

## TP n° 1 : *shell* et connexion à distance avec *ssh*

Avant toute chose, assurez-vous que vous connaissez votre nom d'utilisateur et votre mot de passe pour les machines de l'UFR. Il s'agit du même nom d'utilisateur et du même mot de passe que vous utilisez pour vous connecter à l'instance GitLab de l'UFR (<https://gaufre.informatique.univ-paris-diderot.fr>), ainsi qu'aux ordinateurs des salles de TP du bâtiment Sophie Germain.

Connectez-vous maintenant *sur une des machines de la salle de TP* (et non sur votre machine personnelle), et ouvrez une fenêtre de terminal.

### 1 Travailler à distance avec *ssh*

Au cours des séances suivantes et pendant la préparation du projet, vous aurez parfois besoin de travailler en ligne de commande sur *lulu*, qui est un des serveurs de l'UFR d'informatique. Pour se connecter à distance à cette machine, on utilise le programme *ssh* (« *secure shell* »). Cette section explique comment configurer ce dernier.

Commencez par configurer la connexion depuis une machine de l'UFR. Ensuite, vous pourrez éventuellement configurer la connexion depuis votre machine personnelle.

#### 1.1 Pour vous connecter depuis une machine de l'UFR

**Étape 1 :** Sur la machine locale, créez une paire clé publique / clé privée en entrant la commande suivante :

```
$ ssh-keygen -t ed25519
```

On vous proposera de sauvegarder la clé dans le fichier `vous-repertoire-personnel/.ssh/id_ed25519` : acceptez en appuyant simplement sur entrée. Choisissez ensuite une « *passphrase* » : un mot de passe que vous devrez taper à chaque fois que vous utiliserez la clé. **Assurez-vous de la retenir** car vous en aurez besoin pour les séances suivantes et pour le projet.

**Étape 2 :** Ajoutez votre clé publique à la liste des clés autorisées à se connecter à votre compte :

```
$ cat ~/.ssh/id_ed25519.pub >> ~/.ssh/authorized_keys
$ chmod 600 ~/.ssh/authorized_keys
```

**Étape 3 :** Connectez-vous à la machine *lulu* en entrant la commande suivante sur la machine locale :

```
$ ssh lulu
```

On vous demandera peut-être de vérifier l'identité du serveur *lulu* : assurez-vous que l'empreinte (« *fingerprint* ») est bien `SHA256:kMo3xkwhzbL77aFHC1gpicEWvQDH403eC4kEUZP1DA`, puis validez en tapant `yes`. Enfin, entrez la *passphrase* que vous avez choisie pour votre clé. Vous devriez voir apparaître l'invite suivante :

```
xxxxxxxx@lulu:~$
```

Voilà, vous êtes connecté(e) ! Vous pouvez vous déconnecter en tapant la commande `exit`.

**Étape 4 :** Vous pouvez maintenant vous connecter depuis n'importe quelle machine de l'UFR en tapant simplement `ssh lulu` et en entrant la passphrase de votre clé.

## 1.2 Pour vous connecter depuis votre machine personnelle

**Étape 1 :** Sur votre machine, créez une paire clé publique / clé privée en entrant la commande suivante :

```
$ ssh-keygen -t ed25519
```

**Étape 2 :** Copiez votre clé publique vers les machine de l'UFR et connectez-vous à la machine `nivose` en entrant les commandes suivantes sur votre machine (remplacez `xxxxxxxx` par votre nom d'utilisateur à l'UFR) :

```
$ scp ~/.ssh/id_ed25519.pub xxxxxxxx@nivose.informatique.univ-paris-diderot.fr:tmp.pub
```

```
# On vous demandera de vérifier l'identité de nivose :  
#   SHA256:Tp/LWjUf9EBKXfi8JKjJviglrkErE9UguELfnxosaYM  
# Puis on vous demandera votre mot de passe de l'UFR
```

```
$ ssh xxxxxxxx@nivose.informatique.univ-paris-diderot.fr
```

```
# On vous demandera à nouveau votre mot de passe de l'UFR
```

Maintenant que vous êtes connecté(e) à `nivose`, entrez les commandes suivantes pour autoriser la connexion avec la clé de votre machine, puis vous déconnecter :

```
xxxxxxxx@nivose:~$ cat ~/.ssh/tmp.pub >> ~/.ssh/authorized_keys  
xxxxxxxx@nivose:~$ exit
```

