ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

Piscine J0 — TP

version **nº 1**

Assistants C/Unix 2014 <acu@epita.fr>

Copyright

Ce document est destiné à une utilisation au sein d'EPITA (site) uniquement.

Table des matières

*. `https://acu.epita.fr`

1 Avant-propos

L'objectif de ce TP est de vous aider à comprendre comment fonctionne le parc informatique du campus, de voir ce qu'il est possible d'y faire et ce qui ne doit pas y être fait, d'apprivoiser les commandes élémentaires dans un environnement et de configurer quelque peu votre compte.

De ce fait, les parties de cours sont particulièrement généreuses mais les manipulations obligatoires sont élémentaires et peu nombreuses. Il n'est pas possible de *tout* connaître par cœur, même avec beaucoup d'ancienneté. En revanche, il faut savoir comment trouver l'information et il y a un minimum vital que tout utilisateur se doit de maîtriser : c'est à cela que ce TP va vous initier.

Soyez curieux et posez-vous des questions : essayez de combiner les différentes notions abordées pour tenter de tirer un maximum de connaissances de cette session.

2 Premier contact avec le PIE

2.1 Généralités

2.1.1 Le BOCAL

Le BOCAL est l'entité responsable du bon fonctionnement de l'informatique sur l'ensemble du pôle. Si vous rencontrez un problème de dysfonctionnement sur l'une des machines ou un service de l'école, c'est le BOCAL qu'il faut contacter.

Pour contacter le BOCAL, privilégiez leur intranet : intra-bocal.epita.fr. La connexion s'effectue avec votre login et votre **mot de passe PPP**. Vous pouvez ensuite créer un **ticket** pour signaler le problème rencontré. Si (et seulement si) vous n'êtes pas en mesure de créer un ticket, il faut vous déplacer aux bureaux du BOCAL :

— au Kremlin-Bicêtre : prendre l'escalier à droite de la cafétéria puis la première à gauche et descendre l'escalier en colimaçon.
— à Villejuif : au sous-sol, mais les bocaliens n'y sont pas toujours présents.

Les machines du campus font partie d'un large réseau appelé **PIE** : *Parc Informatique des Écoles*. Pour connaître l'ensemble des services mis à votre disposition sur le PIE (et gérés par le BOCAL), nous vous invitons à consulter l'importante documentation disponible directement sur leur intranet.

Une autre responsabilité du BOCAL est de s'assurer que vous ne faites rien d'interdit en salle machines ou sur vos comptes personnels. En cas d'infraction, vous risquez un ***close compte***. Lorsque vous êtes en situation de *close compte*, vous ne pouvez plus accéder aux ressources de l'école, ce qui peut-être particulièrement gênant pour un rendu à effectuer. Pour vous sortir de cette situation, vous pouvez vous présenter au BOCAL ou, si vous préférez, utiliser directement l'intranet du BOCAL, à l'onglet **TIG**.

Les assistants n'ont pas le pouvoir de retirer les *close comptes* : inutile donc de venir nous voir dans ce cas.

2.1.2 Règlement et charte du PIE

L'utilisation des ressources du PIE est soumise à l'acceptation d'une charte et à un règlement. En vous inscrivant à l'école, vous avez choisi de les respecter.

La charte peut et **doit** être consultée : cf. l'intranet du BOCAL.

Le règlement est affiché sur toutes les portes d'entrée des salles machines. Pour rappel, en SM, il est interdit de :

— boire ou manger ;
— se balancer sur les chaises ;
— avoir un rack sans login inscrit sur la face avant ;
— relier un appareil au réseau électrique ou informatique sur des prises non prévues à cet effet ;
— déranger les autres dans leur travail ;
— abuser des ressources de l'école ;
— quitter son poste sans le laisser propre et rangé, chaise comprise ;
— jouer (jeux en Flash inclus) ;
— regarder des vidéos ;
— manquer de respect envers les personnes.

Ces règles s'appliquent quel que soit le terminal que vous utilisez (machine, laptop ou téléphone) et sont aussi généralisables aux salles de cours. D'autres interdictions ou règles de conduite vous seront rappelées durant la piscine. Pour l'instant, soyez bien sûrs d'assimiler les précédentes.

2.1.3 Chronos et liens utiles

Pour ouvrir un navigateur Web, par exemple Mozilla Firefox, tapez `firefox` dans votre shell et validez avec la touche Entrée.

Chronos Le site chronos.epita.fr héberge les emplois du temps. Ceux-ci peuvent être assez changeants à EPITA ; aussi, une consultation très régulière de Chronos est fortement encouragée.

Le laboratoire 3IE propose une application mobile pour consulter ces emplois du temps : EpiLife, et ils sont également disponibles au format ICS sur le portail de l'intranet des assistants.

Intranet d'EPITA Le site intranet.epita.fr est l'intranet général de l'école. C'est ici que figureront vos bulletins de notes. Il s'agit également du support de communication du service des relations avec les entreprises.

Intranet du service chargé de la communication Le site intracom.epita.fr est l'intranet du service en charge de la communication de l'école. Il permet de s'inscrire aux JPO et aux salons, de consulter ses points de communications (points ENAC) et de connaître les dates des conférences organisées sur le campus. Un calendrier annuel des actions de communication y est disponible.

Pour vous connecter sur cet intranet, utilisez votre mot de passe SOCKS.

Intranet des assistants Le site acu.epita.fr est l'intranet des assistants : ACU (nous) et YAKA (les suivants). Pour vous y connecter, utilisez le login et le mot de passe qui vous ont été délivrés par les assistants. Très probablement, ce login sera le même que celui attribué par le BOCAL. Le mot de passe sera quant à lui différent.

Sur cet intranet, vous trouverez les sujets de projet, les TP, vos traces, vos notes, etc. L'intranet Assistants est le support principal de communication entre les Assistants et les étudiants.

Dans la section *Documents*, vous trouvez des documents dont les assistants considèrent la lecture indispensable. Si vous ne les avez pas encore lus, faites-le pour demain.

Un document particulièrement important est présent dans cette section : **la nétiquette**. Elle présente les règles et conventions utilisées pour communiquer *via* Internet : courriels, news, etc. Les assistants mettent un point d'honneur à ce que les règles de la nétiquette soient respectées lors de tout échange. Si votre message ne s'y conforme pas, il risque de se retrouver sans réponse, voire rejeté.

2.2 Terminal et Shell

Afin de commencer à travailler, vous allez devoir ouvrir un terminal (rxvt, xterm, etc.), ou plus précisément un émulateur de terminal [1]. Il exécute un logiciel particulier appelé *shell* (sh, tcsh, zsh, bash, etc.) qui vous permet d'interagir avec le système d'exploitation. Son rôle est de lire les commandes que vous lui fournissez et de les exécuter.

Les commandes peuvent être très complexes, comme vous le verrez par la suite, mais également très simples. Par exemple, l'exécution d'un programme se fait par une commande comme suit : `programme [options] [données]`

Les informations supplémentaires que vous pouvez transmettre à un programme, comme les *options* ou les *données*, doivent toutes être séparées les unes des autres par des espaces. Les options sont facilement reconnaissables car commencent toujours par un tiret `-`.

Attention : certaines données peuvent également commencer par un tiret. Il peut-être alors utile d'indiquer au programme où se terminent les options et où commencent les données. Cela se fait en insérant dans la *ligne de commande* un double tiret `--`.

Enfin, lorsqu'un programme se termine, il transmet au terminal une *valeur de retour* qui indique la manière dont s'est terminé l'exécution : sortie normale, erreur d'utilisateur, problème critique, etc. La notion de *valeur de retour* sera approfondie ultérieurement.

Durant toute cette année, et particulièrement durant la piscine, vous allez apprendre à utiliser la ligne de commande et découvrir à quel point c'est un outil radicalement plus simple qu'une interface graphique. À partir de maintenant, toutes les commandes dont nous allons parler devront donc être entrées et exécutées dans le shell. Nous vous encourageons à vous documenter, pour apprendre à utiliser ses fonctionnalités avancée ; ainsi qu'à le personnaliser.

Attention : toute personnalisation effectuée sur votre rack ne doit pas consister à simplement recopier un exemple trouvé sur Internet, cela serait considéré comme de la triche. Vous devez être capable d'expliquer le comportement de chaque élément de personnalisation.

2.3 NetSoul

NetSoul est un **protocole d'identification réseau** propre au pôle IONIS. Dès que vous ouvrez une session graphique sur une machine du PIE, vous êtes automatiquement identifié auprès du serveur NetSoul : `ns-server`.

Lorsque vous vous connectez aux ressources de l'école à partir de votre machine personnelle, vous devez vous identifier auprès du serveur NetSoul. Si vous n'êtes pas identifié, l'accès à ces ressources peut vous être refusé automatiquement.

Une liste non exhaustive de clients NetSoul à installer chez vous ou sur votre compte est disponible sur le site du Bocal.

1. Un terminal est, historiquement, une machine périphérique, permettant de communiquer avec un ordinateur ; par exemple : VT100, IBM 3270, etc.

2.4 HOME

Sur votre rack, vous disposez d'un espace disque personnel. Lors de votre première connexion, un certain nombre de fichiers, propres à la configuration de votre environnement, sont déjà présents.

