TP 01 : Introduction à Linux: ligne de commande et utilisation distante

Note préalable. Le contenu de ce TP est une reprise améliorée (et (re)traduite si nécessaire) des TP des années précédentes, ainsi merci à Farzad JAFARRAHMANI, François THIRÉ, Francis HULIN-HUBARD et Stefan SCHWOON pour les versions précédentes.

Introduction. Le premier TP du cours *Architecture et Système* fait à la fois office d'introduction et de révision des connaissances de base du système d'exploitation GNU/Linux installé sur les ordinateurs de la salle de TP et que vous pourrez utiliser tout au long de votre cursus. Les notions abordées ici seront considérées comme acquises et maîtrisées pour l'ensemble des cours à venir, quelque soit le module. Ce premier TP vous invite à (re)découvrir la ligne de commande.

Note. Si vous souhaitez utiliser votre propre ordinateur et que celui-ci a été installé sous Windows, vous avez plusieurs options :

- vous connecter à distance sur la salle machine avec un logiciel comme PuTTY (`https://www.chiark.greenend.org.uk/~sgtatham/putty/`) ;
- installer une distribution Linux sur votre machine, soit :
  - (recommandé pour les débutants) en profitant de la fonctionnalité «sous-système Windows pour Linux» des Windows 10 récents (voir `https://docs.microsoft.com/fr-fr/windows/wsl/install-win10` pour l'installation) qui vous permet ensuite d'installer une distribution Linux comme Ubuntu depuis le *Windows Store* qui cohabite avec votre installation Windows – c'est une sorte de machine virtuelle ;
  - en installant Linux directement sur la machine

Terminal. Suivant l'option choisie, vous disposez de différentes manières d'ouvrir le terminal (et la ligne de commande). Sur les sessions de la salle machine, dans le menu des applications, vous pourrez trouver sous le nom de «Terminal». Dans le cas d'une installation d'une distribution via la fonctionnalité sous-système Windows pour Linux, le terminal est directement ouvert en ouvrant l'application correspondant à la distribution choisie.

Partie 1 - Ligne de commande

1 Premiers pas

1.1 Système de fichiers

Dans Linux tout est considéré comme un fichier. Les répertoires (*directories*, ou dossiers (*folders*)) sont eux-mêmes des fichiers, même s'ils contiennent d'autres fichiers. Pour les gérer de manière ordonnée, on les représente comme un *arbre*. Chaque nœud est un fichier, et les dossiers sont des nœuds pouvant avoir des fils. La racine du système de fichier est le répertoire racine (ou *root*), noté par `/`. Voici quelques sous-répertoires importants de la racine :

— `/home` qui contient les répertoires personnels des utilisateurs [1] ;
— `/home/<username>` votre dossier privé. Dans la plupart des *shells*, le caractère ~ est interprété comme le chemin vers votre dossier personnel [2]
— `/bin` l'endroit où la plupart des applications système de base sont stockées.

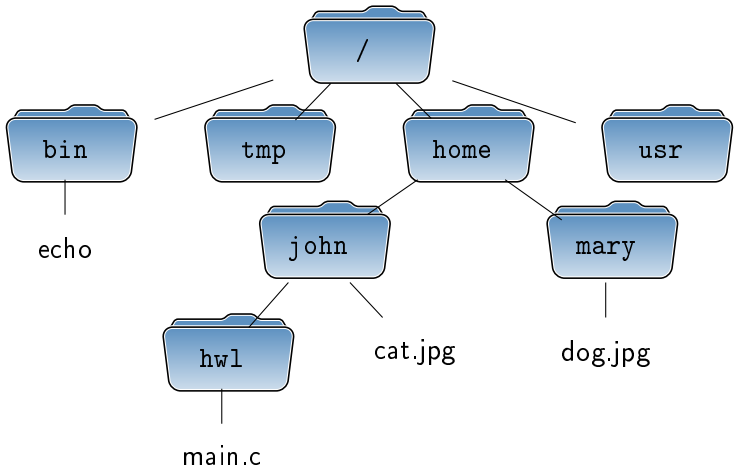

FIGURE 1 – Un exemple de système de fichiers

1.2 Shell et Terminal

Un terminal est un environnement d'entrée-sortie texte, qui peut donc lire les entrées clavier et renvoyer du texte sur l'écran.

Le terminal n'exécute pas lui-même les commandes qui y sont entrées, mais les envoie à un *shell*, un «interpréteur de ligne de commande». Ce *shell* reçoit les commandes texte et essaie de les interpréter puis de les exécuter. Il existe plusieurs types de *shells*, comme `sh`, `ksh`, `zsh`, `bash`... Le *shell* par défaut de la salle machine du département est `bash`, ce que vous pouvez normalement vérifier en utilisant la commande `echo $0`.

Voici quelques astuces utiles lorsque l'on utilise la plupart des *shells* :
— **Tabulation.** Pendant que vous écrivez une commande (ou certains paramètres reconnus des commandes), vous pouvez appuyer sur le bouton Tab () et le *shell* va essayer d'auto-compléter votre commande. S'il y a plusieurs options possibles, il ne fera rien mais un deuxième appui vous donnera l'ensemble de ces options.
— **Copier-coller.** Les raccourcis claviers usuels pour copier-coller activent d'autres fonctionnalités au sein d'un shell. En effet, Ctrl+C sert au sein d'un shell à essayer d'arrêter un programme qui serait en cours. Usuellement, les raccourcis Ctrl+Shift+C / Ctrl+Shift+V fonctionnent mais certains terminaux peuvent utiliser d'autres raccourcis.
— **Raccourcis pour les dossiers.** Comme décrit plus tôt, ~ est un raccourci pour récupérer le chemin vers votre répertoire personnel. D'autres raccourcis utiles sont `.` (un point), le répertoire actuel et `..` (deux points) le repértoire parent.

2 Premiers pas avec la ligne de commande

Le *shell* est donc un interpréteur de ligne de commandes : pour qu'il fasse quelque chose, il faut donc entrer une commande et l'envoyer à l'interpréteur en appuyant sur entrée. ▶ Par exemple, entrez la commande `ls` : que fait-elle ?

