Pablo Rauzy <pr@up8.edu>
https://pablo.rauzy.name/teaching/gil

Gestion d'identité en ligne

TP Console : Travailler dans le terminal

Objectifs de ce TP, pour se préparer pour la suite du semestre :
- Se repérer dans l'arborescence des fichiers et dossiers.
- Connaître les rudiments de l'utilisation d'un *shell* (interpréteur de commandes).
- Savoir se documenter.
- Éditer des fichiers textes avec nano.
- Travailler à distance avec SSH.
- Garder des sessions persistantes avec tmux.

Exercice 0.

L'arborescence des fichiers et dossiers.

1. Pour être utilisables, les systèmes informatiques sont construits par *couches d'abstractions*. Tout en bas, on a le matériel de stockage (disque dur, SSD, etc.) sur lequel des données sont écrites en binaire. Une des premières abstractions consiste en un *système d'adressage* par *secteurs*, pour se repérer sur le matériel : il s'agit de découper virtuellement le matériel en petites zones de tailles égales et de les numéroter afin d'associer à chacune une adresse (son numéro). Bien sûr cette abstraction n'est pas suffisante pour les humain·es. Plus haut dans les couches d'abstraction, on retrouve donc un *système de fichiers*, qui va servir d'interface en permettant d'associer à un nom qui fait sens pour les humain·es, celui d'un *fichier*, une suite de secteurs contenant les données correspondantes à ce nom.

   Cela n'a pas toujours été le cas, mais les systèmes de fichiers modernes sont organisés en *arborescence* grâce au concept de *dossiers* :
   - un dossier "contient" (c'est-à-dire qu'il correspond à une liste de) des fichiers et des dossiers ;
   - il y a un dossier spécial qui s'appelle la *racine* ;
   - à partir de là on peut voir chaque dossier comme un embranchement, et chaque fichier comme une feuille d'un arbre (d'où le terme "arborescence").

   Imaginez la situation suivante :
   - le dossier racine contient trois dossiers : "foo", "bar", et "baz" ;
   - "foo" contient un dossier "abc" et les fichiers "a", "b", et "logo.png" ;
   - "abc" est vide ;
   - "bar" contient les dossiers "toto" et "lala" ;
   - "toto" contient les fichiers "toto.jpg", "titi.jpg", et "tutu.jpg" ;
   - "lala" contient les fichiers "lolo.txt" et "lulu.txt" ;
   - "baz" contient un dossier "site" et un fichier "README.md" ;
   - "site" contient les dossiers "images" et "styles" et les fichiers "index.html", "cv.html", "projets.html", et "liens.html" ;
   - "images" contient les fichiers "logo.png", "card.png", et "photo.jpg" ;
   - "styles" contient les fichiers "styles.css" et "Luciole.ttf".

   → Dessinez l'arborescence décrite ci-dessus.

2. On appelle un *chemin* l'adresse d'un fichier sur le système de fichier, c'est à dire la suite de dossier qu'il faut traverser dans l'arborescence pour arriver au fichier.

   Les chemins sont écrit en ajoutant un slash (`/`) après chaque nom de dossier pour le séparer du suivant.

   Un chemin peut être *relatif* si il ou *absolu*.

   Il est absolu si il part de la racine (et dans ce cas il commence par un slash).

   Il est relatif si il part d'un autre emplacement (dossier) dans l'arborescence.

   On appelle *dossier de travail* (*working directory* en anglais) le dossier courant, celui dans lequel on se trouve.

   Pour remonter d'un cran dans l'arborescence, on utilise un nom de dossier virtuel "`..`" qui est un alias pour le dossier parent (sauf à la racine ou c'est un alias pour elle-même puisqu'elle n'a par définition pas de parent).

   → Quels sont les chemins absolus des fichiers "README.md", "titi.jpg", et "logo.png" ?

