Introduction aux systèmes d'exploitation

Cahier de TP, automne 2015

Table des matières

TP 1
Prise en main de l'environnement, premières commandes

Objectifs

— Découvrir l'environnement Linux des salles de TPs.
— Expérimenter avec la ligne de commande, découvrir les commandes de base.
— Savoir lire l'aide en ligne (`man`)

1.1 Premiers pas avec Linux

Loguez-vous sur une session Linux des salles de TPR2/R3. Faites une session enregistrée. Dans cette section, on utilisera les menus (en cliquant dessus à la souris).

1.1.1 Web, Mail

EXERCICE 1 ▶ **Web**
Pour vérifier votre connection au web, ouvrez un navigateur web (Firefox, Konqueror, ...) et accédez aux pages suivantes :

— La page du master CCI `http://master-info.univ-lyon1.fr/CCI/`.
— La page du cours de Système `http://laure.gonnord.org/pro/teaching/systemeCCI.html`.

Il n'est pas interdit de les "bookmarquer"...

EXERCICE 2 ▶ **Email, test**
Pour tester votre email, envoyez un mail à un de vos voisins sur son mail "académique", le mail devra avoir comme sujet :

```
[HS] Test de mon email Lyon1
```

et comme contenu

```
Ceci est mon email à Lyon1 .
    Prénom+Nom
```

Vous pouvez faire cet exercice en utilisant le webmail accessible à partir du portail étudiant `http://etu.univ-lyon1.fr/`, ou un logiciel de mail comme Thunderbird si vous l'avez configuré avec les bons paramètres.

Votre email est à regarder régulièrement !

EXERCICE 3 ▶ **À la maison**
Lire attentivement et régulièrement le texte "conseils de communication" disponible à l'adresse :

`http://laure.gonnord.org/pro/conseils.html`

EXERCICE 4 ▶ **Emails, filtres**
Se reporter à l'annexe et créer deux filtres d'email "CCI" et "CCI-Système" (webmail ou maileur classique), afin de gagner du temps et de l'efficacité à la lecture de vos mails.

1.1.2 Le terminal - premières commandes

EXERCICE 5 ▶ **Terminal**
Chercher dans les menus un logiciel nommé “Terminal”, ou “console” ou ...et en faire un raccourci sur le bureau. Ouvrir 4 terminaux à partir de ce raccourci.

EXERCICE 6 ▶ **Lancement à la ligne de commande et en arrière plan**
Lancer firefox en tapant dans un terminal :

```
firefox &
```

(on tape la commande, et on appuie sur la touche entrée).

1.2 Commandes de base

À partir de maintenant toutes les commandes seront tapées dans un terminal. Vous vous reporterez au cours pour la syntaxe des commandes utilisées (et les exemples).

1.2.1 Naviguer dans les répertoires

EXERCICE 7 ▶ **Expérimentations**
Les manipulations suivantes vont vous permettre de créer une hierarchie de répertoires, et de vous y-déplacer.

1. Vous venez de vous connecter/de lancer un terminal, dans quel répertoire vous trouvez-vous ?
2. Faites les manipulations suivantes :
   ```
   cd
   pwd
   cd /tmp
   pwd
   cd ~
   pwd
   ```
3. Regardez le résultat. A quel répertoire le ~ (tilde) correspond-il ?
   On peut toujours aller dans son répertoire principal / répertoire personnel, son “home” (en jargon système) en faisant simplement `cd`.
4. Créez le répertoire `ccic2` dans votre répertoire principal :
   ```
   mkdir ~/ccic2
   cd ~/ccic2
   ```
   ou de façon équivalente avec :
   ```
   cd
   mkdir ccic2
   cd ccic2
   ```
5. Affichez le contenu avec `ls` si vous vous trouvez dans ce répertoire (cela doit être le cas).
6. Créez un fichier vide (dans le répertoire `cci2`) nommé `foo.txt` en tapant la commande :
   ```
   touch foo.txt
   ```
   Réaffichez le contenu du répertoire.
7. `ls` peut également être utilisé comme cela :
   ```
   cd /tmp
   ls ~/ccic2
   ```
8. Faites `cd ~/ccic2` pour vous assurer que vous êtes dans le répertoire `ccic2` avant de continuer...
9. Quel est le contenu de ce répertoire ?
10. Créez un répertoire "toto", puis ré-affichez le contenu (pas du répertoire toto, mais du répertoire où vous vous trouvez, c.à.d. `~/ccic2`)
11. Effacez le répertoire "toto", puis ré-affichez son contenu.
12. Essayez d’effacer le répertoire "toto" à nouveau, que passe-t-il ?
13. Faites la manipulation suivante :
    ```
    cd /tmp
    ls
    ls -a
    ```

`ls -l`
vous pouvez saisir également :
`ls -a -l` ce qui est équivalent à `ls -al`

14. Quelle est la différence d'affichage ? À quoi servent les options `-l` et `-a` ? (faire `man ls` pour lire la documentation en ligne de `ls`).
15. Supprimez le répertoire `cci2` avec son contenu :
`cd ~`
`rm -Rf cci2`
Regardez la doc de `rm` et dire pourquoi cette dernière commande est dangereuse.

EXERCICE 8 ► **Reproduction**
En utilisant les commandes apprises précédemment :

1. À la ligne de commande, créer la hiérarchie suivante : Attention, les "`/`" ne sont là que pour signifier que

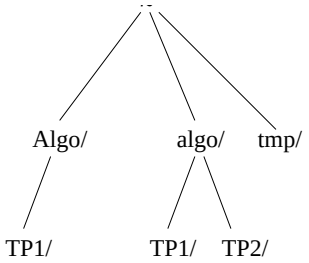


FIGURE 1.1 – Hiérarchie

ce sont des répertoires.
2. Se placer dans le répertoire `algo/TP1/` et y créer un fichier nommé `tp1.c` vide.
3. Supprimer le fichier `tp1.c` et tout nettoyer.