1. Certains utilisateurs spéciaux, comme l'administrateur `root`, ont leur «répertoire personnel» ailleurs dans le système de fichier
2. On peut aussi utiliser `~<username>` pour obtenir le chemin vers le dossier personnel de l'utilisateur `<username>`

Certaines commandes ont des *flags* (drapeaux), qui sont généralement indiqués par un tiret suivi d'un certain nombre de caractères alphanumériques. Ces drapeaux modifient le comportement du programme qui sera invoqué. ► Par exemple, qu'est-ce qui change lorsque l'on exécute `ls -l` ? Certains programmes peuvent aussi être invoqués avec des options plus longues, qui commenceront par deux tirets (par exemple, `ls --group-directories-first`).

2.1 Trouver de l'aide

Il est possible de trouver des informations sur l'utilisation de nombreux programmes sans quitter la ligne de commande en utilisant la commande `man` (pour manuel) qui donnera l'accès à la documentation du programme (ou de la commande) donné(e) en argument.

La navigation à l'intérieur de `man` se fait avec un ensemble de raccourcis claviers. En particulier, vous pouvez utiliser les touches directionnelles ou PageUp/PageDown pour naviguer au sein du contenu du manuel, et vous devrez utiliser la touche [q] pour quitter le manuel. Pour effectuer une recherche, appuyez sur /, entrez la chaîne que vous recherchez[3] puis entrée, et vous pourrez naviguer entre les différents résultats trouvés avec les touches [n] (next) et [p] (previous).

2.1.1 Man pages

► Lancez la commande `man man`.
Toutes les pages dans **man** sont organisées de la même manière :

1. NAME : nom de la commande ;
2. SYNOPSIS : résumé court de la syntaxe de la commande ;
3. DESCRIPTION : description, longue ou courte, de la commande ;
4. OPTIONS (facultative) : description des options supportées ;
5. Un certain nombre de sous-sections (selon les besoins de la personne ayant fait le manuel) ;
6. AUTHOR : le, ou les, personnes ayant développé la commande ;
7. SEE ALSO : des références croisées.

2.2 Manuel de la commande `ls`

À titre d'exemple, ► lisez la page d'aide de la commande `ls` et essayez d'y trouver des informations répondant aux questions suivantes :

1. ► en 10 mots (au maximum), à quoi sert cette commande ?
2. ► quelle option permet d'afficher l'ensemble des fichiers, y compris ceux "cachés" (ceux dont le nom commence par un point) ?
3. ► quelle option permet d'afficher les fichiers avec un format long en liste ?
4. ► quelle option permet d'afficher la taille des fichiers sous un format facilement lisible par un utilisateur ?
5. ► quelle option permet d'afficher récursivement le contenu d'un répertoire ?

2.3 Quelques autres moyens d'obtenir de l'aide

Voici quelques autres options à essayer pour obtenir de l'aide :
— Utiliser l'option `--help` d'une commande vous fournit généralement une courte explication de celle-ci (par exemple, essayez `mkdir --help`) ;

3. Plus exactement, une expression régulière correspondant à ce que vous recherchez ; c'est probablement un petit peu tôt pour en parler en détail cependant

— La commande `whatis` renvoie une description en une ligne de la commande fournie en argtument (par exemple, essayez `whatis ls`) ;
— Utiliser votre moteur de recherche favori sur internet (comme DuckDuckGo) ;
— Demander à quelqu'un.

2.3.1 *Wildcards*

Un *wildcard*[4] est un signe qui prend la place d'un caractère inconnu ou d'un ensemble de caractères. Les *wildcard* les plus fréquemment utilisés sont :
— l'astérisque `*` représente un nombre quelconque de caractères inconnus. Par exemple, `gâteau*` peut désigner `gâteau au chocolat`, `gâteau` ou `gâteau au yaourt` et `*fraises` peut désigner `tarte aux fraises`, `deux fraises` ou `fraises...`
— le point d'interrogation `?` représente un seul caractère. `???te` peut représenter `frite` ou `tarte`.
► Par exemple, utilisez l'un de ces symboles pour obtenir la liste des fichiers démarrant par "u" à la racine de votre système.

2.3.2 Manuel de la commande `mkdir`

Imaginez que l'on veuille créer un sous-dossier dans votre dossier personnel avec le chemin `ArchiSys/tp/01`.
► Lancez la commande `mkdir ArchiSys/tp/01` et observez son retour ;
► Regardez dans la *man page* de `mkdir` comment créer toute la hierarchie de dossiers en une commande.

2.3.3 La commande `echo`

► À quoi sert la commande `echo` ?
► Que fait la commande suivante ? Pourquoi ?

```
1 echo -ne "\n\n *\tHello World $LOGNAME\t\t*\n\n"
```

► Essayez la commande suivante. Quelle est la différence ? Pourquoi ?

```
1 echo -e '\n\n *\tHello World $LOGNAME\t\t*\n\n'
```

2.3.4 D'autres commandes

Dans la suite de ce TP, on utilisera aussi la commande `ps` et `cat`. Que font ces commandes ? Si vous n'avez pas l'habitude de ces commandes, prenez le temps de regarder leurs *man pages*.

3 Redirections

Les commandes que l'on a vu jusque-là envoient leur sortie sur la sortie standard (`stdout`, pour *standard output*). Par défaut, il s'agit de votre terminal. Il est possible de changer la destination de la sortie d'une commande, et par exemple enregistrer directement le résultat d'une commande dans un fichier (ce qu'on appelle une *redirection*.

Notons que certains programmes peuvent aussi produire des sorties dans d'autres places que la sortie standard, comme la sortie d'erreur (`stderr`, pour *standard error output*).

En général, les programmes fonctionnent comme suit :

4. Je n'ai pas su me décider entre caractère ou signe générique, caractère de substitution ou caractère de remplacement...

— si un ou plusieurs fichiers est spécifié dans les arguments du programme, il fonctionnera sur ces fichiers ;
— si aucun fichier n'est spécifié, ce sera l'entrée standard qui sera utilisée (`stdin`, pour *standard input*)

3.1 Premiers pas avec les redirections

► Exécutez chacune des commandes suivantes (dans l'ordre) dans un terminal, regardez leurs résultats et déduisez ce qui se passe (il sera nécessaire d'observer les contenus des fichiers produit après l'exécution de chaque commande) :

```
ls
ls > file1
pwd > file1
ps aux > file2.txt
cd > file3
cat file1 file2.txt > file4
cat file1 >> file1
pwd >> file1
```

► Que font les mots-clefs > et >> utilisés dans ces exemples ?

Lorsque l'on a pas besoin de la sortie d'une commande, on peut l'envoyer dans le fichier spécial `/dev/null`.

