utilisateur@nom_machine message*. Comme nous l'avons expliqué plus haut, le message est écrit dans un fichier propre à l'utilisateur stocké dans les dossier messages_box. Si l'utilisateur destination est connecté en avant-plan dans un autre terminal, il reçoit le message directement sur son terminal. Si il est connecté en arrière-plan il verra le message en tapant la commande finger, comme expliqué au-dessus.

c) Commandes administrateur

Voyons maintenant quelle sont les commandes possibles en mode admin. Pour rappel, le mode administrateur permet uniquement de modifier les informations des utilisateurs et des machines, il ne permet pas de se connecter à une machine.

On compte 3 commandes de base en mode administrateur.

- host : permet à l'administrateur d'ajouter ou d'enlever une machine au réseau virtuel. Dans notre cas, nous avons décidé de séparer, cette commande en 2. Si l'administrateur souhaite ajouter une machine, il doit spécifier l'option–a, ainsi que le nom de la machine qu'il souhaite ajouter. La machine en question est alors ajoutée au fichier machines.txt. Si il souhaite supprimer une machine du réseau

virtuel, il doit spécifier l'option –d avec la machine qu'il souhaite supprimer. Le programme vérifie que le machine existe bel et bien, puis la supprime du fichier machines.txt.

- users : permet à l'administrateur d'ajouter ou d'enlever un utilisateur, de lui donner des droits d'accès à une ou plusieurs machines du réseau ou de lui fixer un mot de passe. Pour ce faire, comme pour la commande *host*, l'administrateur doit spécifier une option, -a si il souhaite ajouter un utilisateur. L'administrateur doit spécifier un mot de passe pour ce dernier, puis celui-ci est ajouté au fichier utilisateurs.txt. Si l'administrateur souhaite supprimer un utilisateur, on vérifie d'abord que celui-ci existe dans le fichier utilisateurs, puis on le supprime de ce dernier. Enfin, si l'administrateur souhaite donner les droits d'accès d'une ou plusieurs machine à un utilisateur, le programme vérifie d'abord que les machines en question existent, puis vérifie si l'utilisateur n'a pas déjà accès à celles-ci, si tout est bon le fichier utilisateurs.txt est mis à jour.

- afinger : permet à l'administrateur de renseigner les informations complémentaires sur l'utilisateur. Dans notre cas, comme nous avons décidé de mettre ces informations dans fichier spécifique à l'utilisateur, nous avons utilisé un éditeur de texte (Vim) pour permettre à l'administrateur de modifier ce fichier. Comme dit plus haut, l'utilisateur accède à ce fichier en utilisant la commande finger.

Conclusion

En conlusion, ce projet nous a permis de mettre en pratique tout ce qu'on a pu apprendre sur la partie Linux dans l'UV LO14. Nous avons pu améliorer considérablement nos réflexes de programmation en BASH, nous avons également pu aller davantage en profondeur sur la manipulation de fichiers puisqu'une bonne partie du projet consistait à écrire dans des fichiers texte puis les lire de différentes façons. Nous avons également pu nous familiariser encore davantage avec l'utilisation de certaines commandes comme *sed* qui ne sont pas si évidentes que ça à maitriser.

Il est sûr que nous nous sommes améliorés tout au long de ce projet, nous avons découvert toutes les possibilités qu'offre le langage BASH en nous documentant principalement sur Internet et dans certains livres. La commande *man* nous a également bien servi à de multiples reprises.

En termes de difficultés rencontrées, la plus grande fut sans doute de créer un historique pour le prompt. C'est à dire, appuyer sur les touches HAUT/BAS afin de récupérer ce qui a été écrit précédemment. Malheureusment, malgrè tous nos efforts nous n'avons pas réussi à l'implémenter. Nous avons également rencontré quelques soucis avec la commande write qui ne fut pas si évidente que ça à coder. En effet, nous avons longuement hésité comment faire pour gérer la réception du message si l'utilisateur est connecté en arriére-plan, nous avons finalement décidé d'utiliser la commande *finger* pour résoudre ce problème.

D'une manière générale, ce projet fut très instructif, et il est fort probable qu'il nous soit très utile pour notre future carrière dans le domaine de l'informatique.