3. Admettons que notre dossier de travail soit le dossier "site".

   → Quel est le chemin absolu de notre dossier de travail ? Quels sont les chemins relatifs des fichiers "README.md", "titi.jpg", et "logo.png" ?

Exercice 1.

Le terminal et le shell.

1. À l'origine, un *terminal* était un équipement informatique connecté à un ordinateur et disposant typiquement d'un écran et d'un clavier pour interagir avec le système.

   Aujourd'hui ce qu'on appelle un terminal n'est plus un équipement physique mais un logiciel simulant cet équipement.

   Un *shell* est un logiciel permettant d'interagir avec le système au moyen de *commandes*. C'est le type de logiciel qui est lancé par défaut dans un terminal. Généralement il s'agit du logiciel Bash, mais ce n'est pas le seul existant.

   Un shell fonctionne le plus souvent sur le principe d'une boucle "lecture, évaluation, écriture" (REPL en anglais pour *read-eval-print loop*).

   Pour cela il affiche une *invite de commande* (un *prompt* en anglais) sur le terminal, c'est-à-dire un court texte contenant quelques informations et après lequel vous pouvez, grâce au terminal qui fait le lien avec votre clavier, entrer une *ligne de commande*.

   Une fois la commande écrite et envoyée au shell par le terminal, le shell *exécuter* la commande. L'exécution d'une commande peut prendre différentes formes suivant le type de programmes impliqués. Quoi qu'il en soit, le terminal servira d'affichage pour la sortie du programme, et quand l'exécution de la commande sera terminée, le shell reprendra la main sur le terminal pour afficher de nouveau son invite de commande et ainsi permettre de lancer la commande suivante.

   → Ouvrez un terminal, et exécutez les commandes `date`, puis `whoami`, puis `hostname`. Que semblent faire ces commandes d'après vous ? En quoi consiste votre prompt ?

2. Les commandes peuvent prendre des *arguments*. La syntaxe pour cela est d'écrire les arguments à la suite de la commande après un espace, en les séparants eux-mêmes par des espaces. Certains de ces arguments peuvent être des *options* qui changent le comportement du programme appelé, en général les options commencent par un tiret (`-`) et doivent apparaître avant les autres arguments.

   Par exemple la commande `echo` fait simplement s'afficher sur le terminal ce qu'on lui passe en argument.

   → Faites s'afficher sur votre terminal "Salut tout le monde".

3. Voici quelques commandes permettant d'interagir avec le système de fichier :
   - `pwd` (pour *print working directory*) : affiche le chemin absolu du dossier courant.
   - `cd` (pour *change directory*) : change le dossier courant en allant dans celui dont on lui passe le chemin en argument. À défaut d'argument elle retourne dans le dossier de travail par défaut : celui qu'on appelle le "home" de l'utilisateur·ice et dont le chemin absolu est abrégé par le symbole tilde (`~`).
   - `ls` (pour *list*) : affiche la liste des fichiers et dossier du dossier courant ou de celui dont on lui passe le chemin en argument.
   - `mv` (pour *move*) : déplace ou renomme le fichier dont le chemin est passé en premier argument vers l'emplacement donné par le second argument.
   - `cp` (pour *copy*) : copie le fichier dont le chemin est passé en premier argument vers l'emplacement donné par le second argument.
   - `rm` (pour *remove*) : supprime le fichier dont le chemin est passé en argument.
   - `mkdir` (pour *make directory*) : crée le dossier dont le chemin est passé en argument.
   - `rmdir` (pour *remove directory*) : supprime le dossier dont le chemin est passé en argument.

   → Créez un dossier "semestre-1" dans votre home, déplacez-vous dedans, puis créez un dossier "gil" pour ce cours, déplacez vous dedans. Quel est le chemin absolu de votre dossier de travail à présent ? Comment l'écrire de façon abrégée ?