Sauf mention contraire, modifier ces fichiers est donc risqué car vous risquez de rendre votre environnement instable. Renseignez-vous toujours sur les effets de vos changements dans les configurations.

Si votre compte est devenu inutilisable, il est possible de le restaurer ; pour cela, rendez-vous au laboratoire des assistants.

2.5 Le navigateur

2.5.1 Gestion des mots de passe

L'utilisation d'Internet va en quelque sorte de pair avec l'utilisation de mots de passe. Aujourd'hui, un individu moyen possède environ 7 mots de passe. Mais les grands consommateurs d'informatique en possèdent généralement beaucoup plus !

Évidemment, il devient alors particulièrement tentant d'essayer d'obtenir un peu d'aide auprès des outils informatiques pour les retenir à notre place… et c'est là que la situation devient dangereuse, surtout si nous ne sommes pas sur notre ordinateur personnel.

Sur le PIE, des dizaines d'étudiants se trouvent en permanence dans la même salle machines que vous, et qui sait combien sur le pôle n'attendent que la première occasion pour vous faire du mal ? Vous l'aurez compris : sauvegarder ses mots de passe systématiquement dans votre navigateur ou tout autre logiciel est particulièrement dangereux dans un environnement partagé. Lorsque vous sauvegardez un mot de passe, dites-vous qu'à partir de maintenant il est potentiellement entre de mauvaises mains…

2.5.2 Page d'accueil

Un mini-portail local vous présente un certain nombre de liens utiles et vous permet de faire des recherches facilement.

Conservez cette page d'accueil durant toute la durée du semestre ACU.

3 Commandes élémentaires

3.1 Documentation et aide

3.1.1 man

man est probablement la commande que vous aurez le plus à utiliser cette année. Comme son nom l'indique, elle donne accès à une page de manuel.

Utiliser cette commande doit donc devenir un réflexe. Les assistants n'auront pas toujours le temps de vous aider. Pour vous simplifier la vie, et surtout vous éviter des remarques désagréables, utilisez toujours le man d'une commande avant de demander de l'aide sur cette dernière.

Les pages de manuel ne se limitent pas aux seules commandes que vous pouvez taper dans un shell. Vous pourrez également l'utiliser quand vous commencerez à coder en C et bien plus.

En effet, vous pouvez spécifier le domaine pour lequel vous chercher des informations pour une commande en utilisant *man* comme ceci : `man [number] command`, en prenant soin de remplacer `[number]` par un des nombres suivants :

1. Programmes exécutables ou commandes shell
2. Appels système (fonctions fournies par le noyau)
3. Appels de fonctions provenant de la bibliothèque C standard
4. Fichiers spéciaux (souvent situés dans `/dev`)
5. Formats de fichiers et conventions (par exemple `/etc/passwd`)
6. Jeux
7. Divers
8. Commandes d'administration système (souvent pour le root uniquement)
9. Routine du noyau [non standard]

Par exemple, `man chmod` est équivalent à `man 1 chmod` qui est différent de `man 2 chmod`. En effet, par défaut, man sélectionne la page de manuel de niveau minimal. Notez également que certaines pages de manuel peuvent exister dans plusieurs sections.

Dorénavant, à chaque nouvelle commande qui vous est présentée ou que vous rencontrez, consulter le man associé doit être un réflexe.

3.1.2 whatis, apropos, info

Dans votre quête de réponses et de documentation, il peut devenir nécessaire de seconder la commande *man* par d'autres :

— `whatis` affiche la description présente dans l'entête des pages de manuel associées à une commande.
— `apropos` cherche le nom et la description des pages de manuel qui contiennent le mot-clé sur lequel porte la recherche. Très pratique lorsque l'on cherche une commande dont on ne connaît pas le nom.
— `info` est une alternative à `man`. Certains logiciels proposent de la documentation uniquement accessible via `info`.

3.1.3 Exercice : `man`, `info` et `whatis`

1. Pour apprivoiser une commande, rien de mieux que de la manipuler. Utilisez donc successivement `man`, `info` et `whatis` pour décrouvrir en détail ce que font les commandes suivantes :
   — `info`
   — `whatis`
   — `apropos`
   — `man`
   — `pwd`
   — `ls`
   — `rm`
   — `cp`
   — `mv`
   — `cat`
   — `echo`
   — `touch`
2. Quelle est la différence entre `whatis`, `man` et `info` ? Ont-ils un point commun ?
3. À l'aide d'une option de `man`, affichez le chemin absolu du man de la commande `find`.
4. Recherchez le mot-clé `delete` dans l'ensemble des man à l'aide de l'option appropriée.

3.1.4 Exercice : `apropos`

Si vous ne connaissez pas le nom exact de la commande que vous cherchez, vous pouvez utiliser la commande `apropos`. Essayez-la avec les mots-clés suivants :

— `copy`
— `rename`
— `pdf`

3.2 Metalock

Dans l'école, il y a des personnes mal intentionnées. Laisser votre session ouverte et sans surveillance, c'est risquer qu'une de ces personnes vienne faire n'importe quoi avec votre compte :

— effacer ou modifier des fichiers, peut-être votre travail ;
— récupérer vos mots de passe ou votre travail ;
— faire quelque chose passible de close compte, voire interdit par la loi ;
— etc...

Pour vous protéger tout en vous évitant d'avoir à ouvrir et fermer une session constamment, il existe un programme : `metalock`. Pour maîtriser son fonctionnement, le plus simple est encore de l'essayer.

3.3 Arborescence et navigation

Sous UNIX, tout est fichier. Il s'agit de l'information élémentaire à comprendre lorsque l'on utilise un système d'exploitation de la famille des Unix, qu'il soit BSD ou Linux. Ainsi, un répertoire est un fichier, différent des fichiers ordinaires certes, mais reste un fichier.

3.3.1 ls

`ls` est une commande qui permet de lister des fichiers, répertoires compris. `ls` signifie *LiSt*. Plusieurs options sont disponibles :

— `-a` montre les fichiers cachés, c'est-à-dire ceux qui commencent par un point ;
— `-l` affiche des détails supplémentaires pour chaque fichier :
   — nature du fichier (répertoire, lien, ...),
   — droits,
   — propriétaire,

— groupe,
— taille,
— date de modification,
— nom.

D'autres options sont disponibles, à vous de lire le man.

3.3.2 Exercice : visualisation

1. Listez le contenu du répertoire `/etc`.
2. Affichez les propriétés du répertoire `/etc`.
3. Quels sont les droits des fichiers `/etc/passwd`, `/etc/services` ?
4. Affichez les fichiers cachés dans le répertoire racine de votre compte. Qu'est-ce qui fait qu'un fichier est un fichier caché ?

3.3.3 cd

Déplacements de base La commande `cd` permet de se déplacer dans l'arborescence, en passant d'un répertoire à un autre. `cd` signifie *Change Directory*. En arguments, les chemins relatifs ou absolus sont acceptés pour se déplacer.

Raccourcis Il existe plusieurs raccourcis utiles dans l'utilisation de `cd`.

— Sans argument, la commande vous ramène dans votre *home*.
— L'argument `~` (*tilde*) revient à ne pas utiliser d'arguments.
— En revanche, si `~` est suivi du nom d'un utilisateur, par exemple `~login_x`, la commande vous déplace dans le répertoire *home* de cet utilisateur, à condition d'avoir le droit d'y accéder.
— La destination `..` fait remonter d'un niveau dans l'arborescence.
— Enfin, `-` revient dans le répertoire visité juste avant le répertoire dans lequel vous êtes actuellement.

Navigation avancée Lors d'une navigation intense, le répertoire courant est constamment modifié pour diverses raisons. Il peut-être alors utile de remonter l'arborescence parcourue sans avoir à taper le chemin à chaque changement de répertoire.

Pour cela, il existe un mécanisme d'historique sous forme d'une pile :

— Pour ajouter un répertoire à la pile, il faut le donner en argument à la built-in `pushd`.
— `dirs`, éventuellement avec l'option `-v`, permet d'afficher la pile, et donc l'historique.
— Enfin, `popd` dépile le dernier chemin de la pile et l'utilise comme nouveau chemin courant.

3.3.4 pwd

La commande `pwd` permet de se situer dans l'arborescence. Elle affiche en effet le chemin complet jusqu'au *répertoire courant*, le répertoire dans lequel vous êtes actuellement. *pwd* signifie *Print Working Directory*.

3.3.5 Exercice : parcours

1. Placez-vous dans le répertoire `/var/log` et listez son contenu.
2. Allez dans le répertoire `~mysql`. Quel en est le chemin complet ?
3. Retournez à la racine de votre compte.
4. Placez-vous dans votre répertoire `private`, puis revenez en arrière.

Ce répertoire est un conteneur chiffré, déchiffré au moment de votre connexion. De base, il contient uniquement vos identifiants NetSoul.

3.4 Manipulation de fichiers et répertoires

Bien que tout soit fichier sous Unix, les opérations de manipulation sont parfois différents s'il s'agit d'un répertoire ou d'un fichier ordinaire. Néanmoins, certaines commandes restent strictement identiques quel que soit le contexte d'utilisation.