1.2.2 Afficher du texte

Utilisation des commandes `cat, less, more`.

EXERCICE 9 ► **cat, etc.**
Pour tester ces commandes on va manipuler les fichiers qui se trouvent dans `/etc/dictionaries-common`. Placez-vous dans ce répertoire.

1. Tapez :
`ls -l`
`cat words`
`cat words words`
Que fait la commande cat ? Peut-on visualiser de gros fichiers ?
2. Utilisation de `less` :
`less words`
puis
   — tapez sur la touche "espace"
   — tapez sur la touche "espace"
   — tapez sur la touche "espace"
   — tapez sur la touche "b"
   — tapez sur la touche "b"
   — tapez sur la touche "flèche du haut"
   — tapez `/foo` puis la touche entrée ou saut de ligne)
   — tapez `/z` puis (la touche entrée ou saut de ligne)
   — tapez sur la touche "flèche du haut"
   — tapez sur la touche "h" (regardez 30 secondes et continuez)
   — tapez sur la touche "q" (pour sortir de help)
   — tapez sur la touche "q" (pour sortir de less)
Que fait less quand on tape "/quelquechose" ?

3. Comparez `less` et `more`.

1.2.3 Commandes pour fouiller le texte

EXERCICE 10 ▶ **Utilisation simple de `wc` et `grep`**
Toujours dans le répertoire `/etc/dictionaries-common` :
1. Que fait la commande `wc <nomfichier>`, et ses options `-l` et `-w` ?
2. Que fait la commande `grep` :
   ```
   grep house words
   grep maison words
   ```

1.2.4 Éditer/modifier un fichier

Il existe des commandes pour modifier un fichier à la ligne de commmande, mais ici nous allons faire plus simple.

EXERCICE 11 ▶ **Fichier**
Créer un fichier texte dans le répertoire courant en utilisant un éditeur de texte :

```
gedit monjolifichiertexte.txt &
```

(on verra ce que signifie `&` plus tard)
1. Écrire vos noms et prénoms, sauver, et quitter l'éditeur.
2. Vérifier à l'aide de `less` que ce fichier contient bien ce que vous y avez écrit.
3. Quelle taille a ce fichier ?

1.2.5 Chercher des fichiers dans un répertoire

Nous allons expérimenter la recherche de fichiers.
Un petit exo pour découvrir la puissance de la commande `find` :

EXERCICE 12 ▶ **Découverte de find**
Exécuter :
1. `man find`
2. puis :
   ```
   find /etc
   find /etc -name "*.d"
   find /etc -name "*.cfg" -ls
   ls -R /etc
   ```
3. Que veut dire l'étoile ici ?
4. À quoi sert l'option `-name` ?
5. Commenter la différence entre `ls -R` et `find`.
6. Essayer la commande `find /etc -exec wc '{}' +` puis la commande
   `find /etc -name "*.cfg" -exec wc '{}' +`
   Que fait l'option `-exec` ?
7. Que fait l'option `-iname` ?

Noter qu'on parlera des caractères d'échappement plus tard .

1.3 Entrées et sorties standard, redirections et "pipe"

Quand un programme démarre, lui sont associées une entrée standard et une sortie standard. L'entrée standard est normalement associée au clavier, et la sortie standard est associée à l'écran, et plus précisément, la fenêtre où le programme est exécuté.

L'objet de ces exercices est de découvrir ces entrées/sorties et de voir comment on peut les rediriger. Un bilan sera fait en cours par la suite.

EXERCICE 13 ▶ **Lecture sur l'entrée standard**
Expérimentons encore !

1. Que fait la commande `sort` (voir la doc en ligne) ? Chercher dans la documentation la phrase qui dit ce que fait cette commande quand elle n'a pas de fichier en paramètre.
2. Essayer :
   ```
   sort
   a
   c
   d
   ```
   puis taper Ctrl + D (fin de fichier).
3. De même, essayer de trier les chiffres 2, 11, 1, puis expliquer pourquoi 11 est considéré inférieur à 2. utiliser l'option adéquate de `sort` pour réaliser le tri numérique.

La commande `sort` a lu l'entrée standard, au lieu d'un fichier.

Donc on voit que `sort` n'a pas lu un fichier mais a "lu" ce que vous avez tapé. Ces entrées et sorties standards peuvent être redirigées :

— la sortie peut être écrite dans un fichier et non pas affichée (avec >)

— la sortie peut être ajoutée à la fin d'un fichier et non pas affichée (avec »)

— l'entrée d'une commande peut être prise à partir d'un fichier et pas du clavier (avec <)

— la sortie d'un programme devient l'entrée d'un autre programme (avec | tube ou "pipe")

EXERCICE 14 ▸ **Découverte de la sortie standard et de la commande echo**

On se reportera souvent au manuel (man).

1. Que fait la commande `echo` ?
2. Faites les manipulations suivantes :
   ```
   cd ~/ccic2
   echo foo
   echo foo > fichier
   cat fichier
   echo toto
   echo toto > fichier
   cat fichier
   ```
   Observez que le deuxième echo écrase le contenu "foo" du fichier.
3. Avec >>, on ajoute à la fin sans écraser le fichier :
   ```
   echo foo > fichier2
   cat fichier2
   echo toto >> fichier2
   cat fichier2
   ```
4. À l'aide de sort, faire en sorte de générer un fichier `fichier.sorted` où les nombres 1, 2, 11 sont dans l'ordre croissant (ne pas oublier le caractère de fin de fichier). Le résultat de la commande est-il affiché sur le terminal ?
5. Tapez :
   ```
   sort < fichier.sorted
   sort < fichier.sorted > fichier.sorted2
   cat fichier.sorted2
   ```
   et expliquez.
6. Pourquoi les fichiers `fichier.sorted` et `fichier.sorted2` sont-ils différents ? (vous pouvez utiliser la commande `diff`).

