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 normalement 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`). Si vous ne vous en rappelez pas, cherchez dans votre boite mail de l'université un e-mail émanant de Laurent Pietroni : vous devriez y trouver vos identifiants.

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, d'une part pour une connection entre machines du réseau de l'UFR (section 1.1), et d'autre part depuis une machine personnelle (section 1.2). Les deux seront nécessaires, mais commencez de préférence par configurer la connexion depuis une machine de l'UFR, c'est un peu plus simple.

1.1 Pour vous connecter depuis une machine de l'UFR

Connectez-vous sur une des machines de la salle de TP, et ouvrez un terminal.

Étape 1 : 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 `votre-rep-perso/.ssh/id_ed25519` : acceptez en appuyant simplement sur entrée. Choisissez ensuite une « passphrase » : un (long) 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`. Si c'est le cas, assurez-vous que l'empreinte (« fingerprint ») est bien l'une des trois suivantes :

```
(ECDSA)   SHA256:F/SDQYSQNwHLPRlBJUN7PsLJToi8NyPWOCp3/oqlOLO
(ED25519) SHA256:kMo3xkwhzbL77aFHWC1gpicEWvQDH4O3eC4kEUZPlDA
(RSA)     SHA256:nxdxoYSKfpUd6kSWUEJfWhJvFBDaMCLSnwGtEbAMAXo
```

puis validez en tapant `yes`. Enfin, entrez la passphrase que vous avez choisie pour votre clé. Vous devriez voir apparaître l'invite suivante (où `xxxxxxxx` est votre nom d'utilisateur à l'UFR) :

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

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

Vous pouvez maintenant vous connecter à `lulu` 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 : créez sur votre machine un fichier `~/.ssh/config` et écrivez-y les lignes suivantes (remplacez `xxxxxxx` 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

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

Étape 3 : entrez la commande suivante (sur votre machine) afin de copier votre clé publique vers les machines de l'UFR :

```
$ scp ~/.ssh/id_ed25519.pub nivose:tmp.pub
```

On vous demandera de vérifier l'identité de la machine `nivose`, assurez-vous qu'il s'agit d'une des trois suivantes :

```
(ECDSA)   SHA256:8TdUwAXmNj+Q5dCHo+WdJpnqIB6yyxiM+HnIBjbDBzI
(ED25519) SHA256:Tp/LWjUf9EBKXfi8JKjJviglrkErE9UguELfnxosaYM
(RSA)     SHA256:qGEBkOMPsEgYkZk/rx4kD93Z9Ze60lzcTiN1RZQx6Vo
```

puis validez en tapant `yes`. Ensuite, on vous demandera votre mot de passe de l'UFR. Une fois la copie effectuée, entrez la commande suivante pour vous connecter à la machine `nivose` :

```
$ ssh nivose
```

On vous demandera à nouveau votre mot de passe de l'UFR. Une fois connecté(e) à `nivose`, entrez les commandes suivantes pour autoriser la connexion avec la clé de votre machine, puis vous déconnecter :

```
$ mkdir -p .ssh       # inutile si le répertoire existe déjà
$ cat ~/tmp.pub >> ~/.ssh/authorized_keys
$ exit
```

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 de `lucy.informatique.univ-paris-diderot.fr`, l'empreinte doit être l'une des trois suivantes :

```
(ECDSA)   SHA256:PU/C7DzGz49Xt1eVQg20t5A+1p3mJgoVDAJYG8Wykzw
(ED25519) SHA256:LGPCbRoZbtOlG2LPm3xJoDvkWYScnYyRkvlGUHqPaDc
(RSA)     SHA256:SB999C4wpDDVZZaXnpw+0CFgkMfcvxa03XARM3Op1Xs
```

Ensuite, on vous demandera de vérifier celle `lulu.informatique.univ-paris-diderot.fr`, l'empreinte doit être l'une des trois suivantes :

```
(ECDSA)   SHA256:F/SDQYSQNwHLPRlBJUN7PsLJToi8NyPW0Cp3/oqlOL0
(ED25519) SHA256:kMo3xkwhzbL77aFHWC1gpicEWvQDH4O3eC4kEUZPlDA
(RSA)     SHA256:nxdxoYSKfpUd6kSWUEJfWhJvFBDaMCLSnwGtEbAMAXo
```

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`.

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 :
`https://gaufre.informatique.univ-paris-diderot.fr/poulalho/sy5-2023-2024`.