4. Pour vous documenter sur une commande, vous pouvez afficher son aide en l'appelant avec l'option `--help` ou `-h`, ou vous pouvez consulter sa page de manuel avec la commande `man` à laquelle vous passez le nom de la commande en argument.

   → Consultez le manuel de la commande `ls`. Comment lui demander d'afficher tous les fichiers y compris les fichiers cachés (ceux dont le nom commencent par un point) ?

Exercice 2.

L'édition de fichiers textes avec Nano.

1. Au moins pendant vos études dans cette licence, vous allez passer énormément de temps à programmer. Cela implique notamment d'éditer des fichiers textes (du code écrit dans différents langages de programmation).

   Il existe de nombreux éditeurs de texte. Le bloc-note de Windows est probablement ce qui se fait de plus basique. Il existe aussi des éditeurs très puissants que vous ne manquerez pas d'utiliser **plus tard**.

   Pour l'instant je vous recommande **fortement et avec insistance** de vous contenter d'utiliser un éditeur de texte intermédiaire, avec suffisamment de fonctionnalité pour être à l'aise pour la programmation, mais qui ne fera pas tout pour vous pour autant.

Sinon, vous risquez de vous rendre dépendant·es d'outils qui font une partie du travail pour vous et d'être coincé·e le jour où ces outils dysfonctionnent ou ne sont pas disponibles sur votre station de travail. Utiliser des outils simples et faire les choses par vous-même et par ailleurs la meilleure façon d'apprendre.

L'éditeur de texte nano est un programme TUI (*terminal user interface*), c'est à dire qui s'exécute dans le terminal. Il est installé par défaut sur presque tous les systèmes et correspond parfaitement à ce que nous cherchons : très basique par défaut et simple d'utilisation avec son aide intégrée, peut facilement être configuré pour offrir les fonctionnalités nécessaires à programmer confortablement (affichage des numéros de lignes, coloration syntaxique, indentation automatique), et peut s'avérer assez puissant une fois pris en main (autocomplétion, manipulations avancées de texte, ancres, macros clavier, etc.).

Seulement quand vous vous sentirez limité par nano vous devriez changer pour un éditeur plus puissant : Emacs, vim/nvim, Kate, kakoune, Helix, micro, VSCode, Notepad++ (si vous êtes coincé·e sur un Windows), TextMate (si vous êtes coincé·e sur un Mac), etc. Mais croyez moi, vous avez *largement* le temps avant d'en arriver là.

→ Dans votre terminal, lancez `nano`. Quels sont les différents éléments que vous voyez à l'écran ? Comment pouvez vous enregistrer un fichier ? Comment quitter l'éditeur ? Comment obtenir de l'aide ?

2. Écrivez quelques lignes (ce qui vous passe par la tête), puis enregistrez le fichier en l'appelant "tp0.txt".
   → Où se trouve le fichier à votre avis ?
3. Quittez nano.
   → Comment vérifiez que vous avez correctement répondu à la question précédente ?
4. Vous pouvez visualiser le contenu de votre fichier texte directement dans le terminal avec la commande `cat`.
   → Quelle ligne de commande permet d'afficher le contenu de votre fichier "tp0.txt" dans le terminal ?
5. Pour ouvrir un fichier existant dans nano, il suffit de lui passer en argument le chemin vers ce fichier.
   → Comment connaître les options que nano peut prendre en argument sur la ligne de commande ? Laquelle permet d'activer l'affichage des numéros de lignes ?
6. Comme tous les logiciels qui acceptent plein d'option de configuration en ligne de commande, nano dispose d'un fichier de configuration permettant d'activer des options par défaut pour ne pas avoir à les spécifier à chaque fois. Le nom de ce fichier est ".nanorc" et il doit être dans votre home (il est très courant que les fichiers de configuration soient des fichiers cachés dans le home avec le nom du logiciel et "rc" à la fin [1]). Ce fichier dispose d'une page de manuel "nanorc" en plus de celle de nano.
   → Configurez votre nano pour qu'il active par défaut au moins la numérotation des lignes. Éventuellement, prenez le temps de regarder les autres options et d'activer ce que vous souhaitez : indentation automatique, support de la souris, utilisation d'espaces plutôt que tabulations, largeur des tabulations à 4, et n'hésitez pas à personnaliser son interface selon vos goûts (couleurs etc.), vous allez passer du temps dans ce logiciel !