EXERCICE 15 ▸ **Découverte du pipe**

Toujours dans le même répertoire :

1. Visualiser les lignes de `fichier.sorted` qui contiennent le caractère "1".
2. Compter le nombre de ligne de ce fichier.
3. Mais comment fait-on si l'on veut compter le nombre de lignes qui contiennent le numéro "1" dans `fichier.sorted` ?
   ```
   cat fichier.sorted
   grep 1 fichier.sorted
   wc fichier.sorted
   ```

`grep 1 fichier.sorted | wc`
Vérifier que cette solution ne génère pas de fichier temporaire.

L'avantage du tube est que si jamais une opération est bloquée (la production de la sortie, par exemple) on peut "consommer" l'entrée entre-temps (par exemple, chercher la ligne avec la lettre 1). Les concepts de production et consommation sont très importants.

EXERCICE 16 ▶ **Encore le pipe**
Commenter la/les différence(s) entre :

```
find /etc | grep cfg
ls -R /etc | grep cfg
```

EXERCICE 17 ▶ **Application/Révision**
Source : `http://www.lifl.fr/~forget/CoursSyst/`

1. Visualiser le répertoire `/usr/bin` page par page
2. Stocker l'aide de la commande `cat` dans le fichier `catAide.txt`
3. À l'aide de la commande `cat` construire le fichier texte de nom `titreCat.txt` contenant le texte suivant (**NB** : Ctr + D permet de "saisir" un caractère de fin de fichier) :
   ```
   ----------------
   - COMMANDE CAT -
   - EXPLICATIONS -
   ----------------
   ```
4. Stocker dans le fichier `catTitreAide.txt` la concaténation des fichiers `titreCat.txt` et `catAide.txt`
5. Visualiser le fichier `strange.txt`. Comme ce dernier n'existe pas, que se passe-t-il ? Relancer cette commande en redirigeant la sortie d'erreur vers le fichier `err.txt` ? Relancer cette commande en redirigeant la sortie d'erreur vers `/dev/null`, que se passe-t-il ?
6. Écrire le fichier `C` suivant :
   ```
   #include <stdio.h>
   int main()
   {
     fprintf(stdout, "message␣sur␣la␣sortie␣standard\n");
     fprintf(stderr, "message␣sur␣la␣sortie␣d'erreur\n");
     return 0;
   }
   ```
   puis le compiler et l'exécuter. Lors de l'exécution, voyez-vous une différence dans l'affichage des deux messages ? Pourquoi ? Redirigez maintenant la sortie standard vers `/dev/null`. Que constatez-vous ?
7. Afficher uniquement les 5 dernières lignes de l'aide de la commande `sort`
8. Afficher tous les fichiers commençant par la lettre 'f' du répertoire `/dev` ainsi que les informations sur ces fichiers (droits, taille, ...)

TP 2
Pratique de la ligne de commande

Objectifs

— Pratiquer les commandes vues en cours.
— Savoir de déplacer dans une arborescence Unix à la ligne de commande.
— Savoir lire le `man`.

Avant toute chose, terminez l'énoncé précédent.

Remarque : il est possible que nous soyions en avance sur le cours. S'y reporter en cas de problème.

2.1 Quelques nouvelles commandes utiles

EXERCICE 1 ▶ **Commandes** `gzip`, `tar`

Dans cet exercice, on va commencer par mettre en pratique les connaissances du TP1.

1. À l'intérieur d'un répertoire `Systeme/TP2/`, **créer** un répertoire nommé `toto` contenant deux fichiers texte, `titi` (contenant le seul mot titi) et `tata`, (contenant le seul mot tata)
2. Vérifier avec `ls` et `less`.
   La commande suivante n'a pas été vue en cours, mais elle est très utile, notamment lorsque l'on veut communiquer un ensemble de fichier à quelqu'un.
3. Créer une archive du répertoire `toto` et de son contenu : [1]
   ```
   tar -cvf toto.tar toto # crée l'archive
   ls -l toto.tar
   cat toto.tar
   tar -tvf toto.tar # donne le listing (le contenu)
   ```
4. Créer un autre répertoire (`Systeme/TP2/testtar`, par exemple), y-mettre l'archive et la désarchiver à cet endroit (`tar -xf`). Supprimer ensuite le répertoire `Systeme/TP2/toto`.
5. Compresser l'archive tar à l'aide de `gzip` (option `-9` ou `-1`). Vérifier que l'archive compressée est moins grosse que l'archive initiale.

Remarque : `tar` peut lui-meme faire appel à gzip (compression/décompression) en ajoutant l'option `-z` : `tar -cvzf toto.tgz toto` pour compresser, par exemple.