3.4.1 touch

La commande `touch` permet de modifier les dates d'un fichier, c'est-à-dire de mettre la date de dernière modification à l'instant présent. Par défaut, `touch` crée un fichier ordinaire vide si le fichier passé en argument est inexistant.

3.4.2 mkdir

La commande `mkdir` crée un répertoire. `mkdir` signifie *MaKe DIRectory*. Pour créer une hiérarchie de répertoires inexistants, l'utilisation de l'option `-p` est nécessaire.

3.4.3 mktemp

La commande `mktemp` crée un fichier temporaire (dont le nom est quelconque et qui n'existe pas encore) et affiche son nom une fois la création terminée. `mktemp` signifie *MaKe TEMPorary file*.

Il est possible de spécifier à `mktemp` une forme à utiliser pour le nom du fichier, en donnant en argument un nom contenant un ou plusieurs `X` qui seront remplacés par des valeurs uniques lors de la création.

3.4.4 cp

La commande `cp` copie des fichiers. `cp` signifie *CoPy*. Quelques options utiles :

— `-R` copie récursivement les sous-répertoires.
— `-f` force la copie.
— `-i` prévient si la cible existe déjà pour ne pas l'écraser.

3.4.5 mv

La commande `mv` déplace un ou plusieurs fichiers vers une nouvelle destination. `mv` signifie *MoVe*. Quelques options utiles :

— `-f` force la copie.
— `-i` prévient si la cible existe déjà pour ne pas l'écraser.

3.4.6 rmdir

La commande `rmdir` supprime un ou plusieurs répertoires si et seulement si ceux-ci sont vides. `rmdir` signifie *ReMove DIRectory*. L'option `-p` permet la suppression d'une hiérarchie de répertoires.

3.4.7 rm

La commande `rm` permet de supprimer des fichiers qui, par défaut, ne sont pas des répertoires. `rm` signifie *ReMove*. Quelques options utiles :

— `-r` supprime récursivement (supprime les répertoires non vides).
— `-f` force la suppression.
— `-i` prévient avant de supprimer.

Attention : il n'est pas possible de récupérer des fichiers effacés par la commande rm. Toute suppression est donc **définitive**.

3.4.8 Exercice : création

1. À la racine de votre compte, créez un répertoire `newfish` et un répertoire `aculover`.
2. Dans ce répertoire `newfish`, créez l'arborescence suivante, en une seule commande : `~/newfish/nage/dans/la/piscine/`.
3. Créez un fichier `nemo` et un fichier `.Pterois` à la racine de votre compte.

3.4.9 Exercice : copie

1. Copiez le fichier `.Pterois` dans le répertoire `aculover`.
2. Copiez le répertoire `aculover` de votre compte dans le répertoire `newfish`.

3.4.10 Exercice : déplacement

Déplacez le fichier `.Pterois` se trouvant toujours à la racine de votre compte vers le répertoire `aculover` qui se trouve toujours également à la racine de votre compte, en demandant l'autorisation avant d'écraser.

3.4.11 Exercice : suppression

Attention : sous Unix il n'y a pas de corbeille, toute suppression est définitive !

1. Déplacez vous dans le dossier `newfish` et supprimez récursivement tout ce qui s'y trouve.
2. Revenez à la racine de votre compte, et supprimez le dossier `aculover`, sans utiliser la commande `rm`.

3.5 Affichage et comparaisons de fichiers

3.5.1 Affichage

echo La commande `echo` est probablement la built-in shell la plus simple. Elle est cependant très utile. `echo` prend en argument *n* arguments et les affiche à l'écran. L'option `-n` fait que `echo` ne revient pas à la ligne après avoir affiché tous ses arguments.

cat La commande `cat` permet de voir le contenu d'un ou plusieurs fichiers ou flux. `cat` signifie *conCATenate*. S'il y a un ou plusieurs fichiers en arguments, `cat` affiche le contenu de ceux-ci, les uns à la suite des autres. Sinon, il lit sur l'*entrée standard* : le clavier.

Quelques options utiles :

— `-n` affiche le numéro de chaque ligne en face de celle ci.
— `-e` affiche en fin de chaque ligne un symbole `$`.

less et more Les commande `less` et `more` sont deux pagineurs : ils permettent de faire défiler agréablement la *sortie standard*, ou affichage. Si on ne leur fournit pas de fichiers en argument, tout comme `cat`, ils lisent sur l'entrée standard.

`less` vous oblige à quitter explicitement la lecture en pressant la touche `q` puis nettoie l'affichage, tandis que `more` termine lorsque le bas du fichier est atteint et le laisse affiché dans le terminal. Avec `more`, il n'est cependant pas possible de remonter dans le document si on le fait défiler.

3.5.2 Comparaison de fichiers

Lorsque l'on travaille sur un fichier, il est important d'effectuer régulièrement des sauvegardes. On peut arriver à un point où des modifications que l'on a effectué rendent un programme inutilisable. Il est alors très intéressant d'avoir la possibilité de voir ce qu'on a bien pu changer dans notre fichier, en mettant en évidence les différences par rapport à la version sauvegardée.

diff La commande `diff` permet de comparer deux fichiers passés en argument, en affichant toutes les différences entre eux. L'option `-u` affiche un résultat dit *unifié*, plus standard et agréable à regarder.

cmp La commande `cmp` permet également de comparer deux fichiers passés en argument, mais elle n'affiche pas les différences, seulement si les fichiers sont identiques ou non.

3.5.3 file

La commande `file` détermine le type d'un ou plusieurs fichiers passés en argument, en leur appliquant un certain nombre de tests. `file` peut également prendre en argument, avec l'option `-f`, une liste de chemins contenue dans un fichier texte.

3.5.4 Exercice : affichage et comparaison

1. Quel est le type des fichiers `/etc/motd`, `/bin`, `/dev/zero` ?
2. Affichez votre fichier `.xsession` avec la commande `cat`, en affichant les numéros de ligne et les retours à la ligne.
3. Affichez le fichier `TODO` page par page, puis recherchez votre login.
4. Quelles sont les différences entre `most` et `less` ?
5. Comparez les fichiers `avant.txt` et `apres.txt` disponibles comme fichiers annexes sur l'intranet des assistants, et mettez en avant leurs *différences*.
6. Quelle est la différence entre `cmp` et `diff` ?
7. Que fait l'option `-u` de `diff` ?

3.6 tar

3.6.1 La commande tar

La commande `tar` permet de créer une archive, y ajouter des fichiers et en extraire, au format *tar*. Voici les options les plus fréquemment utilisées :

— `-c` crée une nouvelle archive, ou écrase une archive existante, y ajoutant les fichiers spécifiés.
— `-r` ajoute les fichiers spécifiés à une archive existante.
— `-t` liste le contenu de l'archive.
— `-x` extrait les fichiers de l'archive.
— `-f archive` emplacement de l'archive.
— `-j` utilise la compression bzip2.
— `-z` utilise la compression gzip.
— `-p` préserve les permissions des fichiers ajoutés à l'archive.
— `-v` mode verbeux.

Quelques exemples d'utilisation :

— `tar cvzf archive.tgz fichier1 dossier1 ...` : création d'une archive `archive.tgz` compressée au format gzip, avec le mode verbeux.

— `tar cvjf archive.tgz fichier1 dossier1 ...` : création d'une archive `archive.tgz` compressée au format bzip2, avec le mode verbeux.
— `tar xvzf archive.tgz` : extraction d'une archive `archive.tgz` compressée au format gzip, avec le mode verbeux.
— `tar xvjf archive.tar.bz2` : extraction d'une archive `archive.tar.bz2` compressée au format bzip2, avec le mode verbeux.

3.6.2 Exercice : archivage

1. Faites une archive compressée au format gzip du répertoire `aculover` se trouvant à la racine de votre compte que vous appellerez `aculover.tgz`.
2. Affichez le contenu de votre archive sans pour autant la décompresser.
3. Déplacez votre archive compressée dans `/tmp`, et décompressez-y-la.

3.7 alias

3.7.1 La commande alias

Beaucoup de commandes prennent en argument des options très pratiques que l'on souhaiterait utiliser directement. La built-in `alias` permet de créer des raccourcis, appelés couramment `alias`.

Sans argument, `alias` affiche la liste des alias définis dans le shell courant.

Enfin, pour supprimer un alias il faut utiliser la built-in `unalias`.

Quelques exemples d'utilisation :

— `alias ll='ls -l'` : création d'un alias `ll` dont l'exécution produit le même résultat qu'un appel à la commande `ls -l` ;
— `unalias ll` : suppression de l'alias `ll` précédemment créé.

3.7.2 Exercice : vos premiers alias

Renseignez-vous sur ce que fait l'option `-i` de `cp`, `mv` et `rm`, et créez les alias `cp`, `rm`, et `mv` appropriés.