Exercice 3.

Travailler à distance avec SSH.

1. "SSH" signifie *secure shell*. C'est un protocole permettant de se connecter à distance sur une machine de manière sécurisée.
   On passe en argument à la commande `ssh` le nom d'utilisateur et l'hôte avec la syntaxe `login@host`.
   Depuis chez vous, vous pouvez vous connecter au Bocal via le serveur Belouga qui joue le rôle de passerelle avec la ligne de commande : `ssh login@bocal.up8.edu`.
   → Connectez-vous sur Belouga.
   (Comme pour fermer un shell, vous pouvez vous déconnecter avec la commande `exit`, ou avec `ctrl+d` quand l'invite de commande est vide.)
2. Sur Belouga, vous avez accès à vos fichiers, mais c'est tout : il n'y a presque aucun logiciel installé.
   Il vous faut donc vous connecter à un autre ordinateur qui n'a pas juste le rôle de passerelle sur le réseau. Par exemple, celui que vous avez devant vous pendant ce TP [2].
   En revanche, toutes les machines n'ont pas un nom d'hôte facile à retenir comme "bocal.up8.edu" ! Vous pouvez trouver l'adresse IP avec laquelle une machine est identifiée sur son réseau avec la commande `hostname -I`. Généralement, les adresses sur les réseaux locaux commencent par "192.168.".
   → Trouvez l'adresse IP de votre machine sur le réseau du Bocal, puis utilisez là pour vous y connecter en SSH depuis Belouga.
3. Bien sûr, ce n'est pas très pratique de devoir faire toutes ces étapes… mais c'est une situation tellement courante que SSH permet de faire tout ça d'un coup, comme on le verra plus tard [3].

1. "rc" vient de loin dans l'histoire, il s'agit à l'origine d'un programme créé par Louis Pouzin pour le système CTSS (un ancêtre d'UNIX) en 1963 : RUNCOM, comme "run commands".
2. On notera qu'il est donc important de ne pas éteindre ou débrancher les machines des salles informatiques : d'autres personnes ont peut-être du travail en cours dessus, y compris à distance !
3. Pour celleux que ça intéresse, il s'agit d'utiliser des clés SSH plutôt qu'une authentification par mot de passe, et ensuite d'utiliser l'option `-J` pour la passerelle et/ou une configuration automatique spécifique pour un hôte en particulier dans son fichier `~/.ssh/config`.

Exercice 4.

Faire persister sa session de travail avec tmux.

1. Pour l'instant, si vous êtes en plein travail depuis chez vous et que vous vous déplacez à l'université ou l'inverse, vous devez tout fermer, vous déconnecter, et vous reconnecter et tout rouvrir une fois arrivé à destination. C'est peu pratique !

   Le logiciel tmux vous permets de gérer des terminaux virtuels persistants, dont vous pouvez vous détacher et auxquels vous pouvez vous réattacher plus tard.

   Quand vous le lancez, le logiciel tmux prend donc la main sur votre terminal et se place comme une couche intermédiaire entre celui-ci et les programmes que vous utilisez (le shell, nano, etc.).

   → Lancez la commande `tmux`. Que voyez-vous sur votre écran de terminal ?

2. Comme tous les outils dans le terminal, tmux se contrôle essentiellement au clavier. Pour ne pas interférer avec les programmes qui s'exécuteront dans tmux, celui-ci utilise un *préfixe* unique pour l'ensemble de ses commandes : `ctrl+b`.