EXERCICE 2 ▶ **Commande** `diff`

À l'aide d'un éditeur de texte, dans le répertoire `/Systeme/TP2`

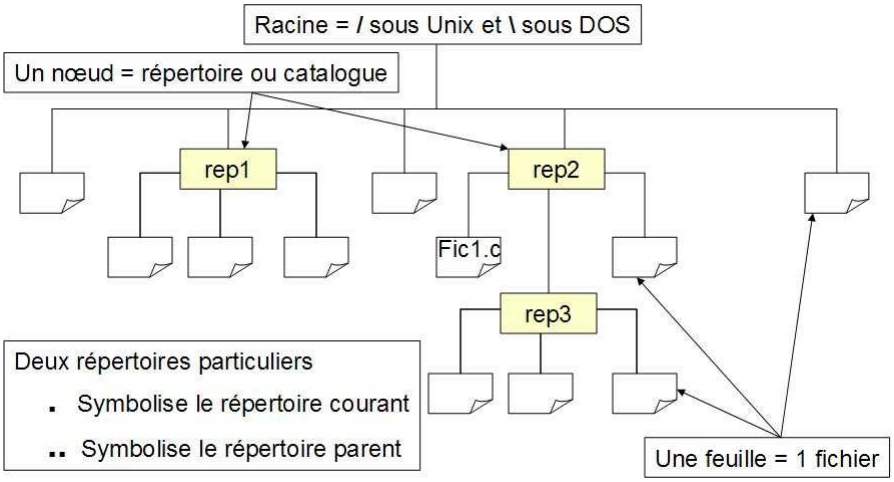
1. Créez un fichier1.txt avec 3 lignes différentes.
2. Créez un fichier2.txt qui est égal au fichier1.txt avec une ligne en moins.
3. Créez un fichier3.txt qui est égal au fichier1.txt avec en plus une ligne avec une phrase.
4. Créez un fichier4.txt qui est égal au fichier1.txt sauf que la deuxième ligne contient autre chose.
5. Faites `diff` entre les différents fichiers.
6. (Bonus pour les personnes en avance). Jouer avec la commande `patch` pour reconstruire le fichier2.txt à partir du fichier1.txt et de la sortie de `diff`.

On se contentera de retenir que `diff` n'imprime rien si deux fichiers sont identiques, et sinon donne des informations sur les différences entre les deux.

2.2 Arborescence de fichiers : chemins

À la (re) découverte des chemins relatifs et absolus. Toutes ses notions seront revues dans le cours "système de fichiers".

1. les dièses sont des commentaires, ils ne sont pas à recopier



— *Chemin absolu* d'un fichier :nom "complet" du fichier = chemin d'accès depuis la racine. Exemple, sous Unix : `/rep/fic1.c`.

— *Répertoire de travail* (répertoire courant) : répertoire sur lequel un utilisateur est positionné à un moment donné. Ce répertoire est nommé " . "

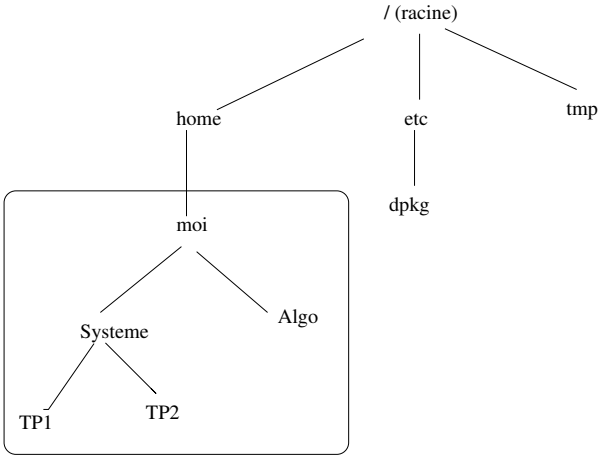
— *Chemin relatif* d'un fichier : référence d'un fichier à partir du répertoire de travail.

— *Répertoire personnel* (sous Unix) : répertoire sur lequel est positionné un utilisateur suite à son authentification. Ce répertoire est nommé " " (tilde). On parle aussi de "home utilisateur" ou de "racine personnelle". Sous Unix c'est souvent un sous-répertoire de `/home`.

EXERCICE 3 ► **Navigation relative et absolue**

Réaliser les manipulations suivantes :

1. Vérifier que la partie entourée du schéma suivant est bien conforme à votre répertoire personnel en salle de TP :



   Si ce n'est pas le cas, effectuer les modifications nécessaires (à l'aide des commandes `mv` et `mkdir` ?)
2. Se placer dans le répertoire TP2. Pour copier un fichier vers le répertoire TP1, vous pouvez au choix (réaliser chacune de ces manipulations) :
   — utiliser le chemin relatif (du répertoire TP1 par rapport à où je suis, ie TP2) : `cp nomfichier ../TP1/`
   — utiliser le chemin absolu du répertoire TP1 : `cp nomfichier /home/moi/Systeme/TP1/`
   — utiliser le raccourci de votre *home* : `cp nomfichier ~/Systeme/TP1/`
3. Se placer à la racine de votre répertoire personnel. *Sans se déplacer*, et en s'inspirant des manipulations précédentes :
   — copier un des fichiers du répertoire `/etc/dpkg` dans le répertoire `TP2`.

— lister le contenu du répertoire `TP1`.

— déplacer les fichiers qui n'ont rien à faire de `TP1` vers `/tmp`.

2.3 Pratique des variables du shell

Les exercices qui suivent pourront être reportés à plus tard, en cas de retard sur l'avancée des TPs.

EXERCICE 4 ► **Les variables**
Expérimentons :

1. Affectation des variables : (notez qu'il n'y a pas d'espace avant et après le caractère égal)
```
a=10 ; b=a; echo $b
b=$a ; echo $b ; echo "$b"
a=3; echo $b
```
2. Affectation des variables et affectation du résultat (re-notez les différents types de quotes)
```
datedujour=$(date)
echo $datedujour
datedujour=la date du jour est $(date)
datedujour=‘la date du jour est $(date)’
echo $datedujour
datedujour=" la date du jour est $(date)"
echo $datedujour
```

Remarque : Le ‘ (anti-quote ou backtick) est équivalent au `$(...)`

EXERCICE 5 ► **Variables locales, portée, commande** `export`
Variables locales et leur exportation (visible par d'autres programmes)

1. Tapez les commandes suivantes : (sans les commentaires)
```
b=1
echo $b
echo $$ # affiche le PID du processus du shell (son numéro)
bash
echo $$ # vous êtes dans un nouveau shell
echo $b # b n’est visible que sur le premier shell
exit # quitter le deuxième shell
echo $$ # vous êtes de nouveau sur le premier shell
export b
bash
echo $$ # vous êtes dans un nouveau shell
echo $b
exit
```
2. Que fait `export` ? Pourquoi n'exporte-t-on pas toujours toutes les variables ?