4 Entrées/sorties

4.1 Flux, entrées/sorties standards

Pour communiquer avec l'utilisateur, les exécutables acceptent ou produisent des flux, respectivement en entrée ou en sortie. Il existe des entrées et sorties dites *entrées/sorties standards*, puisque définies pour tous les programmes :

— entrée standard/**stdin** : c'est le flux par lequel arrivent les informations. Par défaut, il s'agit du clavier.
— sortie standard/**stdout** : c'est le flux par lequel sortent les informations lorsque tout va bien et si le programme a besoin d'écrire quelque chose. Par défaut, il s'agit du shell dans lequel s'exécute le programme.
— sortie d'erreur/**stderr** : c'est le flux par lequel un programme bien fait va signaler des erreurs. Par défaut, c'est, comme pour stdout, le shell dans lequel s'exécute le programme.

4.2 Les redirections

Note : Les différents shells ne gèrent pas tous exactement de la même manière les redirections. Les présentations effectuées ici sont assurées de fonctionner sous Zsh.

4.2.1 Notions fondamentales

Les redirections permettent de rediriger un flux sortant vers un fichier. Il en existe deux types :

— la redirection simple, opérateur `>`.
— la redirection double, opérateur `>>`.

La redirection double concatène le flux redirigé au contenu du fichier, tandis que la redirection simple en écrase le contenu. Dans les deux cas, si le fichier de destination n'existe pas, il est créé. Les exemples suivants sont équivalents :

```
42sh$ echo 42 > /tmp/test
42sh$ > /tmp/test echo 42
```

Par défaut, les redirections travaillent sur la sortie standard d'un programme. Pour spécifier un autre flux, on préfixe la redirection par le numéro qui identifie ce dernier. Par exemple, pour la sortie d'erreur, nous pouvons avoir 2> ou 2>> selon la redirection souhaitée.

4.2.2 Agrégation

Il est possible de faire de l'agrégation de flux, c'est à dire rediriger un flux vers un autre flux. Ceci s'effectue avec l'opérateur `>&`.

Par exemple pour rediriger la sortie d'erreur vers la sortie standard, on écrira `2>&1`. Mise en situation :

```
42sh$ cat fichier_qui_n_existe_pas /etc/services > /tmp/test 2>&1
```

Dans l'exemple précédent, la sortie d'erreur est redirigée vers la sortie standard, qui est elle même est redirigée dans le fichier `/tmp/test`.

4.2.3 `/dev/null`

Le fichier `/dev/null` est un fichier un peu spécial : tout flux redirigé vers ce fichier est absorbé, un peu à la manière d'un trou noir. Il n'est pas possible de lire le contenu envoyé de dans ce fichier, qui apparaît toujours comme étant vide.

Il est intéressant d'utiliser `/dev/null` lorsque l'on filtre la sortie d'un programme pour absorber les flux jugés inintéressants.

4.3 Les indirections

Les indirections permettent de passer des données à un flux d'entrée. Comme pour les redirections, il existe les indirections simples ou doubles.

4.3.1 Indirection simple

L'indirection simple, d'opérateur `<`, permet de passer le contenu d'un fichier à un flux d'entrée. Ainsi, les commandes suivantes sont équivalentes :

```
42sh$ cat < /etc/services
42sh$ < /etc/services cat
```

4.3.2 Here-documents

L'indirection double, d'opérateur `<<`, est appelée *here-document*. Tout comme l'indirection simple, elle permet de passer des données au flux d'entrée d'un programme mais, cette fois-ci, au lieu de lire le contenu à partir d'un fichier, l'indirection double va lire à partir de l'entrée standard directement. Il est donc nécessaire de spécifier juste après l'opérateur de double indirection la séquence qui indiquera la fin de l'entrée.

Exemple d'utilisation :

```
42sh$ cat << EOF
Pour chaque étudiant,
la piscine est un moment uniqueEOF
et inoubliable
EOF
Pour chaque étudiant,
la piscine est un moment uniqueEOF
et inoubliable
```

4.4 Les tubes

Un tube de communication, couramment appelé ***pipe***, permet de rediriger la sortie d'une commande vers l'entrée d'une autre commande. L'idée est la même que pour les redirections et les indirections, sauf que celles-ci ne fonctionnent qu'avec des fichiers. Pour faire communiquer deux commandes directement, on utilise donc un *pipe*, d'opérateur `|`.

Ainsi, la séquence en deux lignes :

```
42sh$ commande1 > /tmp/sortie_commande1
42sh$ commande2 < /tmp/sortie_commande1
```

est équivalente à la séquence en une seule ligne :

```
42sh$ commande1 | commande2
```

Mieux, il est tout à fait possible de relier plusieurs commandes avec des pipes :

```
42sh$ commande1 | commande2 | commande3
```

4.5 Exercice

1. Copiez votre fichier `.zshrc` vers le fichier `zshrc-save` sans utiliser `cp`, et en utilisant seulement les redirections.
2. Créez l'alias `z` sur la commande `metalock` à la fin du fichier `.zshrc`, sans ouvrir le fichier, à l'aide des redirections. Attention à ne pas écraser l'ancien fichier !
3. Utilisez uniquement la commande `wc` et une redirection pour compter le nombre de lignes dans le fichier `/etc/passwd`.
4. Faites la même chose que précédemment avec un `|` (*pipe*), la commande `cat` et la commande `wc`.
5. Remplissez votre `.idoles` avec au moins 2 lignes, en utilisant uniquement `cat` et les redirections, le tout en une seule commande.

5 Les droits

5.1 Concept

La notion de droit est intimement liée aux fichiers : les droits définissent les individus et les actions que ces individus peuvent réaliser sur un fichier.

5.1.1 Permissions

Il existe 3 types d'opération possible sur un fichier, et chaque type est associé à une valeur numérique :

— lire, symbole '`r`' comme *Read*, valeur *4*.
— écrire, symbole '`w`' comme *Write*, valeur *2*.
— exécuter, symbole '`x`' comme *eXecute*, valeur *1*.

Ces valeurs, vous l'aurez remarqué, correspondent à une succession de puissances de 2 :

```
valeur  : 4 2 1
symbole : r w x
```

Bien évidement, il est possible d'autoriser plusieurs opérations sur un même fichier. On combine pour cela plusieurs permissions et on obtient ainsi une nouvelle valeur. Voici donc l'ensemble des combinaisons possibles et leur équivalence numérique :

— 0 `---` : aucun droit
— 1 `--x` : droit d'exécution (droit minumum pour pouvoir exécuter un binaire)
— 2 `-w-` : droit d'écriture
— 3 `-wx` : droit d'écriture et d'exécution
— 4 `r--` : droit de lecture
— 5 `r-x` : droit de lecture et d'exécution
— 6 `rw-` : droit de lecture et d'écriture (droits usuels pour un fichier)
— 7 `rwx` : tous les droits (droits usuels pour un script shell)

5.1.2 Groupes

Les droits UNIX ne se limitent pas à se donner des permissions à soi-même, ce serait bien dommage. Pour un même fichier, il est possible de définir trois combinaisons de permissions bien distinctes :

— les permissions pour l'utilisateur, c'est à dire soi-même.
— les permissions pour le groupe, chaque utilisateur pouvant appartenir à plusieurs groupes, il s'agit du groupe auquel a été légué le fichier.
— les permissions pour *les autres*, en sommes : tous les utilisateurs.

Lorsque l'on représente les droits pour un fichier, on représente donc les permissions accordées à chacun des groupes présenté ci-dessus, dans l'ordre *utilisateur groupe autres*.

Nous avons vu qu'il est possible de voir les droits associés à un fichier avec la commande `ls -l`. Regardons de plus près un exemple :

```
-rwxr-x--x 1 acu prof  141 Jul 20 17:51 AUTHORS
```

Ici, l'utilisateur *acu* à tous les droits, le groupe *prof* peut lire et exécuter, les autres peuvent seulement exécuter : `rwxr-x--x`. La *notation octale* correspondante, vous l'avez deviné, est : `751`.

Attention : sur le PIE, donner les droits en écriture sur ses fichiers à tout autre utilisateur que vous est passible de close compte.

5.2 chmod

La commande *chmod* permet de modifier les droits d'accès sur un fichier ou un répertoire. *chmod* signifie *CHange MODe*. Il existe trois manière pour donner des droits à un fichier.

5.2.1 Approche octale

Comme vous avez pu le voir ci-dessus, il est possible d'indiquer les permissions d'un fichier par l'utilisation d'une combinaison de 3 chiffres : on appelle cela le mode *octal*, où chaque chiffre correspond à une combinaison de permissions pour un groupe donné.

chmod comprend parfaitement cette notation pour modifier les permissions d'un fichier. Il suffit d'indiquer à chmod le mode souhaité ainsi que le fichier sur lequel on souhaite appliquer les modifications pour que celles-ci deviennent effectives.

Par exemple, pour donner les permissions `640` à un fichier, on appelera chmod de la manière suivante :

```
chmod 640 fichier
```

5.2.2 Approche symbolique

chmod accepte également une notation plus *symbolique* lors de l'attribution de permissions à un fichier. Ces symboles sont divisés en deux groupes :

1. Les symboles de permissions :
   - '`r`', comme *Read*, permission de lecture
   - '`w`', comme *Write*, permission d'écriture.
   - '`x`', comme *eXecute*, permission d'exécution.
2. Les symboles de groupe :
   - '`u`', comme *User*, permissions de l'utilisateur.
   - '`g`', comme *Group*, permissions du groupe.
   - '`o`', comme *Others*, permissions du groupe.