3.2 Le *pipe*

Le *pipe* (littéralement, tuyau) est aussi une forme de redirection, mais cette fois-ci au lieu de rediriger la sortie standard vers un fichier comme précédemment, on la redirige (ou connecte) sur l'entrée standard de la commande suivante.

3.2.1 `cat`

► Exécutez la commande `cat` dans un terminal. Écrivez quelques mots, puis pressez Enter. La ligne est dupliquée. Pourquoi ?

Lorsque l'entrée utilisée pour une commande est l'entrée standard (celle du terminal), vous pouvez utiliser Ctrl + D pour indiquer la fin de celle-ci.

(Petite note utile, l'abréviation `^D` désigne Ctrl + D)

3.2.2 Exemples de *pipe*

► Si vous ne connaissez pas encore ces commandes, jetez un coup d'œil rapide à la description et/ou au manuel de `grep` et `less`.

► Exécutez chacune des commandes suivantes (dans l'ordre) dans votre terminal, regardez les contenus des fichiers manipulés et déduisez ce qui se passe :

```
ls -R > file1
less file1
cat file1
cat file1 | less
ps aux
ps aux | grep -v root
ps aux | grep root
ps aux > file2
cat file2 | grep -v root
```

► Comment enregistrer (en une suite de redirections) la liste des processus n'appartenant pas à `root` dans un fichier ?

3.3 Redirection d'entrées

Il est aussi possible de rediriger l'entrée d'une commande. ► Que fait la commande suivante ?

```
cat <<eob
a
b
c
eob
```

En résumé,
— < permet d'envoyer un fichier spécifié dans l'entrée du programme ;
— << permet d'envoyer du texte multi-ligne en entrée de programme (► que fait son argument, dans ce cas ?) ;
— <<< permet d'envoyer une chaîne de caractères en entrée de programme.

4 Gestion des processus

La gestion des processus (*process management*) désigne ici la manipulation de programmes en cours d'exécution depuis le terminal. Il existe aussi des interfaces graphiques, en particulier sous Windows, qui font ce que l'on va voir dans cette section.

Un programme en cours d'exécution est appelé un *processus*. Les processus sont identifiés par le système par un identifiant nommé le **PID** (*Process IDentifier*).

Sous Unix, dans la ligne de commande, un processus peut être exécuté de deux manières :
— en arrière-plan (in the *background*) ou en tâche de fond : lorsqu'un processus est exécuté en arrière-plan, il est possible de lancer d'autres processus dans le shell. Le processus en arrière-plan peut toujours imprimer sa sortie dans le terminal, mais ne recevra pas d'entrée depuis le clavier ;
— en avant-plan (in the *foreground*) : c'est le fonctionnement normal. Ce processus reçoit l'entrée clavier, et le shell va attendre que le processus soit terminé avant de laisser la possibilité de lancer une nouvelle commande.

Nous allons voir qu'il est possible de basculer un processus d'un état à l'autre dans le terminal, et d'en terminer certains. Nous allons aussi voir qu'il est possible de lister les processus en cours d'exécution dans un terminal grâce à la commande `jobs`.

4.1 Processus en arrière-plan

⚠Pour ceux qui ont installé leur ligne de commande via le sous-système Windows pour Linux (en attendant des identifiants pour la salle machine), et qui n'ont pas installé de serveur graphique (serveur X), il sera conseillé de remplacer `xterm` par une commande qui ne termine pas comme `ping 8.8.8.8`. Vous ne pourrez pas faire toutes les questions mais vous pourrez tout de même expérimenter. Observez sur les machines du département (ou une autre machine ayant une distribution Linux graphique entièrement installée).

En ajoutant une esperluette `&` à la fin d'une commande va faire que le processus créé va fonctionner en arrière-plan. ► Exécutez les commandes suivantes

```
xterm &
emacs monfichier &
```

Dans ce cas, le shell va exécuter la commande `xterm` et rendre le contrôle à l'utilisateur après avoir affiché le PID, et faire de même avec la commande `emacs`.

Voici des exemples de sorties possibles, après avoir exécuté `xterm &` :

```
[1] 1325
```

et après avoir exécuté `emacs &` :

```
[2] 1360
```

Dans ce cas, l'identifiant du processus `xterm` est 1325 et l'identifiant d'`emacs` est 1360. Respectivement, le shell (et `jobs`) vont y faire référence avec les numéros 1 et 2.

4.1.1 Basculer d'un état à l'autre

Lorsqu'un programme est lancé en avant-plan, il n'est plus possible d'interagir avec le shell qui l'a lancé. Pour de nouveau interagir avec le shell, il faut basculer le processus en arrière-plan. La combinaison Ctrl + Z va suspendre ce processus, et on peut alors utiliser la commande `bg` pour le basculer en arrière-plan et le réveiller.

Dans l'autre sens, il est possible de basculer le dernier processus envoyé en arrière-plan (par exemple en ayant ajouté un `&` après la commande) de nouveau en avant-plan avec la commande `fg`.

— ► Lancez un nouveau terminal `xterm` depuis un terminal ;
— ► Lancez la commande `ls` dans le nouveau terminal ;
— ► Suspendez le nouveau terminal (depuis le premier terminal) ;
— ► Essayez de lancer de nouveau `ls` dans le nouveau terminal, que se passe-t-il ? pourquoi ?
— ► Envoyez le nouveau terminal en arrière-plan ;
— ► Relancez `ls` dans le nouveau terminal. Que se passe-t-il ? Pourquoi ?

4.1.2 Application et états de processus

► Créez un nouveau fichier avec le nom `script1` et le code suivant à l'intérieur (utilisez `cat` et une redirection) :

```
while true ; do
 for i in {1..5} ; do
  echo bla
  sleep 1
 done
 echo -n ">> "
 read a
 echo $a
done
```

⚠ Suivant comment `cat` est utilisée, il se peut que le fichier produit ne soit pas exactement celui que vous avez écrit. À votre avis, pourquoi ? (Vous pouvez regarder la section 6 pour plus de détails)

► Exécutez la commande `bash script1` ; lorsque la ligne dans le terminal affiche >> entrez un caractère puis confirmez avec Enter .

► Après quelques exécutions, suspendez le processus. Que se passe-t-il ?

► Reprenez le processus dans l'avant-plan. Quel est le comportement de celui-ci ?
► Basculez le processus en arrière-plan. Que se passe-t-il si vous lancez `ls` lorsque >> s'affiche ?
► Reprenez le contrôle du processus script, et arrêtez le.

