

IUT Nancy Charlemagne
Université de Lorraine
2 ter boulevard Charlemagne
BP 55227
54052 Nancy Cedex

Département informatique

Rapport TD Outils-Libres en Latex

Etudiant : Enzo Collot
Année universitaire 2022–2023

Table des matières

1	Efficacité de l'environnement de travail	4
1.1	TD-1 : La souris	4
1.2	TD-2 : Le clavier	5
1.3	TD-3 : Vim et Emacs	6
1.4	TD-4 : Commande history	7
1.5	TD-5 : Alias	8
1.6	TD-8 : Raccourci ZSH	9
1.7	TD-9 : Terminaux	10
2	Client SSH	12
2.1	TD-1-SSH : Utilisation SSH	12
2.2	TD-2-SSH : Authentification Publique	14
2.3	TD-3-SSH : Connexion Automatique	16
2.4	TD-4-SSH : SFTP-SSHFS	18
2.5	TD-5-SSH : Tunnel SSH	21
2.6	TD-6-SSH : Tunnel-Tsock	22
2.7	TD-7-SSH : X11 Forward	23
2.8	TD-8-SSH : Rebonds SSH	24
2.9	TD-9-SSH : Bonus SSH	25
3	Git	28
3.1	TD-1-Git : Initialisation Git	28
3.2	TD-2-Git : Les branches	30
3.3	TD-3-Git : Réintégrer les changements	32

1 Efficacité de l'environnement de travail

1.1 TD-1 : La souris

— Désactiver votre souris au niveau système avec la commande `xinput`

Pour désactiver la souris au niveau système, nous allons utiliser la commande `xinput`.

Je tiens à préciser que je vais effectuer ce test sur mon ordinateur personnel qui fonctionne sous Linux Mint.

Voici ci-dessous, la commande qui permet de désactiver la souris :

```
xinput set-prop "device name" "Device Enabled" 0
```

Dans mon cas, j'ai dû rentrer la commande suivante :

```
xinput set-prop "Logitech Wireless Receiver Mouse" "Device Enabled" 1
```

— Initialiser un fichier dans lequel nous allons lister tous les problèmes d'efficacité rencontrés pendant cette séance.

Priorité	Problème	Correctif
1	Logout difficile au clavier	Raccourci clavier Ctrl+Alt+Suppr/Delete
2	Impossible d'éditer des documents PDF avec Google Drive	Utilisation de LaTeX
3	La souris est bloquée et ne répond plus	Utilisez le raccourci clavier Ctrl+Alt+F1 pour ouvrir une console en mode texte, puis connectez-vous à votre session. Ensuite, utilisez la commande sudo service gdm3 restart pour redémarrer le serveur d'affichage et réinitialiser la souris
4	Impossible d'avoir plusieurs terminaux en parallèle	Sous Terminator, Ctrl+Shift+O pour split le terminal verticalement, Ctrl+Shift+L pour split le terminal horizontalement
5	Accéder au navigateur de fichier	shortcut configurable dans les settings ex : Alt+F
6	Naviguer dans discord sans la souris	Tab pour se déplacer de haut en bas, Shift+Tab pour aller de bas en haut et Ctrl+Tab pour naviguer de gauche à droite
7	La souris ne fonctionne pas sur un ordinateur portable	Utilisez le raccourci clavier Fn+F9 pour activer ou désactiver le pavé tactile, qui peut parfois interférer avec la souris

1.2 TD-2 : Le clavier

— Identifier un site permettant de s'améliorer à taper au clavier.

J'ai trouvé un site qui permet de tester la rapidité de frappe au clavier. Voici le lien ci-dessous :

Site de TapTouch

Pourquoi celui-ci plutôt qu'un autre ?

J'ai choisie ce site car il explique quelques informations sur notre score à la fin et il nous donne des astuces pour nous améliorer.

Inclure quelques screenshots montrant l'interface

Screen de l'interface du site :

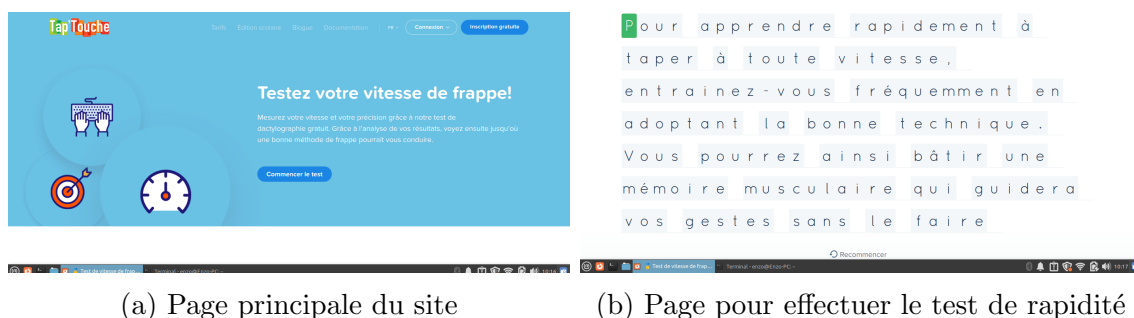


FIGURE 1 – Deux captures d'écran du site TapTouch

Essayer de masquer ses mains pour taper sans regarder

Inclure quelques résultats de rapidité.

Premier test effectué le 22/12/2022 :



Deuxième test effectué le 23/12/2022 :



1.3 TD-3 : Vim et Emacs

- Effectuer les tutoriels pour Vim et Emacs :

Vim : `vimtutor`

Emacs : `emacs`, puis `Ctrl+H+T`

- Paramétrer GNU Readline (le line editor par défaut de bash) pour être en mode Emacs ou Vim

Une fois que j'ai effectué les deux tutoriels, je vais paramétrer GNU Readline en mode Vim. Pour ce faire, je vais utiliser la commande suivante :

```
export EDITOR=vim
```

Après, pour l'avoir de façon permanente, on peut ajouter cette ligne dans le fichier `/.bashrc`.

- Tester les deux 2 modes, en choisir un

J'ai testé ces deux modes et j'ai choisie d'utiliser Vim.

- Paramétrer Emacs ou Vim comme étant éditeur par défaut en plus du mode Readline.

Pour qu'il soit de façon permanente, j'ai ajouté cette ligne suivante dans le fichier `bashrc` comme ci-dessous :

```
export EDITOR=vim
```

1.4 TD-4 : Commande history

— Regarder votre history

Voici ci-dessous, une capture d'écran de mon history :

```
125 cd Documents/cours_serveurs_web/
126 ls
127 cd apache2/
128 ls
129 vim Vagrantfile
130 vagrant up
131 vagrant ssh
132 vagrant halt
133 cd nginx/
134 vim Vagrantfile
135 vagrant up
136 vagrant ssh
137 vagrant halt
138 xinput
139 xinput set-prop "Virtual core XTEST pointer" "Device Enabled" 0
140 xinput
141 xinput set-prop "Logitech Wireless Receiver Mouse" "Device Enabled" 0
142 xinput set-prop "Logitech Wireless Receiver Mouse" "Device Enabled" 1
143 vim
144 export EDITOR=vim
145 ls
146 sudo vim ~/.bashrc
147 history
```

— Y a-t-il des informations sensible ? Comment y remédier ?

