TP n° 1 : *shell* et révisions C

1 Travailler à distance avec « `ssh` »

On utilise le programme « `ssh` » (*"secure shell"*) pour se connecter à une machine distante, par exemple pour travailler sur les machines de l'UFR depuis son ordinateur.

Exercice 1 – *Première connexion avec « `ssh` »*

Depuis un terminal, connectez-vous à la machine « `nivose` » à l'aide de la commande :

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

où `$login` désigne votre identifiant au sein de l'UFR d'informatique. Vous serez amené à rentrer votre mot de passe.

Vous êtes désormais connecté à la machine « `nivose` », qui n'est pas suffisamment puissante pour faire tourner des calculs importants. Pour des raisons de sécurité, les autres machines de l'UFR ne sont pas accessibles de cette façon. On ne peut s'y connecter que par *authentification à clef publique*, une méthode bien plus sûre, qui nécessite la création d'une paire de clefs publique et privée. Déconnectez-vous (ctrl+D).

Exercice 2 – *Création des clefs*

1. Consultez le manuel de la commande « `ssh-keygen` ». À quoi sert cette commande ? Quel est le rôle des options « `-t` », « `-f` », « `-C` » ? Quelles sont les différentes façons de renseigner la phrase de passe d'une clef `ssh` ?
2. Depuis le répertoire « `~/.ssh` », générez une clef de type « `ed25519` », dans un fichier nommé « `id_ed25519` », avec pour commentaire votre adresse e-mail. Utilisez une phrase de passe complexe (par exemple comme indiqué à `https://www.eff.org/dice`), à garder précieusement en lieu sûr.

Le fichier « `id_ed25519` » qui vient d'être créé est une **clef privée**. Par sécurité, cette clef privée ne devrait jamais être copiée sur une autre machine ; pour se connecter par « `ssh` » depuis un autre ordinateur, il convient de créer une nouvelle paire de clefs depuis cet ordinateur. Le fichier « `id_ed25519.pub` » est la **clef publique** correspondante. Elle est faite pour être partagée librement, et nous allons la copier sur « `nivose` ».

Exercice 3 – *Enregistrer sa clef sur une machine distante*

1. Consultez le manuel de la commande « `scp` », qui sert à copier des fichiers entre différentes machines. Quelle technologie utilise-t-elle à cet effet ?
2. Copiez votre clef *publique* sur la machine « `nivose` » :

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

(n'oubliez pas le `:` final).

Finalement, il faut dire aux machines de l'UFR d'accepter notre clef publique pour se connecter avec « `ssh` ». Comme elles partagent toutes la même arborescence de fichiers, il suffit de le faire depuis « `nivose` ».

3. Connectez-vous à nouveau sur « `nivose` ». Que contient votre répertoire « `$HOME` » ?
4. Ajoutez votre clef publique au fichier `~/.ssh/authorized_keys` :

```
$ mkdir -p ~/.ssh
$ cat ~/id_ed25519.pub >> ~/.ssh/authorized_keys
$ rm -f ~/id_ed25519.pub
$ chmod 600 ~/.ssh/authorized_keys
```

5. À quoi sert la commande « `chmod` » ? Pourquoi exécute-t-on cette dernière commande ?

Remarque. Pour copier nos clefs, nous utilisons une vieille machine, « `nivose` ». En général, on peut utiliser l'utilitaire « `ssh-copy-id` » pour automatiser la copie des clefs publiques.

Déconnectez-vous. Vous devriez maintenant être en mesure de vous connecter à une machine de l'UFR. Plutôt que « `nivose` », il est préférable de se connecter à « `lucy` » (ou « `lucette` », qui est le nom de la même machine vue depuis le réseau de l'UFR).

Exercice 4 – *Connexion par clef publique*

1. Connectez-vous à « `lucy` » (ou « `lucette` »). Que vous demande-t-on ?

Pour éviter de rentrer le mot de passe de sa clef `ssh` à chaque connexion, on peut utiliser un **agent ssh**.

2. En consultant le manuel, trouvez à quoi servent les commandes « `ssh-agent` » et « `ssh-add` ».
3. Depuis votre machine, exécutez la commande « `eval $(ssh-agent -t 1h)` ». Que vient-on de faire ?
4. Ajoutez votre clef publique à l'agent ssh nouvellement créé et reconnectez-vous à « `lucy` ».

