TP d'Administration

TP2 : Terminal distant et premiers scripts BASH

Benoît Da Mota, André Rossi, Vincent Vigneron

Janvier 2018

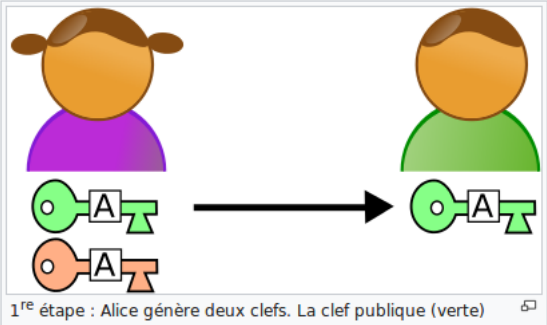
1 SSH : Secure SHELL

Dans de nombreuses situations, il est indispensable de se connecter à distance (réseau local ou internet) à une autre machine, un serveur web ou une machine de développement par exemple. Au département d'informatique de l'Université d'Angers, nous avons plusieurs machines dont certaines sont destinées à la formation des étudiants et au développement. Ainsi, vous possédez un compte (login + password) sur les machines `janus` et `forge`. Pour vous y connecter, il va falloir utiliser un SHELL sécurisé, `ssh`. Regardons la page du manuel :

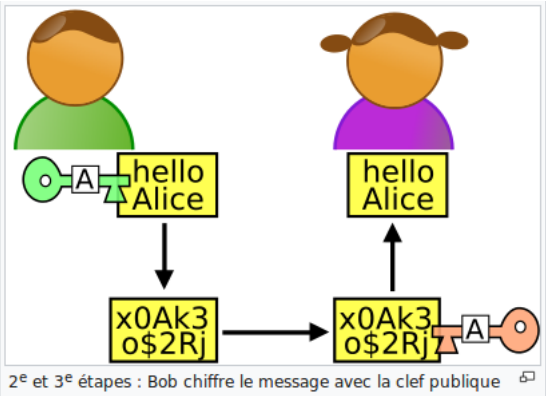
```
etudiant@25B:~$ man ssh
NAME
ssh - OpenSSH SSH client (remote login program)
SYNOPSIS
ssh [-1246AaCfgkMNnqsTtVvXxY ] [-b bind_address ] [-c cipher_spec ] -words [-D port ] [-e escape_char ] [-F configfile ]
     [-i identity_file ] [-L port host hostport ] [-l login_name ] [-m mac_spec ] [-o option ] -words [-p port ] [-R port
     host hostport ] [-S ctl ] [user @ hostname ] [command ]
DESCRIPTION
ssh (SSH client) is a program for logging into a remote machine and for executing commands on a remote machine. It is
     intended to replace rlogin and rsh, and provide secure encrypted communications between two untrusted hosts over an
     insecure network. X11 connections and arbitrary TCP/IP ports can also be forwarded over the secure channel.
...
```

Le manuel fait quelques centaines de lignes et nous renseigne sur de nombreuses options et fonctionnalités. Comme pour de nombreuses commandes, il nous faudrait beaucoup de temps pour couvrir tous les aspects de SSH. La plupart du temps vous consulterez le manuel avec un objectif en tête. Vous pouvez également trouver des tutoriels ou blog posts sur internet, de l'aide sur les forums ou IRC. Les objectifs de cet exercice couvrent des besoins très courants : se connecter à une machine distante, lancer une commande sur une machine distante, transférer des fichiers entre deux machines, connexion en mode graphique (forward X11) et enfin, connexion sans mot de passe. Introduction à la cryptographie par clé publique, cf. Wikipedia :

`https://fr.wikipedia.org/wiki/Cryptographie_asym%C3%A9trique`



1re étape : Alice génère deux clefs. La clef publique (verte) qu'elle envoie à Bob et la clef privée (rouge) qu'elle conserve précieusement sans la divulguer à quiconque.



2e et 3e étapes : Bob chiffre le message avec la clef publique d'Alice et envoie le texte chiffré. Alice déchiffre le message grâce à sa clef privée.