Pour pouvoir le cloner, vous devez indiquer votre clé publique (sur la machine concernée) 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 AAAAC3NzaC1lZDI1NTE5AAAAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   XXXXXXX 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
   ```

   Le cas échéant, vérifiez l'empreinte du serveur, qui doit être l'une des trois suivantes :

   ```
   (ECDSA)   SHA256:cBL3zv2yWz7bxCYTsTmJFLCySuswa5Yebr5HnU1MSBk
   (ED25519) SHA256:nL5LNNjayAr3iK6ZgGdpF47ips2XubTv55SVfrs0//g
   (RSA)     SHA256:vh6HANdYhSbIn9XRJPOB2RsvIuaR6YovKhkyUzmyEsQ
   ```

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

   ```
   $ git clone git@gaufre.informatique.univ-paris-diderot.fr:poulalho/sy5-2023-2024.git
   ```

3 Rappels

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

Exercice 1 – *effets des droits d'accès*

Dans cet exercice et les deux suivants, vous devriez utiliser (au moins) les commandes `ls`, `cd`, `cat`, `chmod`, `mv`, `rm` (liste non exhaustive). Et probablement `man`, bien entendu.

Déplacez-vous dans le répertoire `TP/TP1/work/ex-1-2-3/`.

1. Afficher les permissions du répertoire `Arborescence/Protegee/A`. Pouvez-vous lister son contenu ? Vous y déplacer ?
2. Essayez d'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 le contenu de `Arborescence/Protegee/A` ?
   Afficher ensuite le contenu du fichier avec en plus les numéros de ligne.

3. 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`.
4. 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.
5. Faire en sorte que les fichiers de `Arborescence/Protegee/A/B` soient lisibles uniquement par les utilisateurs connaissant leur nom. Autrement dit, vous devez faire en sorte que la commande

   ```
   cat Arborescence/Protegee/A/B/titi
   ```

   fonctionne *mais* que la commande

   ```
   ls Arborescence/Protegee/A/B/
   ```

   provoque une erreur de permissions.
6. Pouvez-vous supprimer l'arborescence de racine `Arborescence/Vide` avec la commande `rmdir` ? Pourquoi ? Supprimez-la tout de même.
7. Supprimer l'arborescence de racine `Arborescence/Protegee/C`.

Exercice 2 – *inœuds et liens multiples*

1. Afficher toute l'arborescence de racine `Arborescence/Profonde/C` et la dessiner en précisant le numéro d'inœud de chaque fichier.
2. Combien y a-t-il de fichiers ordinaires *différents* dans cette arborescence ?
3. Combien de liens y a-t-il sur le répertoire de nom `Arborescence/Profonde/C` ? Compléter le dessin en ajoutant les liens qui manquent.
4. Ajouter dans le répertoire `Arborescence` un lien sur le fichier de nom `hareng`, nommé `monlien`. Faire également une copie de `hareng` nommée `macopie`.
5. Modifier le contenu du fichier `hareng` *avec une seule ligne de commande* en lui ajoutant la ligne « Ta maison ronde où il nage un hareng saur ». Comparez le contenu des fichiers de noms `hareng`, `monlien` et `macopie`.

Exercice 3 – *un peu de recherche*

Dans cet exercice, vous devrez faire des parcours récursifs, soit à l'aide de `ls`, soit à l'aide de `find`.

L'arborescence de racine `Arborescence/Profonde` est relativement grosse... mais recèle quelques pépites.

1. Combien de fichiers de nom `titi`, `toto` ou `tutu` cette arborescence contient-elle ?
2. Le fichier de référence `Arborescence/Profonde/vigile` doit absolument être supprimé... Il est accessible par de nombreux liens, faites en sorte qu'il n'en reste aucune trace.
3. Vérifier que l'arborescence contient 4 fichiers ordinaires de plus que les `titi`, `toto` et `tutu` précédemment comptés. Où sont-ils ?

Exercice 4 – *comptage, tri, chaînage*

Dans cet exercice, vous devriez utiliser (au moins) les commandes `wc`, `cat`, `sort`, `uniq`, `find`, `head` ou `tail`.
Et `man`, bien entendu.

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

1. Combien y a-t-il de mots dans le fichier `texte.txt` ? Combien de lignes ?
2. Que fait la commande `cat mots.txt | sort` ?
3. Combien y a-t-il de mots *distincts* dans le fichier `mots.txt` ?
4. Quel est est le plus grand nombre écrit dans le fichier `nombres.txt` ?
5. Combien y a-t-il de fichiers ordinaires situés dans *l'arborescence* de racine `dossier` (donc y compris dans tous ses sous-répertoires) ? Combien d'entre eux ont un nom terminant par l'extension `.txt` ?

Exercice 5 – *concaténation, arguments, globbing, redirections*

Déplacez-vous dans le répertoire `TP/TP1/work/ex-5/`.

1. Que fait la commande `cat ligne-01.txt ligne-02.txt > resultat.txt` ? Et la commande `cat ligne-03.txt ligne-04.txt >> resultat.txt` ?
2. Que fait la commande `cat ligne-01.txt - ligne-02.txt > resultat.txt` ? Si elle échoue, remplacer > par >|. 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 6 – *redirection vers un autre terminal*

Sous Unix, les périphériques sont considérés comme des fichiers spéciaux, accessibles par des liens situés dans la sous-arborescence `/dev` du système de fichiers.

1. La commande `tty` retourne la référence absolue du fichier spécial correspondant au terminal dans lequel elle est exécutée. Affichez les caractéristiques (droits, etc.) de ce fichier.

2. Dans un autre terminal, utilisez la commande `echo coucou` en redirigeant sa sortie standard vers le fichier spécial correspondant au premier terminal. Que se passe-t-il ?
3. Connectez-vous à `lulu`, puis faites ce qu'il faut pour que les autres étudiants puissent écrire sur votre terminal. Cherchez également qui est connecté sur `lulu` pour voir avec quel étudiant vous pouvez échanger des messages via vos terminaux.

Exercice 7 – *boucles, variables, conditions*

Déplacez-vous dans le répertoire `TP/TP1/work/ex-7/`.

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 "$FILE" ]
    then
      printf '%s est un répertoire\n' "$FILE"
    fi
  done
```

2. Le répertoire `depots` contient à la fois des fichiers et des sous-répertoires. Certains de ces sous-répertoires sont des *dépôts git* : on les reconnaît au fait qu'ils contiennent un sous-répertoire appelé `.git`. Écrire une commande qui parcourt le contenu du répertoire `depots` et affiche uniquement les répertoires 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 répertoire `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 8 – *substitutions de commandes*

1. Que fait la commande suivante ?

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

2. *(Plus complexe)* Reprenez la dernière question de l'exercice précédent, 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 : la commande `basename` peut être utile)*