La machine « `lucy` », tout comme « `nivose` », n'est pas faite pour exécuter des programmes conséquents. On ne l'utilise que pour se connecter au réseau, avant de rejoindre une autre machine plus capable.

5. Consultez le manuel de la commande « `ssh` ». À quoi sert l'option « `-J` » ?
6. Connectez-vous à la machine « `lulu.informatique.univ-paris-diderot.fr` » en faisant un « rebond » *via* « `lucy` ».

(facultatif) 7. Consultez le manuel de « `ssh_config` ». Rédigez un fichier de configuration `ssh` qui vous permette de vous connecter à « `lulu` » en tapant simplement « `ssh lulu` ».

Exercice 5 – *Utilisation des clefs avec « `git` »*

Pour pouvoir nous connecter par `ssh` à « `gaufre` », le serveur `git` de l'UFR, profitons-en pour l'informer de notre clef publique.

1. 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` ».
2. Vérifiez que l'authentification marche avec la commande :
   « `ssh -T git@gaufre.informatique.univ-paris-diderot.fr` ».
3. Vous pouvez maintenant cloner le dépôt git du cours avec la commande :
   « `git clone git@gaufre.informatique.univ-paris-diderot.fr:poulalho/sy5_2021-2022.git` ».

2 Interagir avec le *shell*

Exercice 6 – *Variables*

Le *shell* que vous utilisez (probablement « `bash` » ou « `zsh` ») est capable de maintenir des variables, dont un certain nombre sont définies à l'avance. Dans une commande, la chaîne « `$var` » est remplacée par le contenu de la variable « `var` ».

1. Quel est le résultat de la commande « `echo $HOME` » ?
2. Que contiennent les variables « `USER` », « `SHELL` », « `TERM` », « `PATH` », « `PWD` », « `OLDPWD` » ? Déplacez-vous dans l'arborescence des répertoires. Comment change la valeur de « `PWD` » et « `OLDPWD` » ?
3. Que fait la commande « `export` » ? Référez-vous au manuel de « `bash` » (section « `SHELL BUILTIN COMMANDS` »).
4. Affichez la liste des variables présentes dans l'environnement de votre *shell*, puis déclarez une nouvelle variable d'environnement « `FOO` » ayant comme valeur « `foo` ». Affichez à nouveau la liste des variables pour vérifier que « `FOO` » est bien déclarée.
5. Le *shell* offre de nombreuses options d'expansion conditionnelle des variables, prenant la forme « `${VAR:···}` », décrites dans la section « `Parameter Expansion` » du manuel de « `bash` ». Quelle commande permet d'afficher le contenu d'une variable si celui-ci existe et est non-vide, et la chaîne « `bar` » sinon ? Testez votre commande avec la variable « `FOO` » (définie plus tôt) et la variable « `BAR` » (non définie).
6. On peut donner une valeur à une variable exportée « `VAR` » en entrant « `VAR=valeur` » dans le shell. Donnez la valeur nulle à la variable « `FOO` » (« `FOO=` »), et retestez la commande de la question précédente.
7. Fermez votre terminal et ouvrez-en un nouveau. Qu'est-il arrivé à la variable « `FOO` » ?

Exercice 7 – *Chaînes de caractères et redirections*

1. Quelle différence observez-vous dans les effets des commandes « `touch "foo bar"` », « `touch 'bar baz'` », « `touch baz quux` » ? Effacez les fichiers créés.
2. Entrez manuellement les commandes :

   ```
   $ echo Plus ne suis ce que i\'ay esté > marot1.txt
   $ echo 'Et ne le sçaurois iamais estre' >> marot1.txt
   ```

   (si la première commande échoue, utilisez « `>|` » plutôt que « `>` »). Affichez le contenu du fichier « `marot1.txt` ».

3. Entrez manuellement les commandes :

```
$ cat >> marot1.txt << EOF
Mon beau printemps & mon esté
Ont faict le sault par la fenestre.
EOF
```

   Que contient maintenant le fichier « `marot1.txt` » ?

4. Entrez les commandes :

```
$ echo >| marot2.txt "\nAmour, tu as esté mon maistre\n\
Ie t'ay seruy sur tous les Dieux;"
```

   Que contient maintenant le fichier « `marot2.txt` » ?