Question 1.1 Connectez-vous à `janus`, lancez quelques commandes et déconnectez-vous.

```
ssh rossi@janus.info.univ-angers.fr
ls
du
exit
```

Question 1.2 Vérifiez l'espace libre de votre home sur `janus` en une seule commande

```
ssh rossi@janus.info.univ-angers.fr df -h .
# ou
ssh rossi@janus.info.univ-angers.fr df -h '$HOME'
# et surtout ne pas utiliser $HOME sans protection
# l'expansion précède l'exécution des commandes
```

Question 1.3 Copiez un fichier depuis votre machine vers votre home sur `janus`.

```
scp ~/monfichier rossi@janus.info.univ-angers.fr:~
```

Question 1.4 Copiez un répertoire depuis votre machine vers votre home sur `janus`.

```
scp -r ~/monrepertoire rossi@janus.info.univ-angers.fr:~
```

Question 1.5 Connecter-vous à `janus` en mode graphique et lancez une application en mode fenêtré.

```
ssh -XY rossi@janus.info.univ-angers.fr
geany &
```

Question 1.6 Générez une paire de clés cryptographiques RSA 2048 bits (par défaut) sans mot de passe.

```
ssh-keygen
```

Question 1.7 Configurez votre compte sur `janus` pour autoriser la connexion avec votre paire de clés. Il faut donc que `janus` « connaisse » votre clé publique.

```
ssh-copy-id -i ~/.ssh/id_rsa.pub rossi@janus.info.univ-angers.fr
```

Question 1.8 Pour vérifier l'absence de mot de passe, faites de nouveau les questions 1.1 à 1.3 (rappel : `history` ou `CTRL+R` pour `reverse-i-search`).

```
# question 1.1
ssh rossi@janus.info.univ-angers.fr
ls
du
exit
# question 1.2
ssh rossi@janus.info.univ-angers.fr df -h .
# question 1.3
scp -r ~/monrepertoire rossi@janus.info.univ-angers.fr:~
```

Conseils :

- regardez les manuels de `ssh`, `scp`, `ssh-keygen`
- cherchez à quoi correspond le fichier `$HOME/.ssh/authorized_keys`
- pensez à la communauté, par exemple : `http://doc.ubuntu-fr.org/ssh`

Question 1.9 Comment peut-on revenir à l'état initial, c'est-à-dire à la situation où la saisie d'un mot de passe est nécessaire pour se connecter à `janus` ?

```
echo "Il suffit de supprimer le répertoire .ssh sur le poste local"
rm ~/.ssh
```

2 Screen : un multiplexeur de terminaux

Nous avons pu voir en cours qu'un processus possède un `PID` et un `PPID`. Les commandes lancées depuis un Secure SHELL n'échappent pas à cette règle. Aussi, si le processus SSH se termine (déconnexion par exemple), les processus enfants seront aussi tués. Un multiplexeur de terminaux permet d'ouvrir plusieurs terminaux dans une même console, de passer de l'un à l'autre et de les récupérer plus tard, voire de les récupérer depuis une autre machine. Screen permet aussi de partager un terminal pour par exemple regarder ce que nous montre un autre utilisateur. Il existe d'autres multiplexeurs de terminaux comme `tmux`. Pour plus d'aide pour l'exercice `man screen` ou `http://doc.ubuntu-fr.org/screen`.

2.1 Exercice/démonstration

Question 2.1 Connectez-vous à `janus` et lancez `screen`.

```
ssh -X -Y rossi@janus.info.univ-angers.fr
echo "Il ne faut SURTOUT PAS appeler un screen \"test\" !"
screen -S machin
```

Question 2.2 Tapez quelques commandes et fermez votre terminal.

```
echo "$HOME"
echo "Ceci est le screen machin"
```

Question 2.3 Reconnectez-vous à `janus`, listez les screens existants et reconnectez-vous au screen de la question 2.1. Que constatez-vous ?

```
ssh -X -Y rossi@janus.info.univ-angers.fr
screen -ls
screen -r machin
echo "On retrouve ce qu'on avait tapé dans la console sous le screen machin.
   screen -r test renvoie \"WriteMessage: Mauvais descripteur de fichier\"..."
```

Question 2.4 Quittez le screen, puis quittez `janus`.

```
<CTRL> + A puis D
exit
```

3 Mes premiers scripts

Quelques exercices simples avec des variables, des tests et des boucles.