   Par exemple, vous pouvez vous détacher de votre session tmux avec `ctrl+b d` (*detach*), c'est-à-dire faire le raccourci clavier `ctrl+b`, puis relâcher ces touches et presser la touche `d`.

   → Dans votre session tmux, ouvrez votre fichier "tp0.txt" dans nano, puis déplacez votre curseur quelque part dans le fichier.

   Ensuite, sans quitter nano, détachez-vous de votre session tmux.

3. Pour vous réattacher à votre session tmux, vous pouvez lancer la commande `tmux attach-session` (que vous pouvez abréger en `tmux attach` ou même `tmux a`).

   Attention, si vous relancer simplement la commande `tmux`, vous allez ouvrir une *nouvelle session*.

   Vous pouvez lister les sessions existantes avec la commande `tmux list-sessions` (que vous pouvez abréger en `tmux ls`). Vous pouvez choisir de vous attacher à une session particulière avec l'option `-t` (*target*) de la commande `attach-session`. Par exemple la ligne de commande `tmux a -t 0` se rattachera à la session 0 même si il y en a d'autres en cours.

   → Rattachez vous à votre session tmux. Que constatez-vous ?

4. Si vous quittez l'ensemble des programmes qui tournent dans votre session tmux, typiquement votre shell (cf exo 1 question 1), celle-ci se fermera aussi automatiquement, n'ayant plus rien à exécuter.

   → Fermez complètement votre session tmux.

5. Au delà de la persistance, tmux dispose d'énormément de fonctionnalités pratiques, comme par exemple le multiplexage du terminal.
   - Vous pouvez créer une autre "fenêtre" (*window*) à l'intérieur de votre session avec `ctrl+b c` (*create*).
   - Pour passer d'une fenêtre à l'autre vous pouvez faire `ctrl+b n` (*next*) ou `ctrl+b p` (*previous*), ou directement avec `ctrl+b N` où *N* est le numéro de la fenêtre.
   - Vous pouvez naviguer visuellement parmi vos fenêtres avec `ctrl+b w` (*windows*).
   - Pour fermer une fenêtre c'est `ctrl+b &`. Quitter tous les programmes qu'une fenêtre exécute la ferme aussi.
   - Vous pouvez couper votre fenêtre horizontalement avec `ctrl+b %` et verticalement avec `ctrl+b "`.
   - Les sous-fenêtres s'appellent des "panneaux" (*pane*). Vous pouvez passer de l'un à l'autre avec `ctrl+b o` (*other*). Sinon `ctrl+b` puis flèche (pratique en cas de découpages multiples).
   - Vous pouvez naviguer visuellement parmi vos panneaux avec `ctrl+b q`. Taper le chiffre qui s'affiche saute dans le panneau correspondant.

   Au lieu de retenir tout ça par cœur, retenez simplement que `ctrl+b ?` liste les raccourcis disponibles. Vous apprendrez ceux qui vous sont effectivement utile au fur et à mesure par l'usage.

   → Relancez une session tmux et jouez un peu avec tout ça pour vous familiariser avec le fonctionnement.

6. Comme nano, tmux peut être configuré via un fichier de configuration. Le sien est `~/.tmux.conf`.

   Ce fichier est constitué d'une série de commandes tmux qui sont exécutées au lancement du logiciel. Les nombreuses commandes disponibles sont listées dans la page de manuel que vous pouvez consulter avec `man tmux`, mais n'hésitez pas également à vous renseigner sur le web.

   Pour activer le support de la souris il suffit de mettre la ligne `set -g mouse on` dans ce fichier.

   Pour ma part j'y ai par exemple mis les lignes suivantes qui permettent d'utiliser les touches `|` et `-` au lieu de `%` et `"` pour les découpages horizontaux et verticaux, ce que je trouve bien plus intuitif :

```
bind | split-window -h
bind - split-window -v
```

Happy hacking !