Oui, il peut y avoir des informations sensibles dans le history comme les mot de passe etc. Pour remédier à cela, nous pouvons utiliser un autre interpréteur de commande comme ZSH.

— Un employé de longue date semble avoir un **history** très court, quelle sont les causes possibles ?

```
cat ~/.bash_history
```

```
sudo apt install curl zsh git
```

```
sh -c "(curl -fsSL https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
```

On peut constater que cette employé à utiliser un autre interpréteur de commande qui est ZSH en le combinant avec OHMYZSH. Donc on ne peut plus voir son history dans `bash_history`.

— Nos history sont souvent pollués par beaucoup de `ls`, `cd`, `pwd` ... Ces commandes sont tellement courtes que les taper entièrement est plus rapide. On veut éviter qu'elles n'apparaissent dans les résultats de recherche de l'history, comment faire ?

Pour éviter d'avoir certaine commande dans notre history, on peut ajouter l'option `HISTIGNORE` dans le fichier `bashrc` :

```
export HISTIGNORE="&:ls:cd:pwd"
```

Voici le resultat ci-dessous :

```
zeyrox@Zeyrox-PC:~$ cd
zeyrox@Zeyrox-PC:~$ ld
ld: no input files
zeyrox@Zeyrox-PC:~$ pwd
/home/zeyrox
zeyrox@Zeyrox-PC:~$
```

(a) Quelques commandes pour tester HISTIGNORE

```
73  cd
74  history
zeyrox@Zeyrox-PC:~$
```

(b) Résultat dans history

FIGURE 2 – Deux captures d’écran montrant un test de HISTIGNORE

1.5 TD-5 : Alias

— Il est possible de créer des alias avec des arguments à l’aide des bash functions :

Ecrire une bash function mkcd mon dossier qui crée le dossier mon dossier puis navigue dedans.

Voici les étapes ci-dessous pour créer une bash function mkcd :

```
enzo@Enzo-PC:~$ function mkcd() { mkdir -p "$@" && cd "$_"; }
enzo@Enzo-PC:~$ alias mkcd="mkcd"
enzo@Enzo-PC:~$ mkcd test
enzo@Enzo-PC:~/test$
```

Ecrire une bash function gitemergency qui add, commit, et push tout son travail sur l’origine, permettant de ne rien perdre en cas d’alerte d’incendie.

Voici les étapes ci-dessous pour créer une bash function gitemergency :

```
enzo@Enzo-PC:~$ function gitemergency() { git add .; git commit -m "Commit rapide"; git push -u origin; }
```


1.6 TD-8 : Raccourci ZSH

- Dans zsh, créer un raccourci **Ctrl + Shift + A** :
 - Lance les services apache et mariadb
 - Log dans le terminal quand tout est lancé
 - Ce raccourci est un interrupteur : apache et mariadb s'arrêtent si j'appuie à nouveau dessus.

Pour répondre à la question, nous allons tout d'abord créer un script bash comme ci-dessous :

```
#!/bin/bash

if [[ $1 == "start" ]]; then
    echo "Starting Apache and MariaDB..."
    sudo service apache2 start
    sudo service mysql start
    echo "Apache and MariaDB started."
else
    echo "Stopping Apache and MariaDB..."
    sudo service apache2 stop
    sudo service mysql stop
    echo "Apache and MariaDB stopped."
fi
```

Maintenant que nous avons créé le script `.sh`, nous allons créer deux fonctions dans le fichier `.zshrc` pour faire fonctionner le script comme ci-dessous :

```
function start_services() {
    echo "Starting Apache and MariaDB..."
    ~/services.sh start
    echo "Apache and MariaDB started."
}

function stop_services() {
    echo "Stopping Apache and MariaDB..."
    ~/services.sh stop
    echo "Apache and MariaDB stopped."
}
```

Maintenant que nous avons créé les deux fonctions, nous allons les combiner dans le fichier `.zshrc` comme ci-dessous :

```
function toggle_services() {
    if [[ $SERVICES_RUNNING == "true" ]]; then
        stop_services
        unset SERVICES_RUNNING
    else
        start_services
        export SERVICES_RUNNING=true
    fi
}

bindkey '^A' toggle_services
```

Il nous reste plus qu'à utiliser la commande `source ~/.zshrc` pour que les modifications soient prises en compte.

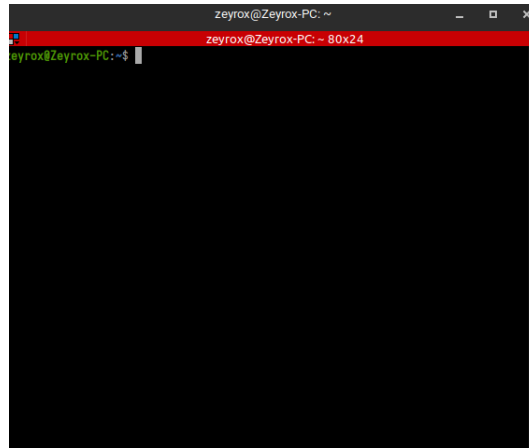
1.7 TD-9 : Terminaux

— Installez et essayez 3+ émulateur de terminaux parmi la liste précédente :

Pour ma part, je vais installer Terminator, Kitty et Tilix. Les installations des différents terminaux sont en dessous :

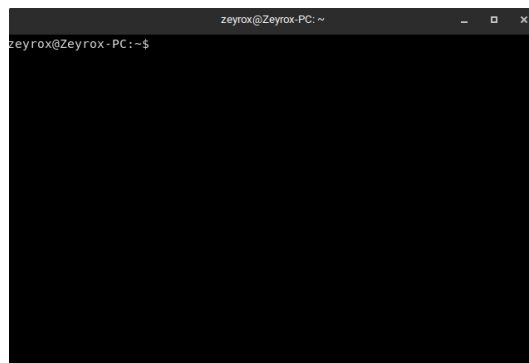
Terminator : `sudo apt-get install terminator`

Image de Terminator :



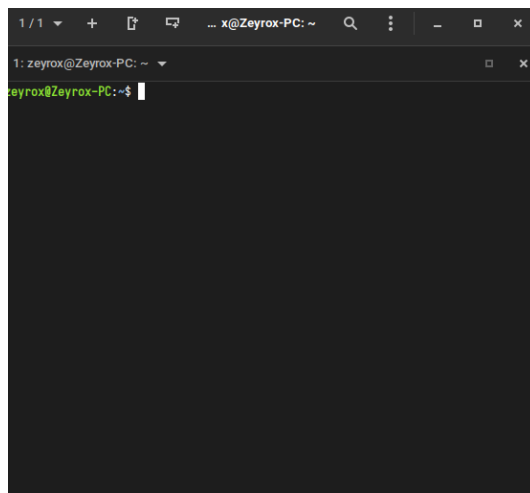
Kitty : `sudo apt-get install kitty`

Image de Kitty :



Tilix : `sudo apt-get install tilix`

Image de Tilix :



- Choisir un terminal et justifier pourquoi
 - Fonctionnalités
 - Rapidité
 - Documentation/Ecosystème

J'ai décidé de choisir terminator car il offre beaucoup de possibilités. Au niveau des fonctionnalités, Terminator permet d'avoir plusieurs terminaux dans un seul terminal (Multiplexage). On peut créer des onglets, définir des raccourcis et bien plus encore.