3.1 Exercice : saisie de valeur

```
#!/bin/bash

echo -e "\\n   \\t   On déclare la variable locale v1 \\n "
v1=quelque chose
echo -e "\\t La valeur de v1 est : $v1 \\n"

echo -e "\\t  Entrez la nouvelle valeur de v1 : "
read v1
if [[ -z $v1 ]]; then
    echo "La valeur de v1 est vide"
else
echo "Nouvelle valeur de v1 : $v1"
fi
```

Question 3.1.1 Exécutez le fichier `SC1.sh` ci-dessus qui comprend un exemple de script bash.

```
bash SC1.sh
```

Question 3.1.2 Indiquez les variables utilisées dans ce script.

```
v1
```

Question 3.1.3 Testez ce script.

Ce script affiche la valeur courante de `v1` qui a été initialisée à `quelque_chose`, et demande la saisie d'une nouvelle valeur. Enfin, il affiche cette nouvelle valeur, ou indique l'absence de valeur le cas échéant.

Question 3.1.4 Modifiez ce script pour répéter la saisie tant que la variable `v1` est non vide.

```
#!/bin/bash

echo -e "\\n   \\t   On déclare la variable locale v1 \\n"
v1=quelque chose
echo -e "v1 = $v1"

while [[ -n $v1 ]]; do
  echo -e "\\t La valeur de v1 est : $v1 (longueur : ${#v1})\\n"
  echo -e "\\t Entrez la nouvelle valeur de v1 (ENTREE pour arrêter) :"
  read v1
done
```

3.2 Exercice : changement des permissions

Question 3.2.1 Écrivez un script bash rendant un fichier exécutable pour son propriétaire, où le nom du fichier est passé en paramètre au script.

```
#!/bin/bash

if [ -z $1 ]; then
  echo "Ce script doit être appelé avec le nom du fichier à rendre exécutable
     en paramètre."
  exit
fi
chmod u+x $1
```

Question 3.2.2 Améliorez le script précédent pour que l'utilisateur puisse préciser s'il veut changer le droit pour le propriétaire uniquement ou pour tout le monde, à l'aide d'un second argument.

```
#!/bin/bash

if [ -z $1 ] || [ -z $2 ]; then
  echo -e "Ce script doit être appelé avec : "
  echo -e "\\tle nom du fichier à rendre exécutable comme premier paramètre,"
  echo -e "\\tla lettre \"a\" (pour tous) ou la lettre \"u\" (pour le proprié
     taire seulement) comme second paramètre."
  exit
fi

if [ $2 = "a" ] || [ $2 = "u" ]
then
  chmod $2+x $1
else
  echo -e "Erreur, le second paramètre doit être \"a\" ou \"u\""
fi
```

3.3 Exercice : calcul du carré d'un entier

Question 3.3.1 Écrivez un script bash qui affiche le carré d'un entier passé en paramètre.

```
#!/bin/bash

if [ -z $1 ]; then
  echo -e "Erreur, ce script doit être appelé avec un argument entier, qui sera
      élevé au carré."
  exit
fi

let "res = $1 * $1"
echo -e "Le carré de $1 est $res."
# echo -e "Ou encore, le carré de $1 est $(($1 * $1))."
```