Pour donner ou enlever des permissions à un groupe, ou utilise deux opérateurs :

- '`+`', pour *donner* des permissions.
- '`-`', pour *enlever* des permissions.

Pour affecter plusieurs groupes en même temps en utilisant la notation symbolique, il est possible de combiner plusieurs séquences en les séparant par des virgules. Par exemple, donner les permissions `640` revient à écrire :

```
chmod u+rw,u-x,g+r,g-wx,o-rwx fichier
```

5.2.3 Approche par référence

Il existe une dernière manière d'appliquer les droits à un fichier : par référence. L'idée est de demander à chmod d'appliquer, pour un fichier A, les droits que possède un fichier B : la référence. Cela s'effectue comme suit :

```
chmod --reference=fichier_B fichier_A
```

6 Job Control

6.1 Exécuter des tâches

Lorsque l'on exécute une commande qui va durer, il est intéressant de pouvoir conserver son shell pour l'utiliser pour autre chose, sans avoir à attendre la fin de la commande pour le récupérer. Pour cela, on peut exécuter une commande en *tâche de fond*, en suffixant la commande par l'opérateur `&`. Le `&` indique au shell que la commande doit être exécutée en *arrière plan* pour ne pas le monopoliser.

Il est possible de lancer une tâche en avant plan et d'avoir soudain besoin d'utiliser son shell. C'est à ce moment que l'on utilise le *job control*. Le job control permet de reprendre la main sur le shell et de s'occuper des tâches qui ont été lancées à partir de ce même shell.

6.2 Interrompre une tâche

Lorsque l'on presse la combinaison de touches `CTRL-z` dans le shell, on suspend la tâche lancée en avant plan : la tâche devient bloquée, on ne peut temporairement plus l'utiliser. On a alors de nouveau l'usage du shell. La built-in `jobs` permet de connaître l'état de toutes les commandes lancées depuis ce même shell.

```
42sh$ jobs
[1]-  Stopped                 emacs
[2]+  Stopped                 vim toto.c
[3]   Running                 firefox &
```

Dans l'exemple ci-dessus, les commandes 1 et 2 ont été suspendues (stoppées) alors que la commande 3 est exécuté en tâche de fond.

6.3 Changer de plan

Il existe 2 built-ins qui permettent de modifier la mise en avant ou en arrière plan des commandes :

— `fg` met la commande en avant plan.
— `bg` met la commande en arrière plan, sans arrêter l'exécution.

Ces deux commandes travaillent par défaut sur la dernière commande suspendue. Pour modifier la mise en avant ou arrière plan d'une autre commande, il est nécessaire d'indiquer son numéro préfixé d'un `%` en argument d'une de ces deux built-ins.

Par exemple `fg %2` met la commande 2 en avant plan.

6.4 Exercice

1. Lancez les commandes `htop`, `emacs` et `cat` en les mettant à chaque fois, une fois l'exécution en cours, en arrière plan.
2. Affichez la liste de tous les jobs lancés.
3. Faites revenir emacs au premier plan.
4. Lancez vim en tâche de fond.

7 Éditeurs

La communauté des programmeurs est principalement divisée entre deux éditeurs de texte lorsqu'il s'agit de développement :

— Emacs ;
— Vim.

Ce sont les deux éditeurs de texte les plus connus dans le monde des logiciels libres.

7.1 Vim

Vim fonctionne sur le principe de cadres et de buffers (tampons) : un buffer contient du texte. Ce système permet de changer le buffer affiché à l'écran, de séparer la fenêtre en cadres pour qu'elle affiche plusieurs buffers en même temps, d'afficher des buffers sur une autre fenêtre, etc.

Il y a deux modes : le mode édition pour saisir le contenu et le mode commande pour effectuer des actions.

Voici une sélection de commandes les plus utilisées :

— `:q` : quitter
— `:q!` : quitter absolument (sans sauvegarder les modifications)
— `:w` : sauvegarder
— `:w fichier` : sauvegarder dans le fichier *fichier*
— `:e fichier` : éditer le fichier *fichier*
— `h j k l` : déplacer le curseur (vous pouvez aussi utiliser les touches directionnelles)
— `^` : aller en début de ligne
— `$` : aller en fin de ligne
— `J` : concaténer la ligne suivante à la fin de la ligne courante
— `Y` : copier la ligne courante
— `P` : coller la ligne courante
— `i` : insérer (passer en mode édition)
— `a` : écrire après le curseur (mode édition)
— `o` : écrire sur une nouvelle ligne (mode édition)
— `R` : remplacer le texte (mode édition)
— `r` : écrire une lettre

En mode édition, vous pouvez passer en mode commande en utilisant la touche `ESC`.

Pour utiliser Vim, tapez `vim` dans votre shell.

7.2 Emacs

Emacs fonctionne sur le même principe de buffers.

Dans Emacs, il est possible d'effectuer n'importe quelle action grâce à des raccourcis clavier :

— 'C-' représente la touche CTRL pressée simultanément avec la lettre qui suit.
— 'M-' représente la touche META pressé simultanément avec la lettre qui suit. META correspond à la touche ESC, mais une autre touche peut lui etre associee comme la touche ALT.

Voici une sélection des raccourcis les plus utilisés :

— C-x C-c : quitter
— C-x C-s : sauvegarder
— C-x C-f : ouvrir un fichier
— C-x k : ferme le buffer actuel
— C-x b : change de buffer
— C-a : aller en début de ligne
— C-e : aller en fin de ligne
— C-s : rechercher
— M-% : remplacer
— C-space : débute une région
— C-w : couper une région
— C-k : couper de la position du curseur à la fin de la ligne
— M-w : copier une région
— C-y : coller une région coupée
— C-_ : annuler la dernière action
— M-x help : aide
— C-x 3 : séparer le cadre courant verticalement
— C-x 2 : séparer le cadre courant horizontalement
— C-x 0 : fermer le cadre courante
— C-x 5 2 : ouvrir une nouvelle fenêtre

Pour utiliser Emacs, tapez emacs dans votre shell.

7.3 Prise en main

Pour obtenir de l'aide sur l'exercice qui suit, reportez-vous encore une fois aux cours disponibles sur l'intranet. Un certain nombre de commandes pour Vim et Emacs y sont détaillées.

1. Ouvrez Emacs et faites les choses suivantes :
   — Découpez votre fenêtre en 2 verticalement.
   — Dans la partie gauche, créez un fichier à la racine de votre compte, dans lequel vous allez écrire 3 lignes au choix.
   — Dans la partie droite, ouvrez votre fichier et remplissez-le à votre convenance, puis sauvegardez.
   — Revenez dans le buffer de gauche, et copiez/collez 3 fois ce que vous avez écrit précédemment, puis sauvegardez.
   — Quittez Emacs.
2. Lancez Vim, puis :
   — Découpez votre fenêtre en 2 verticalement.
   — Découpez votre fenêtre de gauche en 2 horizontalement.
   — Ouvrez le fichier créé précédemment.
   — Effacez les 3 dernières lignes du fichier.
   — Copiez la première ligne 10 fois à la fin du fichier.
   — Sauvegardez et quittez.

8 Git

8.1 Qu'est ce que Git

Git est un VCS (*Version Constrol System*), ou encore un outil de gestion de versions. En d'autres termes, c'est un logiciel vous aidant à conserver un historique en maintenant l'ensemble des versions ou révisions de votre travail. Cette gestion de révisions, pour ceux qui n'utilisent pas de logiciel comme Git (Mercurial, SVN, CVS, ...) le font sûrement déjà sans le savoir « à la main ». En effet, lorsque vous faites un copier-coller « à la barbare » de votre répertoire de travail pour garder une version du logiciel

qui fonctionne « au cas où je casse tout », vous faites en quelque sorte une gestion manuelle, fastidieuse et peu pratique des versions de votre travail.

Après une rapide introduction à un outil tel que Git, vous serez capable de faire des « sauvegardes » de votre travail à un instant *t* autant de fois que vous le désirez. Les sauvegardes seront rapides, consommeront peu d'espace disque par rapport à un bête copier coller et vous permettront de garder une trace chronologique de ces différentes sauvegardes. Enfin, cela vous permettra de facilement revenir sur l'une de ces révisions pour un, plusieurs, ou tous les fichiers de votre projet.

8.2 Pourquoi Git

Pour ceux ayant déjà touché à un outil de ce type, une question vient à l'esprit ``Mais pourquoi Git'' ? Avant tout, même si le troll est une activité des plus divertissantes, il n'y a pas de VCS parfait, chacun a ses défauts et ses avantages. Voici ce qui a motivé notre choix pour Git :

- C'est un VCS dit ``distribué'', c'est-à-dire qu'il peut fonctionner en local sur votre machine sans aucune intéraction avec un serveur faisant office de *dépôt maître*, ce qui n'est pas le cas pour SVN par exemple
- Il est l'outil utilisé pour certains grands projets que vous connaissez sûrement tels que le Kernel Linux (Il fut créé pour l'occasion) ou Android.
- Développé en C et Perl.
- Bien que cet outil soit avant tout développé pour Linux, il fonctionne aussi bien sous Windows que MacOSX. Il pourra donc vous servir pour n'importe lequel de vos projets au cours de votre cursus.