Ensuite, en ce qui concerne la rapidité, terminator est un terminal très rapide et réactif.

Et pour finir, Terminator a une communauté très active et il est très bien documenté. Donc on peut conclure que dans l'ensemble, Terminator est un bon choix de terminal en raison de ses fonctionnalités, de sa rapidité et de sa très bonne documentation. C'est pour cela que j'ai décidé d'utiliser Terminator.

Et de mon point de vue, j'ai toujours utilisé terminator car je trouve qu'il est facile à utiliser et une fois que l'on connaît les différentes combinaisons de touches, il peut devenir très puissant.

- Désinstaller les autres terminaux

Pour désinstaller les autres terminaux, nous allons utiliser la commande suivante :

`sudo apt-get remove (le nom de terminal)`. Par exemple : `sudo apt-get remove tilix`.

2 Client SSH

2.1 TD-1-SSH : Utilisation SSH

— Récupérer et démarrer l’environnement Vagrant sur arche ?

Pour ce nouveau TD, nous allons utiliser Vagrant pour utiliser SSH. Pour ce faire, nous allons télécharger depuis Arche le fichier Vagrant. Une fois que le fichier Vagrant est téléchargé, nous allons exécuter la machine Vagrant avec la commande ci-dessous :

```
enzocollot@MacBook-Air-de-Enzo vagrant % vagrant up  
Bringing machine 'cli' up with 'virtualbox' provider
```

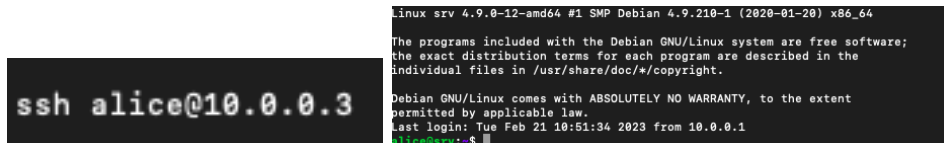
Une fois que la commande est terminée, on doit avoir ceci comme message qui indique que les machines sont bien démarrées :

```
==> cli: Machine 'cli' has a post 'vagrant up' message. This is a message  
==> cli: from the creator of the Vagrantfile, and not from Vagrant itself:  
==> cli:  
==> cli: Vanilla Debian box. See https://app.vagrantup.com/debian for help and b  
up reports  
  
==> srv: Machine 'srv' has a post 'vagrant up' message. This is a message  
==> srv: from the creator of the Vagrantfile, and not from Vagrant itself:  
==> srv:  
==> srv: Vanilla Debian box. See https://app.vagrantup.com/debian for help and b  
up reports
```

— Se connecter à la machine srv avec les utilisateurs alice, bob et carole sans utiliser vagrant ssh ?

Pour se connecter à la machine srv avec les utilisateurs alice, bob et carole sans utiliser vagrant ssh, nous allons utiliser la commande suivante :

Connexion à l'utilisateur alice en ssh :



```
ssh alice@10.0.0.3  
  
Linux srv 4.9.0-12-amd64 #1 SMP Debian 4.9.210-1 (2020-01-20) x86_64  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Feb 21 10:51:34 2023 from 10.0.0.1  
alice@srv:~$
```

(a) Connexion SSH avec l'utilisatrice Alice

(b) Connexion Réussie avec l'utilisatrice Alice

FIGURE 3 – Deux captures d’écran montrant une connexion en ssh au serveur srv avec l’utilisatrice Alice

Maintenant que nous avons réussi à nous connecter au serveur ”srv” avec l’utilisateur ”Alice”, nous allons faire la même chose pour les deux autres utilisateurs.

Connexion à l'utilisateur bob en ssh :

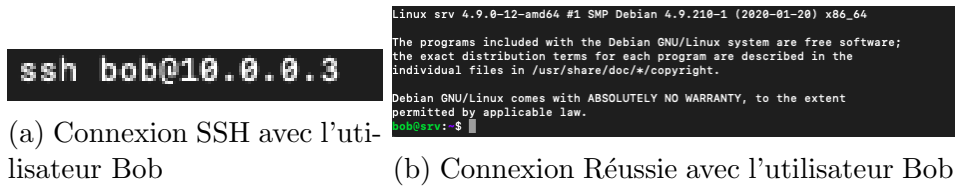


FIGURE 4 – Deux captures d'écran montrants une connexion en ssh au serveur srv avec l'utilisateur Bob

Connexion à l'utilisateur carol en ssh :

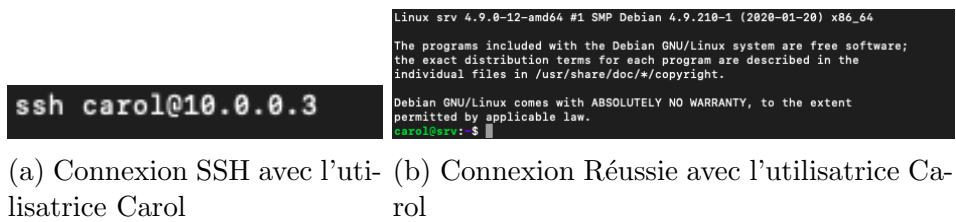


FIGURE 5 – Deux captures d'écran montrants une connexion en ssh au serveur srv avec l'utilisatrice Carol

— Vérifier qu'on est bien sur la machine Vagrant et pas en local ?

Pour vérifier que nous sommes bien sur la machine vagrant et non en local, nous allons utiliser la commande suivante :

```
carol@srv:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:8d:c0:4d brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe8d:c04d/64 scope link
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:9c:cc:58 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.3/24 brd 10.0.0.255 scope global eth1
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe9c:cc58/64 scope link
        valid_lft forever preferred_lft forever
```

On peut constater qu'au niveau des adresses IP, nous avons celle du serveur "srv", donc nous sommes bien sur la machine "vagrant".

— Consulter l' history local, que remarque-t-on ?

Voici, ci-dessous l'history local :

```
390 vagrant up
391 vagrant ssh
392 vagrant ssh srv
393 ssh alice@10.0.2.15
394 ssh alice@10.0.0.3
395 ssh alice@10.0.0.3
396 ssh bob@10.0.0.3
397 ssh carol@10.0.0.3
398 vagrant ssh srv
399 ssh carol@10.0.0.3
enzocollot@MacBook-Air-de-Enzo vagrant %
```

On peut constater que nous ne voyons pas les mots de passe que nous avons entrés pour nous connecter au serveur srv avec un utilisateur.

2.2 TD-2-SSH : Authentification Publique

— Créer une paire de clés privée et publique à l'aide de ssh-keygen ?