EXERCICE 6 ► **Commande** `expr`
Tapez les commandes suivantes et observez :

```
export a=10 ;
export b=5;
echo $a ’+’ $b
expr $a ’+’ $b
echo expr $a ’+’ $b
echo ‘expr $a ’+’ $b‘
echo $(expr $a ’+’ $b)
```

TP 3

Expressions régulières, recherche d'information

Objectifs

— Pratiquer les commandes vues (ou non) en cours.
— Savoir construire incrémentalement une ligne de commande complexe.
— Savoir lire le `man`.

3.1 Problème : recherche d'informations à partir de fichiers complexes

Source : Julien Forget, Univ Lille 1.

À l'école d'ingénieur Polytech'Lille, les étudiants ont accès à des imprimantes dans les salles de TP. Le nombre de pages imprimées est soumis à un quota, et les diverses impressions sont "mémorisées" ("loguées", en jargon système) dans un fichier "log" `/var/log/printaccounting` qui a l'allure suivante :

```
Thu Apr 6 10:30:23 CEST 2000|Vincent.Vandamme|hainaut.priv.eudil.fr|gayant|2
```

Soit : `date et heure | prénom.nom | nom_machine | nom_imprimante | nb_pages`

Le `nb_pages` reçoit le nombre de pages effectivement imprimées ou "aborted" ou "none" selon les cas d'échec à l'impression. Ce fichier `/var/log/printaccounting` se trouve sur un serveur de Polytech et ne vous est pas accessible. Les ingénieurs système de Polytech Lille ont bien voulu nous donner une copie partielle qui se trouve en local sur la page web `http://10.0.0.1/~jpgelas/`.

On se propose de répondre à quelques questions à propos de ce fichier de log, que l'on commencera par enregistrer sur le compte de l'utilisateur `moi` dans le répertoire `Systeme/TP3/`. Profitez des espaces pour prendre des notes.

1. Donnez le nombre de lignes de ce fichier (réponse : 791021)

2. Donnez le nombre de fois où l'impression est "aborted" (réponse : 48613)

3. Donnez le nombre de fois où l'impression est ni "aborted" ni "none" (réponse : 738466)

4. Donnez toutes les impressions à partir d'une machine kipper (kipper01, kipper02, ...) . Vérifiez bien sur le résultat de ne pas avoir affiché les impressions de l'imprimante nommée kipper[1].

5. Donnez les 10 plus grosses impressions. Attention !! Cela peut prendre du temps !! On obtiendra :

```
Tue Jun 11 19:17:36 CEST 2013|Pierre.Rey|gambrinus03.students.private.di|picasso|1448
Mon Nov 12 09:59:10 CET 2012|Aminata.Bah|hainaut.studserv.deule.net|schoeffer|860
Wed Jun 26 16:27:07 CEST 2013|Anthony.Vlietinck|hainaut.studserv.deule.net|picasso|796
...
```

1. Mauvais choix de nom de la part de l'ingé sys, qui du coup ne se facilite pas la tâche !

6. (*Bonus*) Listez les nombres de pages imprimées dans l'ordre décroissant. En cas d'égalité utilisez l'ordre alphabétique des noms d'imprimantes puis des `prenom.nom`. Réalisez un affichage page par page.

7. (*sed+(cut)+grep*) Transformez toutes les lignes "aborted" selon l'exemple suivant :
   `Thu Feb 3 11:54:31 CET 2000|Ahmed.Laib|gayant04|gayant|aborted`
   devient
   `Thu Feb 3 11:54:31 CET 2000|gayant|erreur`

8. Récupérer (au même endroit que précédemment), le fichier `passwd_polytech`. Les lignes de ce fichier sont de la forme :
   `vvandamm:x:1385:1070:Vincent Vandamme:/home/promo2000/vvandamm:/bin/bash`

9. Quel est le numéro utilisateur (`id`) de l'utilisateur de login `ndevesa` ?

10. (*tr+cut+sort+redirection*) Faire une copie *modifiée* de ce fichier, nommée `copie_passwd`, dont les lignes seront de la forme `Prenom.Nom|departementoupromo` et triée dans l'ordre alphabétique sur `Prenom.Nom`. La ligne :
    `vvandamm:x:1385:1070:Vincent Vandamme:/home/promo2000/vvandamm:/bin/bash`
    deviendra donc :
    `Vincent.Vandamme|promo2000`
    (Ma solution utilise 7 "pipe").

11. (*join*, Difficile ! Bonus++) À partir des fichiers `printaccounting` et `copie_passwd` réalisez, à l'aide d'une seule ligne de commande, un fichier "pages_imprimees" qui aura la forme suivante :
    `Prenom.Nom|imprimante|nb_pages|departementoupromo|`

TP 4

Scripts shells 1/2 : scripts simples

Objectifs

— Pratiquer les commandes vues en cours.
— Savoir construire des scripts shells et les exécuter.
On commencera par créer un répertoire nommé `TPscripts`.

4.1 Quelques nouvelles commandes utiles

EXERCICE 1 ▶ **Commande** `ping`
Une première commande réseau utile :

1. Regarder ce que fait `ping google.fr`.
2. Faire 100 requêtes `ping` sur une des machines lyon1 que vous connaissez (`bat710pfg`, par exemple) (chercher l'option adéquate dans le manuel de ping).