Seul Git vous sera présenté aujourd'hui ; vous êtes toutefois libre de vous documenter sur d'autres outils tels que Mercurial ou SVN afin de vous faire un avis sur les différents outils existants. Notez cependant que pour la gestion de vos projets et de vos rendus, seul Git sera disponible.

8.3 Un peu de vocabulaire

Les VCS ont leur vocabulaire bien à eux, nous allons nous arrêter sur quelques termes à connaître pour comprendre la suite des explications :

- **Un dépôt** : Nom donné au répertoire de votre projet géré par Git, seuls les fichiers présents dans ce répertoire (et sous répertoires bien sûr) peuvent être surveillés par le dépôt Git de votre projet.
- **Un commit** : Faire un commit, ou ``commiter'', est l'action de créer une sauvegarde ``locale'' de votre projet.
- **Un push** : Faire un *push*, ou ``pusher'', est l'action de propager vos sauvegardes locales (vos commits) sur un dépôt distant.
- **Une révision** : Nom que l'on donne à une sauvegarde de votre projet à un instant *t*. À chaque commit que vous réalisez, vous créez une nouvelle révision, ou version, de votre projet.
- **Le changeset (ou SHA-1)** : Dans un VCS classique tel que CVS ou SVN, tout le groupe de travail est en permanence sur la même révision du projet, nous ne l'expliquerons pas en détail aujourd'hui, mais le fait que vous puissiez travailler sans connexion à un serveur central dans Git compromet cet état de fait. Ainsi, chaque révision que vous faites n'est plus numérotée dans l'ordre chronologique de leur création, mais se voit octroyer un identifiant unique du type *5500bd8e0ae52775c9abf69749cde9c34001650b*. Par soucis de simplicité, Git vous autorise à n'utiliser que les premiers caractères de celui-ci (minimum 4), tant qu'ils ne produisent pas d'ambiguité sur le commit désigné.
- **le HEAD** : le HEAD est le nom donné à la dernière révision de votre projet, il correspond donc à votre dernier commit.
- **la working copy** : C'est l'état courant des fichiers de votre disque dur. Il diffère régulièrement du HEAD.

Encore un peu chamboulé par tous ces nouveaux termes ? Ne vous en faites pas, ça va venir avec la pratique.

8.4 Les commandes Git

8.4.1 git help

Toutes les commandes Git que vous taperez commencerons par le nom du binaire : **git**.

La commande que vous utiliserez le plus souvent au début est `git help`, elle vous donne une liste non-exhaustive des commandes disponibles sur git. Pour plus d'informations sur une commande, tapez `git help <nomdelacommande>`. Rassurez-vous, nous ne verrons pas toutes les commandes disponibles aujourd'hui mais uniquement celles qui seront indispensables à la gestion de base de vos projets et vos rendus.

8.4.2 git init

Afin que Git puisse surveiller votre projet, il faut initialiser le dépôt. Pour cela, entrez dans le répertoire où est contenu votre projet et tapez `git init`.

Cette commande va créer un nouveau répertoire dans votre dépôt : `.git`, **ne supprimez jamais ce répertoire de votre dépôt**, c'est ici que Git fait toutes ces manipulations magiques qui lui permettent de fonctionner. Dans le cadre de vos projets avec le laboratoire assistants, nous nous chargerons de cette étape, nous verrons plus loin comment procéder pour récupérer votre dépôt.

8.4.3 git add

De base, Git ne surveille aucun fichier dans votre dépôt, il faut lui demander de surveiller explicitement les fichiers qui sont importants pour le prochain commit. Pour cela, on utilise la commande *git add monfichier*. Quelques choses à retenir :

- Quand vous faites un `git add` sur un répertoire, le répertoire et **l'intégralité de son contenu** seront ajoutés à la liste des fichiers qui seront committés.
- N'ajoutez pas au dépôts les fichiers intermédiaires de compilation (les .o, .aux etc...), les binaires résultants de votre compilation, ou encore les fichiers temporaires (`monfichier~`, `#monfichier#`. Nous verrons plus tard comment faire pour que Git les ignore à tous les coups. En d'autres termes n'ajoutez que les fichiers utiles à votre projet, ne pouvant pas être générés.
- Git **ne peut pas surveiller un répertoire vide**
- Avant chaque commit, vous devez faire un `git add` de chaque fichier que vous souhaitez voir mis à jour dans votre dépôt. nous verrons plus loin comment faire pour demander à git de tout mettre à jour rapidement.

8.4.4 git status

Cette commande permet de savoir où en est la surveillance de vos fichiers pour le prochain commit. Le résultat de cette commande est une liste de fichiers, répartie en plusieurs catégories, la première contenant la liste des fichiers qui feront partie du prochain commit. La seconde (optionnelle) les fichiers modifiés qui ne feront pas partie du prochain commit et enfin les fichiers non suivis par Git.

8.4.5 git rm

Comme vous vous en doutez, il permet de supprimer un fichier de votre dépôt. En d'autres termes, cela fait un `rm` du fichier et Git arrêtera de surveiller ses modifications. Notez que vous ne pouvez pas supprimer un fichier que vous avez marqué comme devant être committé.

Rappelez-vous que le rôle de Git est justement de gérer les versions de votre projet, ainsi le fichier n'est pas perdu à jamais, si vous revenez à un commit antérieur à sa suppression, le fichier réapparaitra dans l'état qu'il avait lors de ce commit.

Vous avez également la possibilité de le supprimer uniquement de Git et de le garder sur votre système de fichier à l'aide de l'option `--cached`

8.4.6 git mv

Cette commande vous permet de faire un déplacement propre de vos fichiers au sein de votre dépôt. En effet, si vous passez par la commande Linux standard `mv`, Git considèrera que le fichier a été supprimé et vous devrez ajouter le nouveau fichier par la suite. De plus, cela rendra moins aisé le suivi de vos fichiers.

8.4.7 git log

Cette commande permet de lister l'historique de votre dépôt sous forme de la liste des commits réalisés, avec leur auteur, leur date et la description du commit. Cette commande est très utile pour savoir où vous en êtes ou tout simplement pour retrouver une ancienne version de votre dépôt que vous désirez restaurer.

À la lecture du sujet du soir, vous constaterez la présence obligatoire d'un fichier **`ChangeLog`** à la racine de votre rendu, contenant comme son nom l'indique le log (« journal ») des changements au cours de la réalisation de votre projet. Si vous avez bien suivi la partie sur les entrées/sorties, vous savez donc déjà que l'utilisation de Git vous permettra de générer ce fichier sans aucun effort avant le rendu via la commande magique `git log >ChangeLog`.

8.4.8 git commit

Nous arrivons sur l'une des commandes les plus capitales : le commit. Le commit mettra en application ce que vous avez pu constater avec la commande `git status`. Appeler cette commande lancera votre éditeur de texte favori et vous demandera un commentaire décrivant le commit. **Il est très important de mettre un commentaire clair et utile pour chaque commit sur son but**. Un commit avec un commentaire du genre ``lol'', ``prout'', ``mdr'', ``un commit'', ``ce projet me gonfle !'' ou encore ``salsifi'' ne sert absolument à rien ! C'est ce qui sera marqué dans votre commit qui remplira le log de votre dépôt. Imaginez que vous êtes sur un projet de plusieurs semaines, vous avez fait une modification importante sur une fonctionnalité de votre programme. Cinq jours plus tard vous vous rendez compte que votre modification a cassé une autre fonctionnalité que vous aviez codée auparavant, mais vous avez déjà fait plusieurs dizaines de commits depuis. Que préférez-vous, rechercher dans votre log le commit ayant une description du genre ``Modification de XXX dans la fonctionnalité YYY'' ou passer une heure à passer en revue chaque commit s'appelant ``prout'' et regarder si celui-ci a pu contenir des modifications sur cette fameuse fonctionnalité critique ?

Quoi qu'il en soit, la description du commit n'a pas besoin d'être un roman du moment qu'elle est autosuffisante. Pour gagner en rapidité, vous pouvez utiliser l'option `-m` pour spécifier directement votre commentaire, par exemple `git commit -m "bugfix de la fonction factorielle quand N < 0"` ou même `git commit -m "Exercice 3.2 fonctionnel"`.

Afin de s'assurer que vous utilisez correctement Git lors de vos projets, les logs de vos dépôts seront vérifiés lors de vos soutenances. L'absence de log, un log contenant seulement 2 révisions du type ``création du dépot'' et ``rendu'', ou encore des commits avec une description du style ``prout'' sera sanctionné.

Une astuce de git pour commiter tous les fichiers sans avoir à tous les ajouter réside dans l'utilisation de l'option `-a` ou `--all` lors de votre commit. L'avantage de cette option est qu'elle ne rajoutera pas les fichiers n'ayant jamais été suivis jusque là, vous n'avez donc pas à avoir peur d'ajouter des fichier indésirables à votre dépôt.