4.1.3 Terminer des processus

Pour terminer un processus exécuté en avant-plan, utilisez Ctrl + C dans le terminal exécutant le processus. Pour terminer un processus exécuté en arrière-plan, trouvez son pid et utilisez la commande `kill`.

4.1.4 jobs

En gardant la *man page* de `bash` à portée (sur un terminal séparé), ► exécutez les commandes suivantes (et les combinaisons de touches) et analysez le comportement des processus. Notez qu'il faudra resélectionner le terminal lorsque les applications sont ouvertes.

⚠ Pour que cela marche exactement comme indiqué dans la suite de commandes ci-dessous, ouvrez un nouveau terminal. Mais sinon, vous pouvez intuiter à partir du retour de `jobs` les corrections à effectuer (qui dépendent de ce que vous avez fait avant).

```
emacs &
xterm &
firefox &
emacs
Ctrl + C
jobs
fg %3
Ctrl + Z
bg
fg %2
Ctrl + C
kill %1
jobs
```

► Quel processus est encore actif à la fin de l'exécution de cet ensemble de commandes et raccourcis clavier ?

4.2 Exécuter des processus "à la chaîne"

Lorsqu'un programme termine son exécution, il renvoie une valeur qui dépend de la manière dont le processus s'est arrêté. Cette valeur est habituellement interprétée comme indication de la réussite d'une commande. Elle peut être utilisée pour lancer des commandes à la suite des autres. On va, à cette fin, utiliser les opérateurs "&&" (ET) et "||" (OU).

Plus précisément, en utilisant "&&" la commande à droite de ces symboles sera exécutée si et seulement si la commande à gauche a réussi. En utilisant "||", la commande à droite ne sera exécutée que si la commande se trouvant à gauche a échoué. Par exemple,

```
mkdir dossier/autredossier 2> /dev/null || echo evaluation de la commande de
    droite
```

imprimera "évaluation de la commande de droite", alors que :

```
mkdir dossier/autredossier 2> /dev/null && echo evaluation de la commande de droite
```

n'imprimera rien sur le terminal.

Note : la redirection `2>` est similaire à `>`, mais `2>` redirigera la sortie d'erreur au lieu de rediriger la sortie standard.

4.2.1 Deux exercices

► À l'aide de la commande `grep`, testez si le compte `root` existe sur votre système et affichez le message "root existe" si c'est le cas.

On peut utiliser le fichier `/etc/passwd` ou le retour de `getent passwd` pour obtenir la liste des utilisateurs.

► Écrivez une commande qui affiche uniquement le texte "Firefox est en cours d'exécution" sur la sortie standard si Firefox est actuellement en train de fonctionner sur votre machine et "Pas de firefox lancé" sinon. (Pensez à utiliser les redirections vues plus tôt)

5 Le système GNU/Linux

5.1 Vue d'ensemble

- La racine du système d'exploitation Linux est marquée `/` ;
- Sous Linux tout est un fichier ;
- Un *utilisateur* est une personne ou programme enregistré sur le système (avec un compte) ;
- Les groupes sont utilisés pour regrouper les utilisateurs selon des critères définis par l'administrateur système ;
- Les utilisateurs peuvent appartenir à un ou plusieurs groupes ;
- Les utilisateurs peuvent avoir des «privilèges» selon leur appartenance à des groupes ;
- Pour le système, un utilisateur est un numéro (appelé UID (User Identifier)) associé à un *login* ;
- Pour le système, un groupe est aussi un numéro (appelé GID (Group Identifier)) ;
- Chaque fichier appartient à un unique utilisateur (UID) et unique groupe (GID) ;
- Les droits, ou permissions, associés à un fichier sont : soit l'absence de droits (noté `-`), le droit de lecture (`r`, *read*), le droit d'écriture (`w`, *write*), et les droits d'exécution (ou d'ouverture d'un dossier) (`x`, execution) ;
- Ces droits sont affichés (par la commande `ls -l`) sous la forme de triplets "rwx" pour chaque ensemble auquel ils peuvent être assignés (dans l'ordre suivant : utilisateur, groupe, autres utilisateurs) ;
- La commande `chmod` permet de modifier les droits d'accès d'un fichier ;
- La commande `chown` permet de modifier l'utilisateur et/ou le groupe propriétaires d'un fichier ;
- La commande `groups` vous permet d'obtenir la liste des groupes auxquels appartient un utilisateur ;
- La commande `pwd` vous permet d'obtenir le chemin complet du dossier dans lequel vous êtes actuellement.

5.2 Questions et exemples

► Quels sont les droits sur votre répertoire personnel ? À quel groupe d'utilisateur appartient-il ?

► À quel groupes votre compte utilisateur appartient-il ? À quel groupes le compte `yduplouy` appartient-il ?

► Essayez d'aller dans `/root`. Que se passe-t-il et pourquoi ?

La commande `chmod` permet d'éditer les permissions sur un fichier de deux manières :

— l'une en utilisant des symboles : le premier pour l'ensemble d'utilisateur souhaité (`u` pour utilisateur courant, `g` pour groupe, `o` pour les autres utilisateurs), le deuxième pour indiquer si l'on veut supprimer (`-`), ajouter (`+`), ou définir (`=`) des droits, et ensuite la liste des droits souhaités (identiques à ceux de la commande `ls`).
— l'autre en utilisant la représentation octale des droits ; dans ce cas, il seront définis exactement comme indiqués. Ils doivent être fournis dans l'ordre utilisateur, groupe, autres sous la forme de chiffres qui sont la somme des suivants : 4 pour lecture, 2 pour écriture, 1 pour exécution. [5]

► Créez un répertoire `mondossier`. Quels sont ses droits par défaut ?
  ► En utilisant `chmod`, changez ses droits pour qu'ils correspondent à `rwx------` puis `rwxr-x---`.
  ► Tentez de modifier les droits du répertoire `/root`. Que se passe-t-il ?

6 Variables

Sous Unix, le système est capable d'échanger ou obtenir des informations en utilisant des variables. Ces variables sont rendues accessibles par le shell. Certaines sont des variables du système d'exploitation (les *environment variables*), d'autres sont spécifiques au shell utilisé. Nous allons en étudier quelques unes, comment les modifier, les créer ou les afficher.

La déclaration d'une variable prend la forme suivante :

```
1 nomvar=valeur
```

avec `valeur` une chaîne de caractères et `nomvar` une chaîne composée des caractères alphanumériques ne commençant pas par des chiffres, comme par exemple

```
1 nom="Jean"
```

Ces variables peuvent ensuite être lues en les faisant précéder du symbole $.