EXERCICE 2 ▶ **Commande** `wget`
Une autre commande réseau utile par exemple pour automatiser des téléchargements :

1. À l'aide d'un navigateur, récupérer l'adresse du PDF de ce TP (clic droit à la souris sur le lien pour copier l'adresse).
2. Récupérer ce fichier dans le répertoire courant (`Tpscripts` en utilisant `wget` dans un terminal (en collant l'adresse préalablement sélectionnée, toujours clic droit de la souris, "coller").

4.2 Quelques expériences autour de scripts simples

Préliminaires Si vous ne l'avez pas fait, faites rapidement la section "pratique des variables du shell" de l'énoncé 2 de TP.

Objet Dans cette section, on va taper et expérimenter des scripts simples. On les sauvegardera dans le répertoire `TPscripts`.

EXERCICE 3 ▶ **Script sans paramètre**

Un script shell est une suite de commandes qui auraient pu être exécutés dans un terminal. Transformons donc une ligne de commande simple en script shell :

1. Dans un terminal taper la commande qui permet d'écrire "Bonjour !" sur le terminal.
2. Copiez coller cette commande dans un fichier nommé `hello.sh`, et rajouter en haut de ce fichier la ligne `#!/bin/bash`. Sauver ce fichier.
3. Rendre le fichier exécutable :

   ```
   chmod u+rwx hello.sh
   ```

4. L'exécuter :

   ```
   laure@sorlin:$ ./hello.sh
   ```

5. Utiliser la variable `$USER` dans le script pour que l'exécution dise bonjour à l'utilisateur :

   ```
   Bonjour laure !
   ```

EXERCICE 4 ► **Les paramètres d'un script**
Tapez dans le fichier nommé param.sh :

```
#!/bin/bash

echo '$1' = $1
echo '$2' = $2
echo '$*' = $*
echo '$#' = $#
```

Lancez alors le script avec différents paramètres (sur la ligne de commande) et observez.

EXERCICE 5 ► **Utilisation de test + codes/valeurs de retour**

1. Mettez dans le fichier nommé `dedans.sh` :

   ```
   #!/bin/bash

   if grep $1 $2
   then
   echo $? : $1 se trouve dans $2
   else
   echo $? : $1 ne se trouve pas dans $2
   fi
   ```

2. Combien de paramètres (au moins) prend le script `dedans.sh` ?
3. Créez un fichier `grain.txt` contenant le mot 'grain'.
4. Exécutez :

   ```
   ./dedans.sh grain grain.txt
   ./dedans.sh autre grain.txt
   ./dedans.sh autre fichierquinexistepas.txt
   ```

5. Que contient la variable `$?` (dans le script) dans chacun des cas ?
6. Dans un terminal, regarder quel est le code de sortie (`echo $?`) de grep exécuté en dehors du script, dans les 3 cas (`grep grain grain.txt`, ...) . Vérifier à l'aide du manuel.
7. On veut supprimer la sortie ("standard") de `grep` pour que le seul message affiché en cas de succès soit celui du script (et non la ou les lignes imprimées par `grep` sur la sortie standard). Dans votre script, redirigez la sortie de `grep` vers `dev/null`, puis utilisez `if [ $? -eq 0 ]` pour vérifier que le code de sortie est bien 0.
8. Reproduire le comportement suivant de votre nouveau script :

   ```
    ./dedans.sh grain graincdczc.txt
   grep: graincdczc.txt: Aucun fichier ou dossier de ce type
   1 : grain ne se trouve pas dans graincdczc.txt
   ```

   Expliquer l'impression du nombre 1 plutôt que 2.
9. Traiter intégralement le cas d'erreur (redirection de la sortie d'erreur, et impression d'un message adéquat).
10. Que passe-t-il si on exécute `./dedans.sh mot` (avec un seul paramètre) ? Pourquoi ?
11. Ajoutez un test qui vérifie que le nombre de paramètres est égal à 2, et qui termine avec le code 1 si se n'est pas le cas (`exit 1`).

Faire attention aux espaces (espace après le crochet ouvrant, avant le crochet fermant, et avant/après les opérateurs de comparaison).

EXERCICE 6 ► **boucle** `for`
Tapez, rendez exécutable et exécutez le script suivant :

```
#!/bin/bash

for i in *
do
  echo $i
done
```

Expliquez, puis remplacez l'étoile par `$(ls -l /etc/pass*)`.

EXERCICE 7 ► **boucle**
Tapez, rendez exécutable et exécutez le script suivant :

```
#!/bin/bash

b=0
while [ $b -le 5 ]
do
  echo nom $b
  b=$(expr $b + 1)
done
```

Modifiez ce script pour qu'une valeur soit passé comme paramètre, e.g `boucle.sh 10`, et utilisant sleep pour "endormir" après chaque impression le processus de la valeur passée en paramètre.

EXERCICE 8 ► **Chaînes de caractères**

```
#!/bin/bash

n=$1
res=""
i=0
while [ $i -le $n ]
do
    res="$i,$res"
    i=$(expr $i + 1)
done
echo $res
```

1. Avant d'exécuter, essayer de deviner ce que fait l'appel : `./chaine.sh 3` (manipulation de la chaîne de caractère nommée *res*).
2. Exécuter et expliquer.
3. Que fait l'appel : `./chaine.sh` (sans argument). Faire en sorte d'éviter ce cas proprement.