Pour cette nouvelle étape, nous allons créer une paire de clés privée et publique à l'aide de ssh-keygen. Donc voici les étapes ci-dessous pour le faire :

Pour commencer, nous allons ouvrir un terminal et taper la commande suivante :

```
enzocollot@MacBook-Air-de-Enzo ~ % ssh-keygen
```

Ensuite, il nous demande à quel emplacement nous voulons enregistrer notre clé. Nous laissons par défaut :

```
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/enzocollot/.ssh/id_rsa):
```

Ensuite, il nous demande une passphrase. Vous pouvez en indiquer une, mais j'ai laissé par défaut. Une fois que c'est fini, vous devez avoir cela qui devrait apparaître :

```

The key's randomart image is:
+---[RSA 3072]-----+
|
|   o.. .
|..   . o.+
|o.=   E ++
|+* .   +=
|o.=   . . S.
|+o o . . o
|.o=   ... . .
| .**o+o+.
|===+o+.o.
+---[SHA256]-----+
anzocollot@MacBook-Air-de-Enzo ~ %

```

Voilà, nous avons générer une clé rsa.

- Utiliser la commande `ssh-copy-id` pour déposer la clé publique sur le compte `alice@cli`. Vérifier qu'on peut maintenant se connecter sans mot de passe?

Pour cette nouvelle étape, nous allons utiliser la commande `ssh-copy-id` pour déposer la clé publique sur le compte `alice@cli`. Pour ce faire, nous allons utiliser la commande suivantes :

```
enzo@collo:~$ ssh-copy-id -i .ssh/id_rsa.pub alice@10.0.0.2
```

Une fois que nous avons entrée la commande, il va nous demander le mot de passe de Alice. Une fois le mots de passe entré, nous avons ceci qui apparaît :

```
alice@10.0.0.2's password:
Number of key(s) added:      1

Now try logging into the machine, with: "ssh 'alice@10.0.0.2'"
and check to make sure that only the key(s) you wanted were added.
```

Maintenant, on peut se connecter en compte d'alice sur le serveur cli sans rentrer son mots de passe.

- Déposer manuellement la clé publique sur le compte bob@cli
Les nouvelles clés créés sont dans `/.ssh`
Les clés autorisées à se connecter au serveur sont dans `/.ssh/authorized_keys`

Pour déposer manuellement la clé publique sur le compte bob@cli, vous pouvez suivre les étapes ci-dessous :

Pour commencer, nous allons afficher le contenu de notre clé publique et le copier avec la commande suivante :

[illegible]

Une fois que nous avons copier la clé publique, nous allons nous connecter au compte de bob@cli avec la commande suivante :

```
ssh bob@cli
```

Une fois connecter sur le compte, nous allons verifier si le dossier .ssh existe. Si il existe pas, nous le créons avec la commande suivante :

```
mkdir /.ssh
```

Ensuite, on tape la commande suivante pour créer ou ouvrir le fichier "authorized_keys" :

```
nano /.ssh/authorized_keys
```

Maintenant, on copie le contenu de la clé publique dans le fichier comme ci-dessous :



```
ssh -i AAAAB3NzaC1yc2EAAAADAQABAAQGD1T6VEM6i1XZDe++1ePugFetBq9w1K1HgZokx8VAbm2bV8BT0uFOnADx9GB6xx3m0Z3mxyU0x08A+nLQJ6
```

Une fois que nous avons enregistré le fichier, c'est bon, on peut se connecter au compte de bob@cli sans taper le mot de passe avec la commande suivante :

```
ssh -i /.ssh/id_rsa.pub bob@cli
```

2.3 TD-3-SSH : Connexion Automatique

— Purger le fichier /.ssh/known_hosts si nécessaire ?

Pour cette nouvelle étape, nous allons purger le fichier /.ssh/known_hosts avec la commande suivante :

```
sudo rm known_host
```

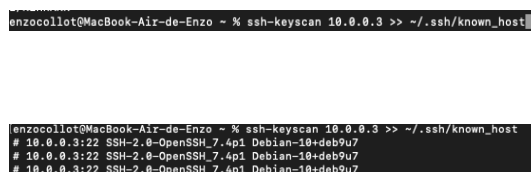
— A l'aide de ssh-keygen et ssh-keyscan , ajouter la clé publique du serveur srv manuellement, décrire les étapes.

Votre client SSH ne doit pas demander d'ajouter la clé publique du serveur à la première connexion.

Voici les étapes pour ajouter manuellement la clé publique du serveur srv au fichier known_hosts à l'aide des commandes ssh-keygen et ssh-keyscan :

Tout d'abord, on va utiliser la commande ssh-keyscan pour récupérer la clé publique du serveur srv. Dans un terminal, on va utiliser la commande suivante :

```
ssh-keyscan 10.0.0.3 >> /.ssh/known_hosts
```



```
enzocollot@MacBook-Air-de-Enzo ~ % ssh-keyscan 10.0.0.3 >> ~/.ssh/known_hosts
# 10.0.0.3:22 SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u7
# 10.0.0.3:22 SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u7
# 10.0.0.3:22 SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u7
```


Ensuite, on va utiliser la commande `ssh-keygen` pour générer une paire de clés SSH (une clé publique et une clé privée) sur l'ordinateur local. Dans le même terminal, tapez la commande suivante :

`ssh-keygen`

```
Your identification has been saved in /Users/enzocollot/.ssh/id_rsa.
Your public key has been saved in /Users/enzocollot/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:qCtBemMrnjZji7monZ8SU6Ep3oxchP3pYzhXaufkZiE enzocollot@MacBook-Air-de-Enz
b.local
The key's randomart image is:
[RSA 3072]-----
  O...
  +.+O.
  .+E+.
  ..BB..
  O=.. . S
  .+B= .
  O *.
  +B= . o
  @+++
  -----[SHA256]-----
```

Une fois que nous avons généré une paire de clés SSH, nous allons copier la clé publique sur le serveur `srv`. Pour ce faire, nous allons entrer la commande suivante dans le terminal :

`ssh-copy-id -i ~/.ssh/id_rsa.pub user@srv`

Ici, `user@srv` on remplace par les utilisateur sur le serveur `srv`. un petit exemple ci-dessous :

```
enzocollot@MacBook-Air-de-Enzo ~ % ssh-copy-id -i ~/.ssh/id_rsa.pub bob@10.0.0.3
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: '/Users/enzocollot/.ssh/id_rsa.pub'
The authenticity of host '10.0.0.3 (10.0.0.3)' can't be established.
RSA key fingerprint is SHA256:FkA9d2hu0zmtkyx0p0H2u0ufu0KtVP0h0Y0M.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already i
nstalled
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- If you are prompted now it is to install th
e new key(s)
bob@10.0.0.3's password:
Number of key(s) added: 1
Now try logging into the machine, with: "ssh 'bob@10.0.0.3'"
and check to make sure that only the key(s) you wanted were added.
```

Voilà, maintenant, on peut se connecter automatique avec l'utilisateur `bob` au serveur `srv`.