6.1 Affichage

Le shell `bash` fournit la commande `echo` qui permet d'afficher ce qui est fourni en argument, en particulier ici la valeur associée au nom d'une variable. Par exemple, la commande suivante affiche la valeur associée à la variable `nom` :

```
1 echo $nom
```

Même si cette notation fonctionne, celle recommandée par `bash` est la suivante :

```
1 echo ${nom}
```

6.2 Modification

Modifier une variable consiste à écraser sa précédente déclaration.

5. En d'autres termes, si on fait la conversion en binaire, on se rend compte que c'est un *bitset*.

6.3 Quelques exemples

1. ► Affichez le contenu de la variable `HOME`. Que représente cette variable ?
2. ► De même avec la variable `PWD`. Que représente cette variable ?
3. ► Que représente la variable `PATH` ?
4. ► Affichez le contenu de la variable `PS1`. Que représente cette variable ?
5. ► Modifiez la variable `PS1` (et observez le changement sur votre shell). Ouvrez un nouveau terminal : que constatez-vous ?
6. ► Modifiez le contenu de la variable `PATH` puis essayez de lancer une commande. Que se passe-t-il ?
7. ► Modifiez le fichier de configuration de `bash` (dans votre répertoire personnel, `.bashrc` [6]) et incluez-y une définition d'une variable (par exemple `PS1`). Quelles sont les différences de comportement ?

6.4 Différences entre les chaînes de caractères

Avec le shell, vous pourrez trouver des chaînes de caractères entre *double quotes* ("bla"), *simple quotes* ('bla') ou *anti-quotes* (`bla`). Essayez les commandes suivantes :

```
echo $HOME
echo "$HOME"
echo '$HOME'
echo `$HOME`
TEST=ls ; echo "`$TEST`"
test=$(ls) ; echo $test
```

Quels sont leurs différents comportements ?

7 Scripts

Un *script* est un ensemble d'instructions ocnsignées dans un fichier, qui seront exécutées séquentiellement. Nous continuerons à utiliser l'interpréteur de commande `bash`. Tous les blocs d'instruction d'un script peuvent être testés directement dans le shell.

Commençons par créer un petit script :

```
cat > myscript
cd /tmp
ls
cd
```

(Utilisez Ctrl + D pour indiquer la fin de la saisie).

On peut alors exécuter le script depuis la ligne de commande, avec `bash myscript`. Il est aussi possible de spécifier le programme à exécuter en début de script, comme par exemple :

```
cat > myscript
#!/bin/bash

cd /tmp
ls
cd
```

6. Ce fichier n'est pas créé automatiquement sur les machines du département ; créez-le si nécessaire.

```
^d
```

Cetite première ligne, commençant par `#!`, s'appelle le *shebang*. Si vous avez un doute sur l'emplacement d'un programme, vous pouvez utiliser la commande `whereis` pour obtenir son chemin complet. (D'ailleurs, il se peut que l'emplacement de `bash` sur votre machine ne soit pas `/bin/bash` mais `/usr/bin/bash` par exemple : vérifiez avec `whereis`.

On pourra alors ensuite invoquer le script comme n'importe quel autre programme, avec `./myscript`. (Pourquoi ce `./`, d'ailleurs ?)

► Pourquoi cela ne fonctionne pas encore ? Trouver comment corriger cela.

7.1 Structures de contrôle

La *man page* `test` vous permet d'obtenir la liste des tests disponibles en `bash`

7.1.1 if then else

La syntaxe pour faire ces conditionnelles est la suivante :

```
if commande(s); then
  commande1;
  commande2;
  elif commande(s); then
  commande(s)
  else commande(s)
fi
```

► En utilisant cette structure, testez l'existence du répertoire `/var/logs`, puis du répertoire `/var/LOg` et affichez des commentaires (différents) suivant si le(s) répertoire(s) ont été trouvés ou non.

Remarque : si vous avez une longue ligne de commande, vous pouvez utiliser
à la fin d'une ligne pour passer à la ligne et continuer la commande actuelle.

► Afficher "Alerte" si vous avez le droit d'écrire dans `/etc`.

7.1.2 for do done

La syntaxe pour des boucles `for` est la suivante :

```
for var in list ; do
 commande(s)
done
```

► Écrire une boucle `for` qui affiche les lettres a, b, c, d à raison d'une par ligne

7.1.3 while . . . do . . . done

La syntaxe pour des boucles `while` est la suivante :

```
while commande(s); do
  commande(s)
done
```

► Que fait la séquence suivante : `ls | while read a ; do echo $a ; done` ?

7.2 Exercice : script "poubelle"

► Écrire un petit script qui prend en paramètre un ensemble de fichiers, et les déplace dans la poubelle (un répertoire `.trash` à la racine de votre répertoire utilisateur, qu'il faudra éventuellement créer). Affichez un message d'erreur si un fichier n'existe pas.

L'ensemble des arguments passés au script peut être récupéré avec `$*`.

Partie 2 - Utilisation distante

8 Secure Shell

L'objectif de cette partie est de donner les bases de la connexion à distance via SSH. OpenSSH est un outil de connexion à distance composé d'un client (le programme `ssh` sous Unix, PuTTY par exemple sous Windows) et d'un serveur (`sshd` sous Linux). Les fichiers de configuration d'OpenSSH se trouvent dans le répertoire `/etc/ssh/`, et les fichiers de personnalisation de l'utilisateur se trouvent dans son répertoire personnel sous le dossier `.ssh`.

8.1 Premières utilisations

La syntaxe d'utilisation du client `ssh` est la suivante sous Linux :

```
ssh [ options ] [ login@hostname ]  [ commande ]
```

► Si un fichier `~/.ssh/known_hosts` existe, renommez le (ou faites en une copie). Pour cela, vous pouvez utiliser les commandes `mv` et/ou `cp`.

► Connectez vous au serveur SSH du département (`ssh.dptinfo.ens-cachan.fr`) avec votre identifiant. Que s'affiche-t-il sur votre terminal ?

► Exécutez la commande suivante :

```
sed -i 's/.\{5\}$/aaaaa/' ~/.ssh/known_hosts
```

► puis reconnectez vous au serveur. Que se passe-t-il ? Pourquoi ?

► Utilisez la commande `ssh-keygen` pour solutionner ce problème.

► Si vous avez renommé le fichier `known_hosts` au début de cette section, restaurez-le.