EXERCICE 9 ► **Un "bashisme" intéressant**
Depuis 2004, Bash propose un mécanisme de tests d'expressions régulières, qui permet souvent de s'affranchir de `grep` ou `sed` :

```
[[ $var =~ expr_reg ]]
```

retourne 0 si la variable "matche" l'expression régulière. Vous testerez avec le script suivant :

```
#!/bin/bash

if [[ $1 =~ [0-9] ]]
then
    echo "le parametre $1 a un chiffre"
else
    echo "paf"
fi
```

Modifiez l'expression régulière pour n'accepter *que* les nombres (suites de chiffres strict, on utilisera donc `^` au début et `$` à la fin). [1]

EXERCICE 10 ▸ **Votre premier script**
Écrivez un script `existuser.sh` qui prend un login ou nom comme paramètre et qui regarde (avec grep) dans /etc/passwd si ce nom existe. Si tel est le cas, il affiche "oui M/Mme nom" existe ou "n'existe pas".

1. La subtile différence entre [[et [en bash est expliquée ici `http://mywiki.wooledge.org/BashFAQ/031`. Il n'est pas demandé/utile de la connaître.

TP 5

Scripts shells 2/2 : un script complexe.

Objectifs

— Pratiquer les commandes vues en cours.
— Savoir construire un script shell complexe.

On travaillera dans le même répertoire qu'au TP précédent. On utilisera les scripts du TP précédent comme "briques de bases" pour ce problème.

Source : François Boulier et Julien Forget (Polytech'Lille). Modifié en oct 2013

Introduction

On voudrait réaliser un script shell `pscopy` recevant trois paramètres sur sa ligne de commande : un entier positif c, et deux noms de fichiers `fin` et `fout`. Le fichier `fin` est censé être un fichier au format PostScript. L'utilitaire doit réaliser un nouveau fichier PostScript, `fout`, contenant c copies de `fin`, les unes à la suite des autres.

Exemple : Supposons que le fichier contenant `cc.ps` contienne une interrogation écrite destinée à 24 étudiants. On obtiendra un fichier `cc-fois-24.ps` contenant 24 copies de l'interrogation écrite par la commande :

```
$ ./pscopy.sh 24 cc.ps cc-fois-24.ps
```

Idée Utiliser l'utilitaire `psselect` (existant) pour réaliser les copies.

```
# la commande suivante réalise une copie de cc.ps dans a.ps
$ psselect 1- cc.ps > a.ps
# la commande suivante réalise deux copies de cc.ps dans a.ps
$ psselect 1-,1- cc.ps > a.ps
# etc.
```

5.1 Etude préalable

Avant de se lancer dans le script Shell, préparons le travail :

1. Récupérez dans votre répertoire de travail `TPscripts` un fichier PostScript que vous utiliserez pour tester votre script :
   — A l'aide de la commande `find`, recherchez un fichier `.ps` dans votre répertoire personnel ;
   — Si vous ne possédez pas de fichier `.ps`, recherchez un fichier `.pdf` et convertissez le en `.ps` à l'aide de la commande `pdf2ps` ;
   — Si vous ne possédez ni fichier `.ps` ni fichier `.pdf`, récupérez en un sur le web sur la page du cours.
2. Vérifiez que la commande `psselect` existe ;
3. Consultez le man de la commande `psselect` ;
4. Quelle serait la commande pour effectuer 4 copies ? Testez cette commande dans le terminal.

5.2 Traitement des paramètres

La suite de votre travail doit être réalisée dans un script shell. À partir d'un terminal où vous vous êtes placés dans le répertoire `TPscripts`, lancez un éditeur (par exemple, avec `emacs & `) et sauvegardez tout de suite votre fichier sous le nom `pscopy.sh`. **Toute modification du script fera l'objet d'un test avant de passer à la suite.**

Tout script Shell doit vérifier avec un soin particulier les paramètres qui lui sont passés, donc :

1. (*Préambule*) Afficher tous les paramètres de la commande (on utilisera pour cela une structure itérative qui parcourt `$*`) ;
2. Ajouter un test vérifiant le nombre de paramètres (il en faut 3) ;
3. Ajouter un test vérifiant que le premier paramètre est un nombre ;
4. Ajouter un test qui interdit d'écraser un fichier existant (cf commande `test`) ;
5. Ajouter un test vérifiant que le fichier source est bien un fichier PostScript. On pourra éventuellement utiliser la commande `file` (et grep !), ou alors `[[ ... =~ PostScript ]]`

Si les tests ne sont pas réussis, on sortira avec l'impression d'une erreur (sur `stderr`) puis `exit(1)`.

5.3 La commande de base

Nous allons maintenant réaliser la commande de base, sous la forme d'une fonction. Pour des raisons de structuration de code, on prendra bien soin de ne pas faire apparaître les tests du paragraphe précédent dans cette fonction. Votre fonction aura la forme :

```
do_copy ()
{
 echo "Appel à do_copy"
 #construction d'une chaine nommée carg

 #impression pour verifier que carg est bien formée
   echo $carg
 #invocation de psselect
   psselect $carg ...
}
```

1. Dans le script, écrire la fonction `do_copy` *avant* les commandes que vous avez déjà écrites. Mettre en commentaire tout le contenu de cette fonction, à part la ligne `echo`. La fonction ne fait donc rien à part imprimer un message.
2. Appeler cette fonction à la suite des vérifications des paramètres : `do_copy $1 $2 $3`. Vérifier que l'appel est bien réalisé.
3. Dans cette fonction, faire imprimer l'ensemble des paramètres passés à la fonction (`$*`). Vérifier.
4. En s'inspirant d'un des exercices précédents, construire la chaîne de caractères décrivant les pages à sélectionner (i.e. décrivant toutes les copies à faire) : pour 4 copies, on construira la chaîne `"1-,1-,1-,1-"`. Tester avec différentes valeurs pour ce paramètre.
5. Appeler `psselect` en utilisant cette chaîne.