5. Finalement, entrez les commandes :

```
$ cat >> marot2.txt <<- HEREDOC
        O si ie pouuois deux foys naistre,
            Comme ie te seruiroys mieulx!
HEREDOC
```

   en préfixant les lignes du poème de **tabulations**, non d'espaces. Que contient maintenant le fichier « `marot2.txt` » ?

6. Entrez la commande « `cat marot1.txt marot2.txt >| marot.txt` ». Que contient le fichier « `marot.txt` » ?

7. On peut insérer le résultat d'une commande *shell* dans une autre commande par le mécanisme de **substitution** (section « `Command Substitution` » du manuel de « `bash` »). Comparez le résultat des commandes « `cat marot.txt` », « `echo $(cat marot.txt)` », « `echo "$(cat marot.txt)"` », et « `echo '$(cat marot.txt)'` ». Qu'observez-vous ?

8. Interprétez le résultat des questions précédentes à l'aide de la section « `QUOTING` » du manuel de « `bash` ».

Pour aller plus loin : `http://shellhaters.org/` (en particulier la section "Language").

3 Révisions de langage C

Pour vous rafraîchir la mémoire, vous pouvez par exemple consulter Gustedt, Jens. 2020. *Modern C.* Shelter Island, NY : Manning Publications Co., disponible en ligne : `https://modernc.gforge.inria.fr/`.

Exercice 8 – *Dictionnaires et listes chaînées*

Un **dictionnaire** est une structure de données qui associe des **valeurs** à un ensemble de **clefs**, chaque clef étant associée à au plus une valeur. Un dictionnaire généralise ainsi un tableau, les indices jouant le rôle des clefs ; contrairement aux tableaux, toutes les clefs consécutives ne sont pas nécessairement présentes dans un dictionnaire.

Soit le fichier d'en-têtes suivant (également disponible sur le `git`) :

```c
/* lldict.h */

typedef struct lldict lldict;

struct lldict {
        int llkey;
        int llval;
        lldict* llnxt;
};

void lldshw(lldict* const dic);
lldict* lldadd(lldict* dic, int key, int val);
lldict* lldmod(lldict* dic, int key, int val);
lldict* llddel(lldict* dic, int key);
int lldlkp(lldict* const dic, int def, int key);
```

Chaque maillon de la liste chaînée représente une paire clef-valeur du dictionnaire. On exige que les maillons de la liste soient ordonnés par ordre croissant de paramètre « `llkey` », et que deux maillons différents possèdent des paramètres « `llkey` » différents. Toutes les fonctions sont à écrire dans un fichier « `lldict.c` » et à tester dans un fichier « `testlldict.c` » qui incluera le fichier « `lldict.h` ».

1. Dans un fichier « `lldict.c` », implémentez la fonction « `lldshw` », qui affiche toutes les paires clef-valeur d'un dictionnaire. Consultez « `man 3 printf` » au besoin.
2. Implémentez la fonction « `lldnew` », qui crée un maillon représentant une paire clef-valeur du dictionnaire, avec le champ « `llnxt` » initialisé à la valeur « `NULL` ». Consultez « `man 3 malloc` » au besoin. Si l'allocation en mémoire a échoué, le programme terminera sur une erreur (« `man 3 exit` »).
3. Implémentez la fonction « `lldadd` », qui ajoute une paire clef-valeur à un dictionnaire si la clef n'y est pas déjà présente, et renvoie un pointeur vers le premier maillon de la liste chaînée. Si la clef est déjà présente, la fonction ne doit pas modifier le dictionnaire.
4. Implémentez la fonction « `lldmod` », qui modifie la valeur associée à une clef déjà présente dans le dictionnaire, et renvoie un pointeur vers le premier maillon de la liste chaînée, ou « `NULL` » si le dictionnaire est vide. Si la clef n'est pas présente, la fonction ne doit pas modifier le dictionnaire.
5. Implémentez la fonction « `llddel` », qui supprime une paire clef-valeur d'un dictionnaire et renvoie un pointeur vers le premier maillon de la liste chaînée après suppression, ou « `NULL` » si le nouveau dictionnaire est vide. Si la clef n'est pas présente, la fonction ne doit pas modifier le dictionnaire. N'oubliez pas d'utiliser la fonction « `free` » pour désallouer la mémoire allouée avec « `malloc` ».
6. Implémentez la fonction « `lldlkp` », qui renvoie la valeur associée à la clef si celle-ci est présente, et la valeur « `def` » si la clef est absente.