8.4.9 git diff

Vous avez fait de nombreuses modifications sur votre fichier `super_projet.c`, vous vous apprêtez à faire un commit car vos ajouts font enfin ce que vous voulez, seulement vous ne vous souvenez plus de ce que vous avez modifié depuis le dernier commit et ne savez donc pas quoi mettre en commentaire de celui-ci ? `git diff super_projet.c` vous listera l'ensemble des modifications que vous avez réalisées sur ce fichier depuis le précédent commit. Vous pouvez également demander à Git de comparer un fichier avec un précédent commit ou même entre deux commits différents (même dans des branches différentes). Pour cela, nous vous laissons regarder du côté de `git help diff`.

8.4.10 git checkout

Nous avons vu comment créer des sauvegardes, mais pas comment les utiliser, c'est là toute l'utilité de `git checkout`. Comme quelques exemples valent mieux qu'un long discours :

- Vous venez d'ajouter plusieurs fichiers, par exemple avec `git add *.txt`, et vous vous rendez compte que cela a également ajouté `top_secret.txt`, malheur ! Pas de panique, si vous n'avez pas encore commit, `git rm --cached top_secret.txt` le retirera de la liste des ajouts.
- Flûte ! Vous venez de faire `git rm fichier_utile.c` alors que vous ne vouliez pas le retirer de la surveillance de Git : `git checkout -- fichier_utile.c` et on en parle plus.
- Vous désirez restaurer vos fichiers `perceval.c` et `caradoc.h` dans leur état de la révision `5500bd8e0ae52775c9abf69749cde9c340` pas de panique :

  ```
  git checkout 5500bd8e0ae52775c9abf69749cde9c34001650b - perceval.c caradoc.h
  ```

  fera le travail !

8.4.11 git tag

Les tags sont un outil important de Git : ils vont vous permettre d'identifier une révision (un commit) à l'aide d'un nom (et non d'un SHA-1). Par exemple, lors de vos rendus, vous devrez avoir un tag particulier appelé `rendu-X` où X correspond à la version du rendu. Comme vous n'aurez pas tous les mêmes SHA-1 sur vos dépôts, nous utiliserons ce tag afin d'identifier l'étape de votre travail que vous aurez choisie de rendre. Afin de taguer votre commit, vous utiliserez `git tag -a <nom de votre tag>`.

Oui mais voilà, un membre de notre groupe a ajouté une nouvelle fonctionnalité ultra-méga-top géniale, je ne peux pas déplacer le tag ? Si, il suffit de taper la commande `git tag -a -f <nom de votre tag>` et le tag précédent sera déplacé vers le commit en cours.

Une contrainte du tag est que celui-ci doit être unique sur tout votre dépôt : vous ne pouvez avoir (même dans deux branches différentes) deux tags portant le même nom.

Vous pouvez lister les tags avec la commande `git tag`.

8.4.12 git clone

Tout au début de ce tutoriel, nous vous expliquions que, pour les projets organisés par le laboratoire des assistants vous n'auriez pas à `git init` votre dépôt. En effet, nous nous occuperons de générer vos dépôts à votre place. Afin de pouvoir travailler dessus, vous allez devoir obtenir une copie locale de ce dépôt sur votre machine. Cette étape se fait à l'aide de la commande `git clone <adresse du dépôt>`. Cela crée un répertoire portant le nom du dépôt ; ca y est, vous êtes maintenant prêts à travailler !

8.4.13 git push

Nous avions vu que lorsque vous effectuez un commit, Git conserve toutes les modifications en local. Vous pourriez, dans le cas du rendu par exemple, avoir besoin de propager votre travail sur les serveurs à partir desquels vous avez cloné votre dépôt. Cette opération se fait simplement par l'intermédiaire de la commande *git push*. Cette opération va envoyer au(x) serveur(s) TOUS les commits que vous avez effectués en local. Il se peut que parfois votre ``push'' soit refusé par le serveur car les

informations que celui-ci contient sont plus récentes que celles que vous essayez de lui envoyer. Dans ce cas, il faut récupérer les modifications distantes avant de pouvoir ``pusher'' (voir section suivante).

Note : Afin de pouvoir pusher vos fichier ainsi que vos tags, vous devez rajouter l'option `--tags`.

8.4.14 git pull

Parfois, vous aurez besoin de récupérer des données qui ont été modifiées par d'autres et envoyées aux serveurs. Exemple : Bob a terminé l'implémentation de la fonctionnalité X et l'envoie au serveur du laboratoire assistants. Alice qui a besoin de X pour tester sa fonctionnalité Y peut donc récupérer le travail de Bob par l'intermédiaire de `git pull`.

8.5 Configuration de Git

Git, comme bon nombre de logiciels sous UNIX, possède un fichier de configuration personnalisable ; mais vous n'allez pas le modifier à la main. Voici les commandes à utiliser pour faire la configuration de base.

1. git config --global user.name ``Login Xavier''
2. git config --global user.email login_x@epita.fr

8.6 Obtenir une clé ssh

L'une des premières étapes de cet exercice est de vous faire générer une paire de clefs SSH qui vous premettront de vous connecter au serveur contenant les dépôts Git. Lorsque le programme vous le demandera, nous vous conseillons de mettre une passphrase pour protéger votre clef d'utilisations non désirées.

```
42sh$ ssh-keygen -b 4096
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ~/.ssh/id_rsa.
Your public key has been saved in ~/.ssh/id_rsa.pub.
The key fingerprint is:
35:ef:a7:f3:69:b1:29:08:f5:bb:e3:20:08:8e:48:d3 cardon_c@acu_master_server
The key's randomart image is:
+--[ RSA 4096]----+
|                 |
|                 |
|          o      |
|  .      ..o     |
| o E.   S. ..    |
|...o . ..  .. .  |
|. . . . .......+ |
|         ...=o+. |
|           .+Bo  |
+-----------------+
42sh$
```

Vos clés seront disponibles dans le répertoire `$HOME/.ssh/`, vous devez copier le contenu de `id_rsa.pub` dans le champ prévu à cet effet sur votre profil intranet.

8.7 Initialisation

Pour ce TP, vous allez utiliser le même dépôt que pour vos exercices de ce soir : `2016/piscine-j0-tp/login_x`. Clonez-le.

Commençons par y créer un répertoire `dummy` rempli d'un répertoire vide s'appelant `vide`, des fichiers `README`, `my_logs`, `useless` à la racine d'un répertoire `foo` et d'un fichier `bar` dans celui-ci.

1. Ajouter les différents fichiers sur le dépôt.
2. Affichez l'état actuel du dépôt à l'aide de la commande `git status`, constatez le résultat suivant :

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
```

```
#
# new file:   ChangeLog
# new file:   README
# new file:   my_logs
# new file:   foo/bar
# new file:   useless
```

Vous pouvez constater que le répertoire `dummy` n'est pas présent.
Ajoutez un fichier vide `toto` dans le répertoire `dummy/vide`.
Faites un commit et envoyez-le sur le serveur.
Note : N'oubliez pas `origin master` !

8.8 Exercice : manipulations avancées

1. Trouvez une commande dans `git help` qui permette de renommer le fichier `foo/bar` en `foo/goo` sans utiliser `git add/rm`.
2. Mettez à jour le `my_logs` avec le message de votre précédent commit sans utiliser d'éditeur de texte.
3. Réalisez un commit avec comme message ``Advanced manipulations'' sans ouvrir l'éditeur de texte.
4. Utilisez `git diff` pour montrer les différences sur le fichier `my_logs` entre la révision `HEAD` et la précédente.
5. Créez un fichier `AUTHORS` à la racine du projet, et ajoutez-y votre login en respectant le format suivant :

```
42sh$ cat -e AUTHORS
* login_x$
42sh$
```

6. Committez avec comme commentaire ``My first AUTHORS''.
7. Mettez un tag `rendu-1` avec le token présent sur l'intra pour le rendu ``Rendu rapide git tp''.
8. Pushez avec prise en compte des tags.
9. Vérifiez sur l'intranet que votre rendu a bien été pris en compte.
10. Ajoutez le texte ``dummy'' dans le fichier `foo/goo`.
11. Committez avec comme commentaire ``useless manips for the tuto''.
12. Annulez votre commit.
13. Committez avec comme commentaire ``Great manipulations for the tutorial''.
14. Taguez votre projet pour le rendu ``rendu git tp''.
15. Pushez avec prise en compte des tags.
16. Vérifiez la prise en compte de votre rendu sur l'intra.

Nous avons fini avec Git pour ajourd'hui, si vous maîtrisez toutes ces commandes, l'utilisation de Git pour vos projets de l'année ne devrait plus avoir de secrets pour vous !

8.9 Pour aller plus loin

Nous n'avons vu qu'une petite parcelle des fonctionnalités de ce que vous offre Git. D'autres commandes Git pourront vous être d'une grande utilité une fois maîtrisées. Cependant, cette journée est déjà très chargée, et maîtriser les commandes vues cet après-midi sera déjà un très bon début. De plus, Internet regorge de tutoriels si vous voulez en apprendre plus sur Git.

9 Sessions à distance

Il est parfois intéressant d'être capable d'accéder à un machine sans avoir à se déplacer physiquement pour :

— tester son programme sur une autre architecture ;
— tester son programme sur un autre système d'exploitation ;
— accélérer un traitement en distribuant les opérations ;
— etc.

OpenSSH est un programme qui permet d'ouvrir une session distante sur un poste de travail, et cela de manière sécurisée : toute communication est chiffrée.

9.1 Généralités

La commande `ssh` permet d'ouvrir une session à distance sur une machine. `ssh` signifie *Secure SHell*. Pour se connecter à un poste de travail, il est nécessaire de fournir son nom en argument à la commande.

Quelques options utiles :

— `-l login` utilise un login différent du login courant. Une autre possibilité est de préfixer le nom de la machine par `login@`.
— `-p port` utilise un port de connexion particulier. Le port par défaut utilisé par ssh est le port 22.

Attention : lors de la première connexion sur une machine, un message de confirmation vous accueillera systématiquement :