— Créer un fichier de configuration pour SSH, vous permettant de vous connecter sur le compte `bob@cli` en tapant `ssh bc`.

Voici les étapes pour créer un fichier de configuration pour SSH qui permettra de se connecter au compte `bob@cli` en tapant simplement `ssh bc` :

Pour commencer, nous allons taper la commande suivante pour créer un nouveau fichier de configuration SSH :

`nano ~/.ssh/config`

```
enzocollot@MacBook-Air-de-Enzo ~ % nano ~/.ssh/config
```

Dans le fichier de configuration, ajoutez les lignes suivantes :

```
Host bc
HostName 10.0.0.2
User bob
█
```

Une fois que nous avons mis les informations dans le fichier de configuration, nous pouvons nous connecter comme ci-dessous :

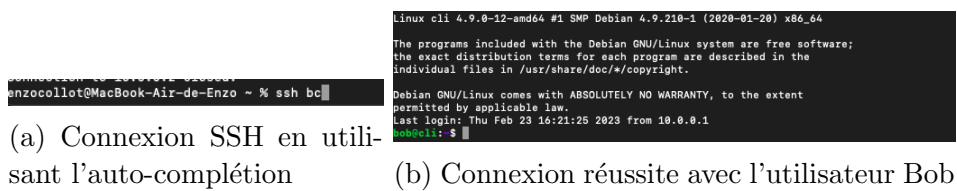


FIGURE 6 – Deux captures d'écran montrant une connexion en ssh au serveur srv avec l'utilisateur bob en utilisant l'auto-complétion

2.4 TD-4-SSH : SFTP-SSHFS

- Vérifiez que vous arrivez à vous connecter au compte `alice@cli` en utilisant SFTP. Copiez-y un fichier par SFTP, puis récupérez un autre fichier. ?

Pour cette nouvelle étape, nous allons nous connecter avec l'utilisateur `alice` en utilisant SFTP. Pour ce faire, nous allons ouvrir un terminal et taper la commande suivante :

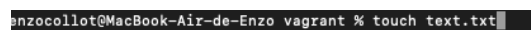
```
sftp alice@10.0.0.2
```



On peut constater que nous sommes bien connectés en sftp au serveur cli avec l'utilisateur Alice.

Maintenant que nous sommes connectés, nous allons copier un fichier de notre machine locale vers la machine virtuelle. Pour ce faire, nous allons créer un fichier texte et l'envoyer sur le serveur. Voici les étapes ci-dessous :

Pour commencer, nous allons créer un fichier texte avec la commande suivante :



Une fois que nous avons créé le fichier, on se reconnecte au serveur avec la commande que nous avons vue précédemment. Une fois connecté, nous allons utiliser la commande suivante :

```
sftp> put text.txt
```

```
Uploading text.txt to /home/alice/text.txt
text.txt      100% 0 0.0KB/s 00:00
sftp> ls
text.txt
```

On peut constater que notre fichier a bien été envoyé sur le serveur.

Maintenant que nous avons envoyé un fichier de notre machine locale vers la machine virtuelle, nous allons prendre un fichier de la machine virtuelle et nous allons le télécharger sur la machine locale. Pour ce faire, nous allons utiliser la commande suivante :

```
sftp> get /etc/apt/sources.list
```

```
Fetching /etc/apt/sources.list to sources.list
/etc/apt/sources.list 100% 446 92.2KB/s 00:00
sftp> exit
enzocollot@MacBook-Air-de-Enzo vagrant % ls
README.md Vagrantfile sources.list srv text.txt
enzocollot@MacBook-Air-de-Enzo vagrant %
```

On peut constater que nous avons bien reçu le fichier du serveur sur notre machine locale.

- Vérifiez que vous arrivez à vous connecter au compte `alice@cli` avec SSHFS.
- Éditez un fichier distant avec un éditeur graphique (par exemple GVIM)
- Vérifiez que la modification est bien répercutée sur la machine virtuelle `alice@cli`.

Pour cette nouvelle étape, je vais répondre aux deux questions en même temps. Pour commencer, nous allons nous connecter au compte `alice@cli` avec SSHFS. Pour ce faire, nous allons utiliser la commande suivante :

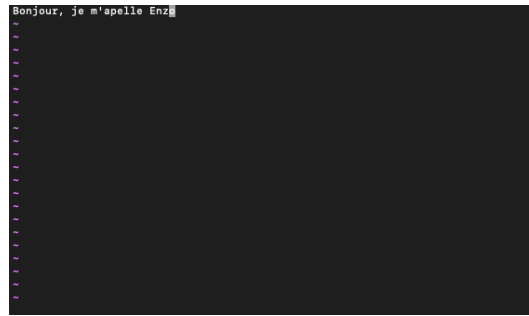
```
sshfs alice@10.0.0.2:/home /tmp/alicecli
```

Vérification du montage dans le dossier `/tmp/alicecli` :

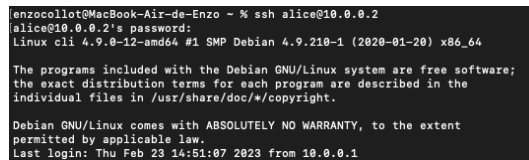
```
enzocollot@MacBook-Air-de-Enzo alicecli % ls
alice bob carol vagrant
enzocollot@MacBook-Air-de-Enzo alicecli %
```

On peut constater que le montage a bien été réalisé.

Maintenant, nous allons modifier un fichier avec vim et regarder si les modification se repercute quand on se connecte directement au serveur. Pour ce faire, nous allons créé un fichier avec vim et mettre des informations dedans comme ci-dessous :



Une fois que nous avons créé et modifié le fichier, nous allons nous connecter au serveur avec alicia comme ci-dessous :



Une fois que nous sommes connectés, nous allons regarder les informations dans le fichier comme ci-dessous :



On peut constater que notre fichier à bien été modifier.

2.5 TD-5-SSH : Tunnel SSH

- Créer un tunnel SSH entre votre poste et srv à travers la machine cli ?

Pour cette nouvelle étape, nous allons créer un tunnel SSH entre mon poste et srv à travers la machine cli. Pour ce faire, nous allons ouvrir un terminal et taper la commande ci-dessous :

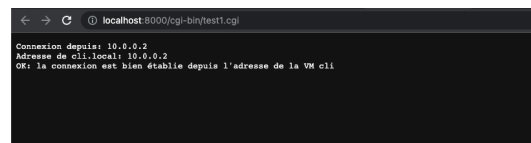
```
enzocollot@MacBook-Air-de-Enzo ~ % ssh -L 8000:10.0.0.3:80 alice@10.0.0.2
```

Maintenant que nous avons créé notre tunnel ss, nous pouvons passer à la suite.

- Tester en ouvrant un navigateur sur votre poste local à l'adresse `http://localhost:8000/cgi-bin/test1.cgi`.
- Pour srv, le navigateur devrait donc provenir de cli, et pas de votre poste local. Si c'est OK, vous devriez avoir le message « OK : la connexion est bien établie depuis l'adresse de la VM cli »

Ici, je vais répondre directement aux deux questions, maintenant nous allons tester la connexion en nous rendant à l'adresse suivante : `http://localhost:8000/cgi-bin/test1.cgi`.