**Étape 3 :** Créez sur votre machine un fichier `/.ssh/config` et écrivez-y les lignes suivantes (remplacez `xxxxxxxx` par votre nom d'utilisateur à l'UFR). Si le fichier existe déjà, ajoutez simplement ces lignes à la fin.

```
Host lulu
  Hostname lulu.informatique.univ-paris-diderot.fr
  User xxxxxxxx
  ProxyJump lucy

Host lucy
  Hostname lucy.informatique.univ-paris-diderot.fr
  User xxxxxxxx
```

**Étape 4 :** Connectez-vous à la machine lulu en entrant la commande suivante sur votre machine :

```
$ ssh lulu
```

On vous demandera de vérifier l'identité du serveur `lucy.informatique.univ-paris-diderot.fr` (l'empreinte doit être `SHA256:LGPCbRoZbt0lG2LPm3xJoDvkWYScnYyRkv1GUHqPaDc`), puis celle de `lulu.informatique.univ-paris-diderot.fr` (`SHA256:kMo3xkwhzbL77aFHC1gp1cEWvQDH403eC4kEUZP1DA`). Enfin, entrez l'éventuelle passphrase que vous avez choisie pour votre clé. Vous devriez voir apparaître l'invite suivante :

```
xxxxxxx@lulu:~$
```

Voilà, vous êtes connecté(e) ! Vous pouvez vous déconnecter en tapant la commande `exit`.

**Étape 5 :** Vous pouvez maintenant vous connecter depuis votre machine en tapant simplement `ssh lulu` et en entrant la passphrase de votre clé.

*(Le reste du TP peut se faire sur votre machine personnelle, sur une machine de la salle de TP, ou bien sur lulu)*

## 2 Clonage du dépôt git du cours

Toutes les feuilles de TP, les transparents de cours et les informations sur le projet seront diffusées à travers un dépôt *git* situé à l'adresse suivante :

`git@gaufre.informatique.univ-paris-diderot.fr:poulalho/sy5-2021-2022.git`.

Pour pouvoir cloner ce dernier, vous devez indiquer votre clé publique à *gaufre*, le serveur *gitlab* de l'UFR.

1. Utilisez la commande `cat` pour afficher votre clé **publique**, qui est située dans le fichier `/.ssh/id_ed25519.pub`. Le résultat doit ressembler à cela :

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXX username@machine
```

2. Connectez vous à *gaufre* (`https://gaufre.informatique.univ-paris-diderot.fr`) à l'aide de votre nom d'utilisateur et mot de passe de l'UFR.

3. Depuis l'adresse `https://gaufre.informatique.univ-paris-diderot.fr/profile/keys`, ajoutez votre clef en copiant le contenu de votre clef **publique** dans le champ Key.
4. Vérifiez que l'authentification marche avec la commande :

```
$ ssh -T git@gaufre.informatique.univ-paris-diderot.fr
```

5. Vous pouvez maintenant cloner le dépôt git du cours avec la commande :