Question 3.3.2 Écrivez un script bash qui affiche la liste des carrés des entiers compris dans un intervalle dont les bornes sont passées en paramètre.

```
#!/bin/bash

if [ -z $1 ] || [ -z $2 ]; then
  echo -e "Erreur, ce script doit être appelé avec"
  echo -e "\\tun premier argument entier qui est la borne inférieure"
  echo -e "\\tun second argument entier qui est la borne supérieure"
  echo -e "Le script retournera le carré des entiers compris entre ces bornes."
  exit
fi

if [ $1 -gt $2 ]; then
  echo -e "Erreur, $1 est supérieur à $2."
  exit
fi

for i in `seq $1 $2`; do
  echo -e "Le carré de $i est $(($i * $i))"
done
```

3.4 Exercice : affichage d'informations sur les fichiers

Question 3.4.1 Écrivez un script bash qui teste la nature de l'argument passé en paramètre, affiche son contenu et son nombre de lignes si c'est un fichier et affiche son contenu si c'est un répertoire.

```
#!/bin/bash

if [ -z $1 ]; then
  echo -e "Erreur, ce script doit être appelé avec un argument qui est un
     fichier ou un répartoire à tester."
  exit
fi
```

```
# if [ -e $1 ] is true if $1 exists
if [ -e $1 ] && [ -f $1 ]; then
  echo -e "Le fichier \"$1\" a $(cat $1 | wc -l) lignes :"
  cat $1
elif [ -e $1 ] && [ -d $1 ]; then
  echo -e "Le répertoire \"$1\" a le contenu suivant :"
  ls $1
else
  echo -e "Fichier ou répertoire \"$1\" non trouvé."
fi
```

3.5 Exercice : l'instruction case de bash

Question 3.5.1 Tapez et exécutez le script suivant.

```
#!/bin/bash

clear
essai=1 ;
while [[ 1 ]] ; do
  echo -e " \\n\\n\\n  Menu - Essai : $essai"
  echo " Afficher le répertoire courant          1 "
  echo " Lister les fichiers                     2 "
  echo " Informations sur un fichier             3 "
  echo " Changement de répertoire                4 "
  echo " n premières lignes d'un fichier         5 "
  echo " Sortie                                  0 "
  echo -n " Choix : "

  read choix;
  echo " "

  case $choix in
    1) pwd ;;
    2) ls ;;
    3) ;; # à compléter
    4) ;; # à compléter
    5) ;; # à compléter
    0) exit ;;
    *) echo "Merci de saisir un entier entre 0 et 5" ;;
  esac
  essai=$(( essai + 1 )) ;
done
```

Question 3.5.2 Expliquez le rôle de la variable `essai` en précisant son type.

La variable `essai` est implicitement de type entier, puisqu'on l'incrémente avec une instruction `$(( ))`. Cette variable sert à compter le nombre de « tours de boucle » de la boucle `while`, c'est-à-dire le nombre de fois que l'utilisateur aura exprimé un choix.

Question 3.5.3 Programmez les choix 3, 4 et 5.

```
#!/bin/bash

clear
essai=1 ;
while [ 1 ] ; do
  echo -e " \\n\\n\\n  Menu - Essai : $essai"
  echo " Afficher le répertoire courant          1 "
  echo " Lister les fichiers                     2 "
  echo " Informations sur un fichier             3 "
  echo " Changement de répertoire                4 "
  echo " n premières lignes d'un fichier         5 "
  echo " Sortie                                  0 "
  echo -n " Choix : "

  read choix;
  echo " "

  case $choix in
    1) pwd ;;
    2) ls ;;
    3) echo -e "Entrez le nom du fichier à examiner : "
    read fic
    if [ -n $fic ]; then
      stat $fic
    fi ;;
    4) echo -e "Entrez le nom du répertoire à atteindre : "
    read rep
    if [ -n $rep ]; then
      cd $rep
      pwd
    fi ;;
    5) echo -e "Entrez le nom d'un fichier du répertoire courant, et le nombre
       de lignes à afficher, séparés par un espace : "
    read fic lin
    if [ -n $fic ] && [ $lin -gt 0 ]; then
      head -n $lin $fic
    fi ;;
    0) exit ;;
    *) echo "Merci de saisir un entier entre 0 et 5" ;;
  esac
  essai=$(( essai + 1 )) ;
done
```