Voici ci-dessous, le résultat de la page :



A screenshot of a web browser window. The address bar shows 'localhost:8000/cgi-bin/test1.cgi'. The page content displays the following text: 'Connexion depuis: 10.0.0.2', 'Adresse de cli: local: 10.0.0.2', and 'OK: la connexion est bien établie depuis l'adresse de la VM cli'.

On peut constater que nous avons bien le message « OK : la connexion est bien établie depuis l'adresse de la VM cli ». Donc notre tunnel SSH fonctionne correctement.

2.6 TD-6-SSH : Tunnel-Tsock

- Se connecter en SSH sur srv .

Pour cette nouvelle étape, nous allons nous connecter au serveur srv via SSH comme ci-dessous :

```
enzocollet@MacBook-Air-de-Enzo ~ % ssh alice@10.0.0.3
alice@10.0.0.3's password:
Linux srv 4.9.0-12-amd64 #1 SMP Debian 4.9.210-1 (2020-01-20) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Feb 23 13:42:25 2023 from 10.0.0.1
alice@srv:~$
```

Maintenant que nous sommes connectés au serveur srv, nous pouvons passer à la suite.

- Se connecter à cli en permettant à cli d'accéder à srv via un tunnel accessible par le port 9000.

Pour se connecter au serveur cli en permettant à cli d'accéder à srv via un tunnel accessible par le port 9000, nous allons créer un tunnel SSH comme ci-dessous :

```
enzocollet@MacBook-Air-de-Enzo ~ % ssh -R 9000:10.0.0.3:22 alice@10.0.0.3
```

Maintenant que nous avons créé le tunnel SSH, nous allons passer à la suite.

- Vérifier que vous arrivez à accéder au serveur web : `wget -nv -O - http://localhost:9000/cgi-bin/test2.cgi`.

Pour vérifier, nous allons utiliser la commande `wget -nv -O - http://localhost:9000/cgi-bin/test2.cgi` comme ci-dessous :

```
alice@srv:~$ wget -nv -O - http://localhost:9000/cgi-bin/test2.cgi
SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u7
Protocol mismatch.
2023-02-24 10:37:05 URL:http://localhost:9000/cgi-bin/test2.cgi [58] -> "-" [1]
```

On peut constater que nous arrivons à accéder à la page web.

2.7 TD-7-SSH : X11 Forward

- Installer une application graphique sur cli (par exemple xclock ou xeyes ,disponibles dans le paquet x11-apps), ainsi que xauth (disponible dans le paquet xauth).

Pour cette nouvelle étape, nous allons une application graphique sur le serveur cli. Pour ce faire, nous allons nous connecter sur le serveur cli et installer les application suivante :

```
sudo apt-get install x11-apps
sudo apt-get install xauth
```

Maintenant que nous avons installer ces deux paquets, nous pouvons passé à la suite.

- Se connecter sur cli en activant le forwarding X11.

Pour ce connecter sur cli en activant le forwarding X11, nous allons utiliser la commande ci-dessous :

```
enzocollet@MacBook-Air-de-Enzo vagrant % ssh -X alice@10.0.0.2
```

Une fois que nous somme connecter, nous pouvons passer à la suite.

- Vérifier que vous pouvez lancer les applications graphiques installées et que les GUI s'affichent sur le poste local.

Nous allons lancer une application graphique via le terminal. Donc voici les étapes pour lancer l'application graphique :

```
alice@cli:~$ xeyes
```



Voila, nous avons réussi à lancer une application graphique X11 depuis le terminal.

2.8 TD-8-SSH : Rebonds SSH

- Configurer `.ssh/config` avec ProxyJump de manière à pouvoir rebondir automatiquement sur cli lorsque vous vous connectez à srv.

Dans cette nouvelle étape, nous allons configurer ProxyJump de manière à pouvoir rebondir automatiquement sur cli lorsque l'on se connecte à srv. Pour ce faire, nous allons créer et modifier le fichier config comme ci-dessous :

```
connection to 10.0.0.3 closed
enzocollet@MacBook-Air-de-Enzo vagrant % nano ~/.ssh/config
enzocollet@MacBook-Air-de-Enzo vagrant %
```



```
GNU nano 2.0.6      File: /Users/enzocollet/.ssh/config
Host srv
  Hostname 10.0.0.3
  User bob
  ProxyJump cli
Host cli
  Hostname 10.0.0.2
  User bob
```

Maintenant que nous avons configuré le fichier config, nous pouvons passer à la suite.

- Vous pouvez utiliser la commande `who` pour vérifier de quelle adresse vous connectez.

Maintenant que nous avons configuré ProxyJump, nous allons nous connecter au serveur srv. Une fois connecté, nous allons utiliser la commande `who` ci-dessous :

```
bob@srv:~$ who
bob      pts/0      2023-02-24 13:42 (10.0.0.2)
```

On peut constater que nous sommes connectés au serveur srv en passant par le serveur cli. Donc le rebond c'est bien effectué.

- Idem en utilisant ProxyCommand au lieu de ProxyJump.

Pour cette nouvelle étape, nous allons utiliser ProxyCommand. Pour ce faire nous allons reprendre notre fichier config et le modifier comme ceci :

```
GNU nano 2.0.6      File: /Users/enzocollet/.ssh/config
Host srv
  Hostname 10.0.0.3
  User bob
  ProxyCommand ssh -W %h:%p cli
Host cli
  Hostname 10.0.0.2
  User bob
```

Maintenant que nous avons configuré le fichier config, nous pouvons passer à la suite.

- Vous pouvez utiliser la commande `who` pour vérifier de quelle adresse vous connectez.

Maintenant que nous avons configuré ProxyCommand, nous allons nous connecter au serveur srv. Une fois connecté, nous allons utiliser la commande `who` ci-dessous :

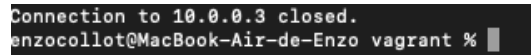
```
bob@srv:~$ who
bob      pts/0      2023-02-24 13:59 (10.0.0.2)
```

On peut constater que nous sommes connectés au serveur srv en passant par le serveur cli. Donc le rebond c'est bien effectué.

2.9 TD-9-SSH : Bonus SSH

— Vérifiez que vous pouvez couper une connexion SSH en utilisant une séquence d'échappement.

Pour cette nouvelle étape, nous allons vérifier que l'on peut couper une connexion SSH en utilisant une séquence d'échappement. Pour ce faire, nous allons dans une fenêtre SSH et nous allons utiliser ce caractère d'échappement :



On peut constater que nous avons réussi à mettre fin à la connexion SSH.

— Installez screen sur cli.

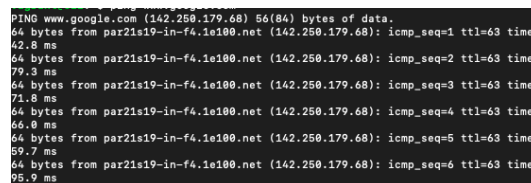
Connectez-vous-y

Lancez un screen , puis une commande dans un screen . - Déconnectez-vous du screen et de cli . Vérifiez que vous pouvez vous reconnecter à cli et au screen , et retrouver votre commande en cours d'exécution.