Remarque pratique. Le serveur SSH du département (`ssh.dptinfo.ens-cachan.fr`) est un point d'accès aux autres machines du département ; nous allons voir plus tard comment "tunneler" une connexion SSH vers ces machines directement.

8.2 Génération d'une clef asymétrique

Remarque. Le détail du fonctionnement du chiffrage par clé sera le sujet du cours *Initiation à la cryptographie* du second semestre. On se contentera ici de donner un rapide guide.

OpenSSH est une implémentation du protocole SSH. Cette implémentation est open source et est libre sous Linux. Elle utilise la commande `ssh-keygen` pour générer des clés asymétriques. Vous pouvez utiliser sa *man page* pour trouver comment fonctionne la génération de clef.

À la fin de cette génération, il sera demandé une phrase secrète (*passphrase*) qui sera utilisé .

— ► Générez deux clefs asymétriques, l'une avec une passphrase, l'autre sans, respectivement dans les fichiers `secure` et `unsecure` de votre répertoire de configuration de `ssh`

— ► Identifiez la partie publique et la partie privée du secret asymétrique en regardant le contenu des fichiers

— ► Déterminez quelle partie de la clef est chiffrée. À quoi sert-il de chiffrer cette partie de la clef asymétrique ? Pourquoi uniquement celle-ci ?

— ► Que se passe-t-il si vous perdez la passphrase ?

8.3 Authentification par clés

Une façon de voir le cryptosystème asymétrique est d'imaginer que la partie publique de la clé est une serrure, et que la partie privée est la clé de cette serrure.

La mise en place de l'authentification par clés consiste à déposer, sur votre compte, dans votre répertoire `.ssh`, à l'intérieur du fichier `authorized_keys`, la partie publique de la clé (la serrure).

Nous préciserons ensuite au programme `ssh` quelle clé utiliser pour déverrouiller la serrure. Suivant la manière dont la clé a été générée, son accès est protégé ou non par un mot de passe.

— Déployez vos clés publiques (générées précédemment) dans votre fichier `authorized_keys` en une commande ;
— Votre répertoire personnel est partagé entre l'ensemble des machines du département : depuis une machine, essayez d'accéder au serveur ssh du département (`ssh.dptinfo.ens-cachan.fr`). Analysez ce qui se passe, et corrigez la commande en cas de besoin.
— Que font les commandes `ssh-keygen -l -f unsecure` et `ssh-keygen -l -f unsecure.pub` ? À quoi servent ces sorties ?

8.4 Personnalisation de `ssh`

Il est possible de modifier un certain nombre de paramètres de fonctionnement d'OpenSSH via le fichier `~/.ssh/config`. La *man page* de `ssh_config` donne une liste de soptions disponibles.

► Générer un fichier de configuration permettant de se connecter au serveur SSH en utilisant l'alias `secure` pour utiliser la clé protégée par mot de passe, et l'alias `unsecure` pour utiliser la clé non protégée. Tester ces configuations.

Piste : La *manpage* de `ssh_config` est plutôt longue, il faut en retenir que le fichier de configuration a la forme suivante (et trouver quels paramètres donner pour spécifier le serveur (ou la machine) sur laquelle se connecter, la clef, et éventuellement le nom d'utilisateur :

```
Host alias1
  Parametre1 blabla
  Parametre2 blibli
Host alias2
  Parametre1 blablo
  Parametre3 blublu
```

► Testez la configuration.

8.5 Utilisation d'un porte-clés

Il est possible de gérer les clefs OpenSSH asymétriques avec un porte clef nommé `ssh-agent`. Ce dernier maintient les clefs en mémoire pour éviter de devoir les recharger et éviter la resaisie des *passphrase*.

► Quels sont les avantages et inconvénients de charger une clef en mémoire ?

► Lancez l'agent et exportez les variables dans votre environnement. Trouvez comment ajouter une clef au porte-clef. Utilisez ce porte-clef pour faire une connexion SSH.

8.6 Différents modes de connexion

Pour introduire cette section, nous allons commencer par une description (plus que) simplifiée du fonctionnement réseau. Une machine connectée au réseau peut être vue comme un "bloc" de 65535 prises. Ces prises, en terminologie réseau, sont appelées des ports. Un programme pouvant être accédé par le réseau va utiliser une de ces prises, ou en des termes plus officiels «écouter» sur un port. Sauf cas particuliers, un port ne peut être utilisé que par un programme à la fois. Les 1024 premiers ports sont réservés par le système, et les ports suivants peuvent être utilisés par les utilisateurs. Un certain nombre de ports

sont réservés pour des services particuliers. Par exemple, le port `22` est celui réservé pour SSH, le port `80` pour le protcole http, le port `443` pour le port https, etc. Les ports que nous utiliserons plus loin pour les redirections de ports devront être «libres», c'est-à-dire utilisés par aucun des programme, et non réservés (donc supérieurs à `1024`).

Les techniques dont on va parler dans cette sous-section vous permettront de vous connecter depuis l'extérieur sur les machines de la salle 1S53, ou plus généralement de vous connecter sur un réseau depuis un autre au travers d'une machine accessible en SSH.

8.7 Connexion par rebond

La façon la plus "bête" de se connecter dans une machine locale inaccessible de l'extérieur est de se connecter en `ssh` vers une passerelle et, sur le shell de cette passerelle, se connecter à la machine locale. Cela ne permet pas d'exporter les applications graphiques (option `-X` de `ssh`) ni d'effectuer directement les copies de fichiers sur la machine cible.

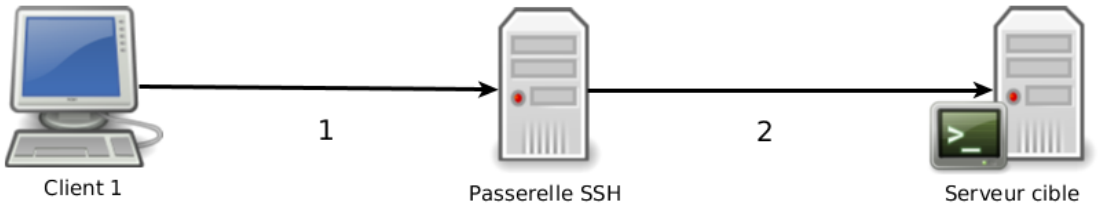

FIGURE 2 – Sans pivot

► Connectez-vous de cette manière sur une autre machine du département (`00.dptinfo.ens-cachan.fr`, en remplaçant `00` par le numéro d'une des machines fonctionnelles) en passant par la passerelle SSH du département (`ssh.dptinfo.ens-cachan.fr`)

8.7.1 Redirection de port

La redirection de port permet de connecter le port x d'une machine distante sur le port y de la machine locale, au travers d'une tierce-machine.