```
The authenticity of host 'hostname (10.42.14.11)' can't be established.
RSA key fingerprint is 73:42:39:f6:c5:e3:21:86:69:f6:dc:d1:0e:0c:18:75.
Are you sure you want to continue connecting (yes/no)?
```

Vous devez répondre `yes` à ce message pour continuer.

Quelques exemples d'utilisation :

— `ssh -l login hostname` : ouvre une session à distance sur la machine *hostname* avec le nom d'utilisateur *login*.
— `ssh login@hostname` : même chose que précédemment.
— `ssh -p 42 hostname` : ouvre une session à distance sur la machine *hostname* sur le port de connexion *42*.

Notez qu'il est également possible de transférer des fichiers d'une machine à l'autre de manière sécurisée avec la commande `scp`. Veuillez vous référer à la commande `scp` pour plus d'informations.

9.2 Accès extérieur

L'ensemble des ressources informatiques que l'école met à la disposition des étudiants est également accessible depuis l'extérieur du PIE, en SSH. Pour vous connecter depuis l'extérieur, il est nécessaire d'ouvrir une session sur une machine de rebond appelée *gate ssh* accessible à l'adresse `ssh.epita.fr`, puis de se connecter sur un poste de travail dédié.

Vous avez à votre disposition des machines OpenSUSE, FreeBSD, etc.

Attention : il est obligatoire de se connecter à une autre machine à partir de la gate SSH. Depuis cette machine, vous n'avez le droit de lancer que les commandes `ssh`, sous peine de close compte !

Notez également que pendant toute la durée de la piscine, et lorsque les assistants le jugeront nécessaire, vous ne pourrez pas vous connecter à l'école depuis l'extérieur.

10 Les news

La consultation régulière des news est **obligatoire** : il s'agit du principal moyen de communication entre les assistants et les étudiants.

Vous devez au minimum lire quotidiennement les newsgroups suivants :

— `epita.adm.adm` (informations de l'administration) ;
— `epita.adm.a1` (informations spécifiques aux ING1) ;
— `epita.adm.apping1` (pour les étudiants apprentis) ;
— `epita.assistants` (informations des ACU/YAKA).

Sous peine de sanctions (close-compte, etc.), il est interdit de poster dans certains newsgroups (les quatre précédents en font partie). Veuillez vous reporter au règlement du BOCAL pour davantage d'informations.

Pensez également à ajouter le newsgroup associé à un projet (ou période) donné. Ces newsgroups sont indiqués sur chaque sujet de projet (par exemple, pour la piscine, vous devez suivre `epita.cours.c-unix.piscine`). Vous avez le droit de poster dans ces newsgroups pour poser toutes les questions non personnelles relatives au projet. Lorsque vous avez une question, pensez à vérifier qu'elle n'a pas déjà été postée dans les news et, dans le cas contraire, posez-la.

De manière générale, toute réponse ou news venant d'un assistant prévaut sur ce qui peut être dit à l'oral ou marqué dans un sujet.

Il existe aussi une multitude d'autres newsgroups sur tout et n'importe quoi : assos, cours, etc. N'hésitez pas à les consulter ni à y poser des questions.

Le newsgroup `iit.test` est également à connaître puisqu'il vous permet de faire vos tests et vérifier que vous poster en respectant la nétiquette.

10.1 Slrn

10.1.1 Fenêtre principale

— `a nom` : ajoute le newsgroup *nom* dans votre fenêtre
— `L exp` : ajoute les newsgroups correspondant à l'expression *exp* dans votre fenêtre
— `HAUT/BAS` : se déplacer de newsgroup en newsgroup
— `s` : s'abonner à un newsgroup
— `u` : se désabonner d'un newsgroup
— `g` : rafraichir la liste de news

— L : affiche/cache la liste de tous les newsgroups abonnés
— c : marquer le newsgroup comme lu
— P : poster une news dans le newsgroup pointé
— ENTER : rentrer dans un newsgroup
— ? : lancer l'aide
— q : quitter slrn

10.1.2 Fenêtre newsgroup

— HAUT/BAS : se déplacer de thread en thread ou de news en news
— ENTER : lire la news
— d : marquer la news ou le thread comme lu
— u : marquer la news ou le thread comme non lu
— c : marquer le newsgroup comme lu
— P : poster une news dans le newsgroup pointé
— f : poster une réponse à la news
— r : répondre par mail à la news
— F : envoyez la news à une tierce personne par mail
— / : rechercher dans la news
— s S : rechercher dans les titres de news (s pour suivant, S pour précédent)
— h : montrer/cacher le corps de la news
— ? : lancer l'aide
— q : retourner à la fenêtre principale
— Q : quitter slrn

10.2 Gnus

Pour utiliser Gnus, vous devez rajouter dans votre fichier .myemacs les lignes suivantes :

```
(add-hook .text-mode-hook .my-text-mode-hook)
(setq user-mail-address "login_x@epita.fr")
(setq user-full-name "Nom avec lequel vous voulez poster")
```

Pour lancer gnus : emacs -f gnus ou, dans Emacs, tapez M-x gnus.

10.2.1 Fenêtre principale

— SHIFT-A SHIFT-A : lister les news
— u : s'abonner à un groupe à partir de la liste
— j : s'abonner à un groupe dont vous donnez le nom
— l : lister les groupes où il y a des messages non lus
— SHIFT-l : lister les groupes auxquels vous êtes abonné
— c : marquer comme lus tous les messages d'un groupe
— ENTER : entrer dans un groupe
— g : pour mettre à jour les nouveaux messages
— M-g : pour mettre à new les messages sur un groupe

10.2.2 Fenêtre newsgroup

— ENTER : lire une news
— d : effacer la news
— a : poster une news
— f : répondre à une news
— SHIFT-f : répondre à une news en en conservant le contenu
— r : répondre à l'expéditeur par mail
— SHIFT-r : répondre à l'expéditeur en gardant la news

10.2.3 Mode rédaction d'une news

— C-c-d : annuler la news
— C-c-c : poster la news

10.3 Autres lecteurs de news

slrn et gnus ne sont pas les seuls lecteurs de news. Il en existe sur tous les OS et même sur vos téléphones. À vous de trouver le lecteur qui vous permettra de mieux suivre les news en toutes circonstances !

10.4 Exercice : mails et news

1. Envoyez un courriel d'au moins 5 lignes racontant vos vacances, via votre adresse courriel d'EPITA, à `netiquette@acu.epita.fr` pour voir si vous respectez la nétiquette.
2. Postez une news dans `itt.test` avec gnus ou slrn, avec les caractéristiques suivantes :
   — Sujet : login_x (votre login) ;
   — Corps : j'aime les ACU, c'est de la folie ! ;

<<<<<<< HEAD - Une signature qui respecte la nétiquette. La nétiquette se trouve sur l'intranet des assistants, dans la partie *Documents*.

3. Demandez à trois de vos camarades de répondre à votre news (en respectant la nétiquette évidement !).

Note : Afin de configurer votre webmail de l'école, connectez-vous sur `https://portal.microsoftonline.com`. Ensuite dirigez vous vers le menu d'option pour aller dans les paramètres. Renseignez une signature et choisissez texte brut pour la rédaction des courriels.

11 L'Intranet des assistants

11.1 Rubriques importantes

L'intranet des assistants possède de nombreuses fonctionnalités. Cependant, certaines sont plus importantes que d'autres et méritent particulièrement votre attention :

— *Documents* : coding style, nétiquette, modes d'emploi divers, etc. ;
— *Projets* : liste des différents projets, groupes, notation, etc. ;
— *login_x* : informations personnelles, clefs SSH, etc.

11.2 Informations personnelles

Connectez-vous sur l'intranet à l'aide de votre login et votre mot de passe qui vous a été fourni par les assistants.

En haut à droite de la page, un menu laissera apparaître un lien avec votre login ; cliquez dessus puis sur ``Profil'' pour commencer à modifier vos informations personnelles.

Les champs suivants doivent être impérativement remplis :

— pseudonyme ;
— numéro de téléphone ;
— date de naissance ;
— adresse email principale ;
— civilité.

12 Conclusion

Voilà pour le premier TP sur votre initiation au sein du PIE. Il est essentiel que vous maîtrisiez les différents éléments présentés car ils sont la base de la réussite de votre première année du cycle ingénieur de l'EPITA.

N'hésitez pas à lire et relire, encore et encore, ce document. Les assistants sont là pour vous aider et vos nouveaux amis sont **man** et les **moteurs de recherche** !

Why so sleepy?