Pour cette nouvelle étape, nous allons installer screen sur cli. Pour ce faire, nous connecter à cli et taper la commande ci-dessous :

```
sudo apt-get install screen
```

Maintenant que nous avons installé screen, nous allons le lancer et taper une commande comme ci-dessous :



Maintenant que nous avons une commande en cours d'exécution, nous allons quitter la session avec la combinaison de touche ci-dessous :

Ctrl + ad

Une fois que nous avons quitter le terminal screen, nous allons y retourner avec la commande ci-dessous :

```
reconnected from 142.250.179.68:8081
vagrant@cli:~$ screen -r
```

```
59.8 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=47 ttl=63 time
53.0 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=48 ttl=63 time
37.8 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=49 ttl=63 time
73.9 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=50 ttl=63 time
70.3 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=51 ttl=63 time
34.4 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=52 ttl=63 time
39.9 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=53 ttl=63 time
54.4 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=54 ttl=63 time
49.4 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=55 ttl=63 time
45.5 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=56 ttl=63 time
58.1 ms
4 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=57 ttl=63 time
46.7 ms
```

On peut constater que nous avons retrouver notre terminal comme nous l'avons quitter.

— Idem avec tmux

Pour cette nouvelle étape, nous allons installer tmux sur cli. Pour ce faire, nous connecter à cli et taper la commande ci-dessous :

```
sudo apt-get install tmux
```

Maintenant que nous avons installer tmux, nous allons le lancer et taper une commande comme ci-dessous :

```
Processing triggers for
vagrant@cli:~$ tmux
```

```
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=1 ttl=63 tim
42.0 ms
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=2 ttl=63 tim
47.0 ms
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=3 ttl=63 tim
39.6 ms
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=4 ttl=63 tim
73.0 ms
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=5 ttl=63 tim
70.3 ms
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=6 ttl=63 tim
51.2 ms
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=7 ttl=63 tim
77.8 ms
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=8 ttl=63 tim
52.6 ms
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=9 ttl=63 tim
53.1 ms
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=10 ttl=63 ti
40.7 ms
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=11 ttl=63 ti
75.0 ms
```

Maintenant que nous avons une commande en cours d'exécution, nous allons quitter la session avec la combinaison de touche ci-dessous :

Ctrl + bd

Une fois que nous avons quitter le terminal screen, nous allons y retourner avec la commande ci-dessous :

```
vagrant@cli:~$ exit  
[detached (from session 0)]  
vagrant@cli:~$ tmux attach
```

```
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=109 ttl=63 tim  
e=57.6 ms  
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=110 ttl=63 tim  
e=41.7 ms  
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=111 ttl=63 tim  
e=87.9 ms  
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=112 ttl=63 tim  
e=39.7 ms  
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=113 ttl=63 tim  
e=70.4 ms  
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=114 ttl=63 tim  
e=70.2 ms  
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=115 ttl=63 tim  
e=65.7 ms  
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=116 ttl=63 tim  
e=57.5 ms  
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=117 ttl=63 tim  
e=53.8 ms  
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=118 ttl=63 tim  
e=47.6 ms  
64 bytes from par21s19-in-f4.1e100.net (142.250.179.68): icmp_seq=119 ttl=63 tim  
e=41.6 ms  
$ ping -c 1 -i 1 142.250.179.68  
"cli" 14:55 24-Feb-23
```

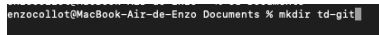
On peut constater que nous avons retrouver notre terminal comme nous l'avons quitter.

3 Git

3.1 TD-1-Git : Initialisation Git

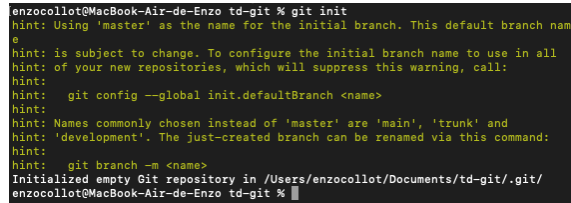
— Créer un nouveau répertoire et y initialiser un repository.

Pour cette nouvelle étape, nous allons créer un nouveau répertoire et initialiser un repository. Pour ce faire, voici les étapes ci-dessous :



```
enzocollot@MacBook-Air-de-Enzo Documents % mkdir td-git
```

(a) Création d'un répertoire pour git



```
enzocollot@MacBook-Air-de-Enzo td-git % git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/enzocollot/Documents/td-git/.git/
enzocollot@MacBook-Air-de-Enzo td-git %
```

(b) Initialisation d'un repository

FIGURE 7 – Deux captures d'écran montrant comment initialiser un repository

— Y copier la configuration Vagrant des TD ssh (Vagrantfile + dossier srv).

Maintenant que nous avons initialisé notre repository, nous allons copier les fichiers demandés dans la question. Voici la commande pour copier les fichiers :

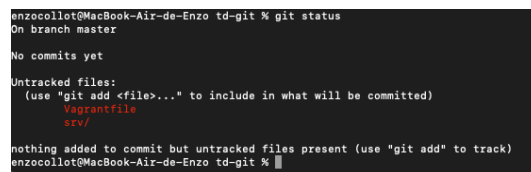
```
cp -r /chemin/Vagrantfile /chemin/td-git
cp -r /chemin/srv /chemin/td-git
```

Une fois que l'on a copié les bons documents aux bons endroits, nous pouvons passer à la question suivante.

— Regarder les modifications détectées par git.

Pour regarder les informations détectées par git, nous allons utiliser la commande suivante :

```
git status
```



```
enzocollot@MacBook-Air-de-Enzo td-git % git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Vagrantfile
    srv/

nothing added to commit but untracked files present (use "git add" to track)
enzocollot@MacBook-Air-de-Enzo td-git %
```

— Démarrer puis arrêter la configuration Vagrant.

Pour démarrer puis arrêter la configuration Vagrant, nous allons utiliser ses deux commandes ci-dessous :

Pour démarrer :

```
vagrant up
```

Pour arrêter :

```
vagrant halt
```

— Regarder à nouveau les modifications détectées par git.

Pour cette nouvelle étapes, nous allons regarder de nouveaux les modification détectées par git. Voici ci-dessous, les modification détectées par git :

```
enzocollot@MacBook-Air-de-Enzo td-git % git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .vagrant/
        Vagrantfile
        srv/

nothing added to commit but untracked files present (use "git add" to track)
enzocollot@MacBook-Air-de-Enzo td-git %
```

— Que remarque-t-on ? Comment peut-on régler le problème ?

On peut constater que nous avons actuellement sur la branche master, des fichiers non suivie et qu'il n'y a pas eu de commit fait.

Pour régler le problème, il suffit à d'ajouter les fichier et de faire un commit tout en ignorant le dossier .vagrant.

— Ajouter les fichiers Vagrant et commit les modifications.

Avant d'ajouter les fichiers Vagrant et commit les modifications, nous allons faire en sorte que git ignore le dossier .vagrant.