```
$ git clone git@gaufre.informatique.univ-paris-diderot.fr:poulalho/sy5-2022-2023.git
```

### 3 Rappels de shell

Pour commencer, déplacez-vous dans le dossier TP/TP1/ du dépôt git du cours, puis lancez la commande `make init`. Ceci créera un dossier TP/TP1/work/ : c'est dans ce dossier que vous devrez travailler. Ne touchez pas au dossier `template/` : vous risqueriez ensuite d'avoir des problèmes pour mettre à jour le dépôt.

#### Exercice 1 – Connaître ses droits

Déplacez-vous dans le dossier TP/TP1/work/ex-1/.

1. Afficher les permissions du répertoire `Arborescence/Protegee/A`.
2. Lister ce que contient le répertoire `Arborescence/Protegee/A`.
3. Afficher dans le terminal le contenu du fichier `Arborescence/Protegee/A/toutou`.  
Pourquoi n'est-ce pas possible ? Faites le nécessaire. Pourquoi pouviez-vous lister ce qu'il y a dans `Arborescence/Protegee/A` ?  
Afficher ensuite le contenu du fichier avec en plus les numéros de ligne.
4. Pouvez-vous créer un fichier `Arborescence/Protegee/B/Test` ? Pouvez-vous changer le nom de `Arborescence/Protegee/B/A` ? Pourquoi ? Quelles sont les autres opérations que vous ne pouvez pas faire dans ce répertoire ?  
Faites le nécessaire pour pouvoir créer `Arborescence/Protegee/B/Test`.
5. Faire en sorte que les utilisateurs de votre groupe puissent lister ce qu'il y a dans votre répertoire de nom `Arborescence/Protegee/A/A`, mais ne puissent ni le modifier, ni accéder à ses fichiers.
6. Faire en sorte que les fichiers de `Arborescence/Protegee/A/B` soient lisibles uniquement par utilisateurs connaissant leur nom.
7. Supprimer l'arborescence de racine `Arborescence/Vide`.
8. Supprimer l'arborescence de racine `Arborescence/Protegee/C`.

#### Exercice 2 – Comptage, tri, chaînage

Déplacez-vous dans le dossier TP/TP1/work/ex-2/.

1. Combien y a-t-il de mots dans le fichier `texte.txt` ? Combien de lignes ? (Voir `man wc`).
2. Que fait la commande `cat mots.txt | sort` ?
3. Combien y a-t-il de mots *distincts* dans le fichier `mots.txt` ? (Voir `man uniq`).

4. Quel est le plus grand nombre écrit dans le fichier `nombres.txt` ? (Voir `man sort` ; chercher l'option qui permet de trier dans l'ordre numérique et non alphabétique).
5. Que fait la commande `find dossier | wc -l` ?
6. En vous inspirant de la question précédente : combien y a-t-il de fichiers situés dans le répertoire `dossier` (y compris dans tous ses sous-dossiers) dont le nom est de la forme `*.txt` ? (Voir `man find`, option `-iname`).

### Exercice 3 – Concaténation, arguments, globbing

Déplacez-vous dans le dossier `TP/TP1/work/ex-3/`.

1. Que fait la commande `cat ligne-01.txt ligne-02.txt > resultat.txt` ?
2. Que fait la commande `cat ligne-01.txt - ligne-02.txt > resultat.txt` ? Que représente l'argument « - » pour la commande `cat` ?
3. Que fait la commande suivante ?

```
$ printf '%s\n' un-argument un-autre-argument un-troisieme-argument
```

4. Comparez ce qu'affichent les deux commandes suivantes :

```
$ printf '%s\n' 'un argument' 'un autre argument' 'un troisieme argument'
$ printf '%s\n' un argument un autre argument un troisieme argument
```

Expliquez la différence.

5. Que fait la commande suivante ?

```
$ printf '%s\n' *.txt
```

6. Utilisez une commande pour concaténer tous les fichiers `ligne-*.txt`, et afficher le résultat.
7. Modifiez la commande précédente pour concaténer les mêmes fichiers, mais stocker le résultat dans `resultat.txt`. Vérifiez en affichant le contenu de ce fichier.

### Exercice 4 – Boucles, variables, conditions

Déplacez-vous dans le dossier `TP/TP1/work/ex-4/`.

1. Que fait la commande suivante ? (Voir `man [` ou `man test`)

```
$ for FILE in fichiers/*
do
    if [ -f "$FILE" ]
    then
        printf '%s est un fichier ordinaire\n' "$FILE"
    elif [ -d "$FILE" ]
    then
        printf '%s est un dossier\n' "$FILE"
    fi
done
```

2. Le dossier `depots` contient à la fois des fichiers et des sous-dossiers. Certains de ces sous-dossiers sont des *dépôts git* : on les reconnaît au fait qu'ils contiennent un sous-sous-dossier appelé `.git`. Écrire une commande qui parcourt le contenu du dossier `depots` et affiche uniquement les dossiers qui sont des dépôts *git*.
3. Que fait la commande `convert champignon.png champignon.jpg` ?
4. Écrire une commande qui parcourt tous les fichiers du dossier `images` dont le nom est de la forme `*.png`, et les convertit en images JPEG en utilisant la commande `convert`. Par exemple, le fichier `cerise.png` sera converti en un fichier appelé `cerise.png.jpg`.

### Exercice 5 – Substitutions de commandes

1. Que fait la commande `date` ?
2. Que fait la commande suivante ?

```
$ printf "Voici la date et l'heure : %s\n" "$(date)"
```

3. (*Plus complexe*) Reprenez la dernière question de l'exercice 3, et modifiez la commande pour qu'un fichier de la forme `abcd.jpg` soit converti en un fichier appelé `abcd.png` (plutôt que `abcd.jpg.png`). Indication : qu'affiche la commande `basename images/abcd.png .png` ?