1. Le poste client se connecte à la passerelle SSH pour lui demander de créer un tunnel redirigeant le port x du serveur cible sur le port y de la machine locale ;
2. La passerelle SSH assure la mise en place de ce tunnel, qui restera actif tant que la connexion restera ouverte ;
3. À l'aide d'un autre terminal sur le poste client, se connecter sur l'entrée du tunnel qui va alors rediriger le trafic directement sur le port du serveur cible.

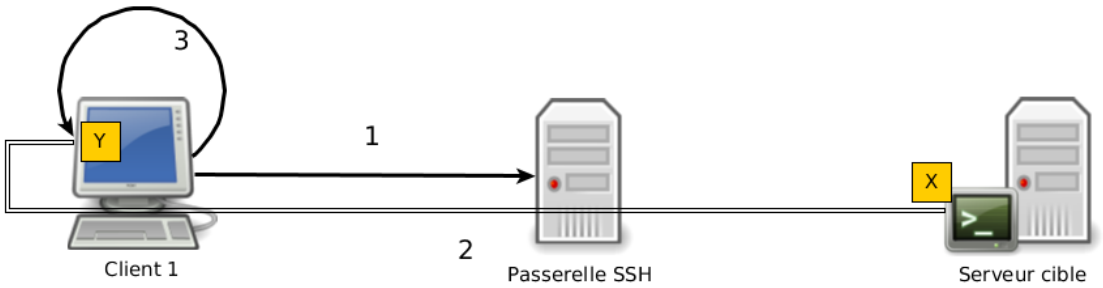

FIGURE 3 – Redirection de ports

Comme vu précédemment, en tant que simple utilisateur, le port y doit être un port non privilégié (supérieur à 1024).

Par exemple, la commande suivante va rediriger le port ssh (22) de la machine `13.dptinfo.ens-cachan.fr` sur le port `2222` de la machine sur laquelle la commande est exécutée, au travers du serveur SSH du département :

```
ssh -L 2222:13.dptinfo.ens-cachan.fr:22 login@ssh.dptinfo.ens-cachan.fr
```

Il sera alors possible de se connecter à `13.dptinfo.ens-cachan.fr` en empruntant le tunnel :

```
ssh -p 2222 login@localhost
```

► Connectez-vous sur une autre machine en utilisant une redirection de port SSH sur le port `1000` puis le port `2000`. Que se passe-t-il ?

8.7.2 Pivot

La méthode du pivot vous permet d'accéder à une machine au travers d'une (ou plusieurs) autres de manière quasiment transparente.

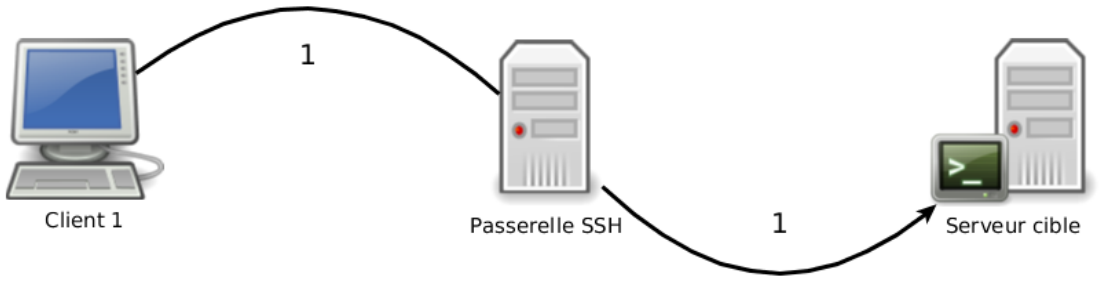

FIGURE 4 – Méthode du pivot

Contrairement à la connexion par rebond, une fois la configuration effectuée, l'utilisation de la passerelle sera transparente pour l'utilisateur. La connexion étant redirigée à la manière d'une redirection de port, l'export X fonctionne.

On peut effectuer cette méthode de deux manières :

— par le fichier de configuration de ssh : l'option `ProxyCommand` ajoutée à la définition d'un alias dans le fichier de personnalisation du client `ssh` vous permet de spécifier la machine utilisée comme pivot.

```
Host <alias_machine>
Hostname <machine.cible>
User <login_sur_machine_cible>
ProxyCommand ssh <login>@<machine.ssh.pivot> -W %h:%p
```

avec `<alias_machine>` le nom utilisé pour appeler cette configuration, `<machine.cible>` la machine sur laquelle nous souhaitons arriver, `<login_sur_machine_cible>` le login à utiliser sur la machine cible (s'il n'est pas précisé, le login de la machine courante sera utilisée), et `<login>@<machine.ssh.pivot>` les paramètres de connexion sur la machine pivot.

► Configurez un alias vous permettant de vous connecter sur une machine du département informatique à travers le serveur SSH. Ajouter l'utilisation de la clé SSH «secure» précédemment générée.

Remarque : vous pouvez utiliser les alias définis dans la création de nouveaux alias.

— via les nouvelles fonctionnalités de `ssh` : si vous disposez d'une version assez récente [7] d'openSSH, vous pouvez disposer de l'option `-J` qui permet de spécifier un pivot lors d'une connexion. On peut alors écrire :

```
ssh -J <login>@<machine.ssh.pivot> <login_sur_machine_cible>@<machine.
    cible>
```

► Si vous disposez de cette version assez récente, testez avec cette option la connexion à une autre machine du département au travers du serveur SSH du département.
Remarque : cette option peut aussi être invoquée dans le fichier de configuration SSH via `ProxyJump login@machiinessh`.

8.7.3 Proxy Socks

Il est aussi possible d'utiliser un serveur SSH comme serveur proxy (mandataire) : il est possible de faire passer le trafic web par ce serveur SSH. En utilisant le serveur SSH du département, cela vous permettra de naviguer comme si vous vous trouviez dans les murs de l'ENS Paris-Saclay (et donc d'utiliser les applications administratives bloquées de l'extérieur).