Pour ce faire, nous allons créé un fichier .gitignore et ajouter la ligne .vagrant comme ci-dessous :

```
GNU nano 2.0.6                                File: .gitignore
.vagrant/
```

Maintenant que nous avons ignorer le dossier .vagrant, nous allons ajouter les fichiers Vagrant et commit les modifications. Voici les étapes ci-dessous :

```
enzocollot@MacBook-Air-de-Enzo td-git % git add.
```

Une fois que nous avons ajouter les fichiers Vagrant, nous allons créé un commit comme ci-dessous :

```
enzocollot@MacBook-Air-de-Enzo td-git % git commit -m "Ajout des fichiers Vagrant"
```

Voilà nous avons commit et ajouter les fichiers Vagrant.

— Vérifier que le commit est bien présent et correct.

Pour verifir que le commit est bien présent et correct, nous allons utiliser la commande suivants :

git log

```
enzocollot@MacBook-Air-de-Enzo td-git % git log
commit e4e2d64fc33c8e95b4f07fa637f25dc190fad20a (HEAD -> master)
Author: EnzoCollot <collot.enzo@outlook.fr>
Date: Sat Feb 25 10:05:46 2023 +0100

    Ajout des fichiers Vagrant
enzocollot@MacBook-Air-de-Enzo td-git %
```

On peut constater que notre commit est bien présent et correct.

3.2 TD-2-Git : Les branches

- Créer une nouvelle branche.

Dans cette nouvelle partie, nous allons créer une nouvelle branche. Pour ce faire, nous allons utiliser la commande suivante :

```
git checkout -b "nom-de-la-branche"
```

```
lenzocollot@MacBook-Air-de-Enzo td-git % git checkout -b ma-nouvelle-branche
```

- Dans le Vagrantfile, ajouter l'utilisateur patrick.

Pour cette nouvelle étapes, nous allons ajouter l'utilisateur patrick comme ci-dessous :

```
useradd --shell /bin/bash --create-home patrick || true
echo alice:1234 | chpasswd
echo bob:azerty | chpasswd
echo carol:secret | chpasswd
echo patrick:test | chpasswd
```

- Dans le Vagrantfile, installer php et l'activer sur apache.

Pour cette nouvelle étapes, nous allons installer php et l'activer sur apache comme ci-dessous :

```
b.vm.provision "shell", inline: <<-SHELL
# Installation de PHP
apt-get update
apt-get install -y php libapache2-mod-php

# Activation de PHP dans Apache
a2enmod php
systemctl restart apache2
```

- Effectuer 2 commits distincts à l'aide de la commande git add -p.

Pour cette nouvelle étapes, nous allons effectuer 2 commits distincts à l'aide de la commande git add -p. Voici les étapes ci-dessous :

Pour commencer, nous allons taper la commande suivante :

```
git add -p Vagrantfile
```

Ensuite nous allons cliquer sur Y.

```
lenzocollot@MacBook-Air-de-Enzo td-git % git add -p Vagrantfile
diff --git a/Vagrantfile b/Vagrantfile
index ea56ce..c3b5d1 100644
--- a/Vagrantfile
+++ b/Vagrantfile
@@ -12,9 +12,11 @@ Vagrant.configure("2") do |config|
  useradd --shell /bin/bash --create-home alice || true
  useradd --shell /bin/bash --create-home bob || true
  useradd --shell /bin/bash --create-home carol || true
+ useradd --shell /bin/bash --create-home patrick || true
  echo alice:1234 | chpasswd
  echo bob:azerty | chpasswd
  echo carol:secret | chpasswd
+ echo patrick:test | chpasswd
  echo Enabling password auth
  sed -i 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
  systemctl restart ssh.service
1/1 Stage this hunk [y,n,q,a,d,s,e,?]? y
```

Ensuite, nous allons faire un commits comme ci-dessous :

```
enzocollot@MacBook-Air-de-Enzo td-git % git commit -m "Ajout utilisateur patrick Vagrantfile"
```

Maintenant que nous avons fait le premier commit, nous allons effectuer le deuxième. Voici les étapes ci-dessous :

Nous allons retaper la commande suivante :

```
git add -p Vagrantfile
```

Ensuite nous allons cliquer sur Y.

```
enzocollot@MacBook-Air-de-Enzo td-git % git add -p Vagrantfile
diff --git a/Vagrantfile b/Vagrantfile
index c3b55d1..a0f4015 100644
--- a/Vagrantfile
+++ b/Vagrantfile
@@ -34,6 +34,14 @@ Vagrant.configure("2") do |config|
   b.vm.hostname = "srv"

   b.vm.provision "shell", inline: <<-SHELL
+  # Installation de PHP
+  apt-get update
+  apt-get install -y php libapache2-mod-php
+
+  # Activation de PHP dans Apache
+  a2enmod php
+  systemctl restart apache2
+
   apt-get -y install apache2 ruby
   a2enmod cgi
   systemctl restart apache2.service
(1/1) Stage this hunk [y,n,q,a,d,e,f]? y
```

Ensuite, nous allons faire un commits comme ci-dessous :

```
enzocollot@MacBook-Air-de-Enzo td-git % git commit -m "Installation de PHP et Activation dans Apache2"
```

Voilà, nous avons effectuer 2 commits distincts.

— Revenir à la branche main , quel est l'état de votre working directory ?

Pour revenir à la branche main ou master, il suffit de taper la commande suivante :

```
git checkout main ou master
```

```
enzocollot@MacBook-Air-de-Enzo td-git % git checkout master
Switched to branch 'master'
enzocollot@MacBook-Air-de-Enzo td-git % git status
On branch master
nothing to commit, working tree clean
enzocollot@MacBook-Air-de-Enzo td-git %
```

On peut constater que dans notre working directory, nous avons plus les fichier Vagrants. Pour les revoir, il suffit juste de retoruner sur notre autre branche et de push vers master.

3.3 TD-3-Git : Réintégrer les changements

— Intégrer les modification de code de l'exercice précédent dans main.

Pour réintégrer les modification de code de l'exercice précédent dans main, nous allons utiliser la commande suivante :

```
git merge nom-de-la-branche
```

```
enzocollot@MacBook-Air-de-Enzo td-git % git merge ma-nouvelle-branche
```

— Inspecter le commit de merge, y'a t'il des spécificités ?

Pour intégrer le commit de merge, nous allons utiliser la commande suivant :

```
git log --online
```

```
enzocollot@MacBook-Air-de-Enzo td-git % git log --online
c89d3d8 (HEAD -> master, ma-nouvelle-branche) ajout php
dff25cc Ajout utilisateur patrick
d9b8188 Modification Vagrant
```

On peut constater que nous n'avons pas de spécificités, juste que maintenant, la branche ma-nouvelle-branche est attaché à la branche master.

— Est-ce que la branche que vous avez créé existe toujours ? Qu'est-ce qu'on en fait ?

Oui la branche que nous avons créé existe toujours. On peut la garder pour des modification future dessus et ensuite, on pourra passer les modification sur la branche principale/