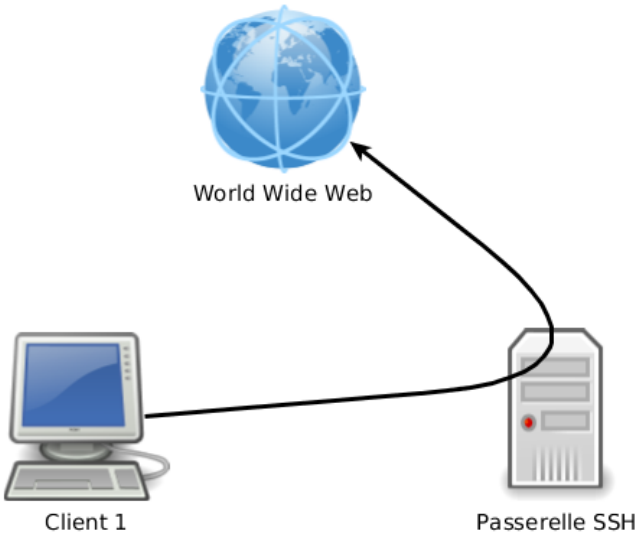

FIGURE 5 – Proxy SOCKS

— ► Cherchez un site affichant votre adresse IP avec votre moteur de recherche favori ;
— ► Dans un terminal, utilisez l'option `-D <portnonréservé>`pour créer un proxy local.
— ► Configurez votre navigateur internet pour utiliser vote proxy local. Sous Firefox :
  — Ouvrez les Paramètres, dans l'onglet Général ; naviguez jusqu'en bas de la page, vous devriez trouver les paramètres réseau. Cliquez sur Paramètres.
  — Dans la nouvelle fenêtre, activez "Configuration manuelle" (ou l'équivalent en anglais), puis renseignez les champs Socks Host (avec `127.0.0.1` ou `localhost`) et Port (avec le port choisi précédemment)
  — Validez ces nouveaux paramètres

  Vérifiez que ces paramètres ont été pris en compte avec le site affichant votre adresse IP. Que constatez-vous ?
— ► Fermez la connexion ssh ouverte (vous devrez peut-être forcer la fermeture avec Ctrl + C), puis rafraîchissez la page web affichant l'IP. Que se passe-t-il ? Pourquoi ?

7. au moins 7.3 ; la version installée sur les machines du département n'est pas assez récente, la section 10.1 parlera de comment installer localement un programme

— ► Quel est l'intérêt d'un Proxy ?

9 Multiplexeur de terminal

L'outil `tmux` permet d'ouvrir plusieurs instances du shell dans un même terminal, ce qui peut être pratique via `ssh`.

— ► Lancez `tmux` dans un terminal
— ► Lancez la commande `top`
— ► Ouvrez une nouvelle instance du terminal avec Ctrl + B (cela indiquera que la suite devra être écoutée par `tmux`), puis C (*create*)
— ► Listez le contenu du répertoire courant
— ► Retourner sur le shell exécutant `top` avec Ctrl + B suivi de P . Utilisez Ctrl + B suivi de N pour revenir sur le shell suivant
— ► Découpez l'écran verticalement (avec Ctrl + B suivi de ")
Vous pourrez naviguer entre les deux instances avec Ctrl + B suivi de ↑ ou Ctrl + B suivi de ↓ .
— ► Découpez l'écran horizontalement (avec Ctrl + B puis %)
Vous pourrez naviguer entre les deux instances avec Ctrl + B suivi de ← ou Ctrl + B suivi de → .
— ► Détachez le tmux du terminal (avec Ctrl + B puis D)
— ► Rattachez le tmux au terminal (`tmux a`)
— ► Vérifiez que `top` fonctionne toujours

Partie 3 - Conseils pratiques (et applications)

Cette partie est principalement une partie d'ouverture, un peu moins guidée que les précédentes. N'hésitez pas à demander de l'aide ou des éclaircissements.

10 Installation de programmes

10.1 Un premier exemple

En tant qu'utilisateur sur les machines du département, vous n'avez pas accès aux commandes d'administration classiques. En particulier, vous ne pourrez pas installer de programme directement avec le gestionnaires de paquet. Il faudra donc installer localement, sur votre compte, les programmes (ou librairies) dont vous aurez besoin. Nous allons dans cette section prendre comme exemple l'installation d'une version plus récente d'openSSH.

- ► Télécharger les sources d'OpenSSH dans un répertoire (par exemple, `openssh-8.7`) avec l'aide de la commande `wget`.
  Le fichier compressé des sources de la version 8.7 est disponible sur `http://ftp.fr.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-8.7p1.tar.gz`
- ► Avec les commandes `tar` et/ou `gz`, décompresser ce fichier (notez la double extension) ; notez qu'il est possible de le faire en une seule commande
- Les fichiers `README` et `INSTALL` donneront quelques pistes supplémentaires, ► Utilisez le script `configure` inclus dans le contenu du fichier compressé pour préparer l'installation, en précisant qu'il faut l'effectuer dans le dossier ~/Applications
  À votre avis, quel est l'objectif de ce script ?
- ► Une fois cette "configuration" faite, compilez les sources avec `make`. Quelle option utiliser pour lancer plusieurs étapes de compilation en parallèle ?
- ► Installez le programme (de nouveau avec `make`, mais avec un argument). Que se passe-t-il ? Corrigez [8].
- ► Vérifiez la version de `ssh`. Commentez le résultat.
- ► Ajoutez le répertoire ~/Applications/bin au PATH.
- ► Vérifiez la version de `ssh`. Commentez le résultat.
- ► Ouvrez un nouveau terminal, et vérifiez la version de `ssh`. Comment faire pour qu'à l'ouverture d'un nouveau terminal, la nouvelle version soit détectée ?

Avec la nouvelle version, vous pourrez désormais expérimenter l'option `-J` de `ssh` ainsi que `ProxyJump` dans le fichier de configuration si vous ne l'aviez pas déjà fait à la section 8.7.2.

11 Pour aller plus loin...

Pour aller plus loin avec la ligne de commande (et tester vos connaissances sur un jeu d'énigmes), vous pouvez aller voir le jeu présenté sur le lien suivant : `https://overthewire.org/wargames/bandit/`. La réussite du niveau n débloque le niveau $n+1$, et il y a 34 niveaux. Ce jeu utilisera des commandes que nous n'avons pas forcément vues.

Attention : Ce jeu n'est pas accessible depuis les machines du département (ni depuis eduroam dans l'ENS Paris-Saclay), il en est fait mention surtout à titre d'ouverture.

8. Il faudra refaire plusieurs étapes précédentes mais avec une correction