5.4 Extensions (optionnel)

Nous allons maintenant réaliser quelques extensions au programme de base. Vous ne devez pas modifier la fonction réalisée dans la section précédente. Il doit de plus rester possible de spécifier les paramètres de la même manière qu'avant l'ajout des extensions :

1. Donner la possibilité de ne spécifier que 2 paramètres (`c` et `fin`), auquel cas on choisira automatiquement le nom du résultat. Exemple : `$pscopy 24 cc.ps` produira son résultat dans le fichier `cc-fois-24.ps` (cf `basename`) ;
2. Ajouter une option permettant de copier tous les fichiers `ps` contenus dans un répertoire (un même nombre de fois). Exemple : `$pscopy -a . 24` copie chaque fichier `ps` du répertoire courant 24 fois ;
3. Ajouter un test qui vérifie que ce répertoire existe bien !
4. (*Plus difficile*) Calculer et afficher le nombre total de pages créées. NB : le nombre de pages produites par `psselect` est affiché sur la sortie d'erreur.

5.5 Pour aller plus loin

Si vous avez du temps, faites l'exercice suivant :

`http://perso.univ-lyon1.fr/jean-patrick.gelas/doc/shell/tp3_sysex-cci.txt`

TP 6

Mini-sujet : Manipulation fichiers, droits

Objectifs

— Manipuler les droits des fichiers et répertoires en Unix.

6.1 Copie d'un fichier d'un répertoire distant

EXERCICE 1 ▶ **Commande** `ssh`
À l'aide de la commande `ssh` ou de son copain `scp` :

— Se connecter sur la machine du voisin avec le compte moi/moi (en récupérant l'adresse IP de ladite machine). Vérifier que le contenu de `tmp` est différent que celui de votre machine locale.

— Copier un fichier de votre compte de votre machine locale vers le `/tmp` de la machine distante.

— Effectuer un `ls` sur une machine distante (sans s'y loguer auparavant).

6.2 Système de fichiers, manipulation en mode utilisateur

Les manipulations de cette section se font comme d'habitude, directement au terminal. On travaillera dans un répertoire `TP6` créé spécialement pour l'occasion.

EXERCICE 2 ▶ **Liens physiques et symboliques**
Expérimentez les liens en Linux/UNIX :

1. Créez un fichier `foo` contenant la ligne `foo is foo`, puis affichez son numéro d'inode (voir le man de `ls`) ;
2. Créez une copie de ce fichier avec `cp` ;
3. Créez un lien physique puis un lien symbolique vers le fichier `foo` (commande `ln`) ;
4. Affichez le contenu de ces quatre fichiers ;
5. Comparez les inodes de ces quatre fichiers et expliquez ;
6. Supprimez le fichier `foo` et regardez le contenu des trois fichiers restant ;
7. Créez un nouveau fichier `foo` contenant la ligne `foo is not so foo` ;
8. Consultez à nouveau les inodes des quatre fichiers et leur contenu. Expliquez.

6.2.1 Création d'utilisateurs, groupes, et droits

Dans notre salle de tp, les machines ne possèdent qu'un seul utilisateur "normal", l'utilisateur `moi`. Pour expérimenter les droits, nous allons d'abord donc devoir créer des nouveaux utilisateurs.

EXERCICE 3 ▶ **Gestion des droits en mode super utilisateur**
Cette fois, nous allons passer en mode super utilisateur.

1. Devenir superviseur (avec la commande `su` mot de passe dans notre configuration `moi`)
2. Créer deux groupes, BLEU, ROUGE avec la commande `groupadd` (faire `groupadd -help`)
3. Créer quatre utilisateurs `ciel, cyan, carmin,magenta`, qui appartiennent à leur groupe correspondant (BLEU pour les deux premiers, ROUGE pour les deux suivants). Vous utiliserez les commandes `adduser` et `addgroup`.

4. Ouvrir quatre terminaux. Changer d'utilisateur avec `su <login>` dans chaque fenêtre devenir un utilisateur différent. Faire ensuite les exercices qui suivent.

EXERCICE 4 ► **Gestion des droits**
Pour cet exercice vous serez l'utilisateur `cyan` :

1. Affichez les droits de votre racine (personnelle) ;
2. Repérez le nom de votre groupe (commande `id`) ;
3. Créez un répertoire `Public` dans votre racine et faites en sorte que les membres de votre groupe puissent lire les fichiers qu'il contient. **NB** : votre répertoire racine doit rester illisible pour les membres de votre groupe ;
4. Créez un répertoire `Scripts` dans le répertoire `Public`, puis créez dans le répertoire `Script` un script Shell effectuant un simple `ls`. Faites en sorte que le script soit exécutable par les membres de votre groupe. Testez avec l'utilisateur `ciel`.
5. L'exécution du script shell modifie-t-elle les droits du fichier, son propriétaire ?

Dans la suite, vous serez de nouveau l'utilisateur "moi". Supprimez proprement vos utilisateurs "jouets".

6.3 Bonus : expérimentations pour le systeme de fichier

A ne faire que si vous êtes en avance.

EXERCICE 5 ► **Fichiers et allocation de blocs**
Les commandes suivantes vont nous permettre de comprendre comment sont alloués les nouveaux fichiers sur le disque.

1. A l'aide de la commande `stat` consultez le nombre d'inodes libres du système de fichier contenant votre répertoire personnel ;
2. Créez un fichier `essai` vide à l'aide de la commande `touch` puis regardez comment a évolué le nombre d'inodes libres ;
3. Avec `stat`, regardez la taille du fichier `essai` (taille en octets et nombre de blocs) ;
4. Ajoutez un caractère dans le fichier `essai` et consultez à nouveau les informations ci-dessus ;
5. Recommencez après avoir ajouté quelques caractères de plus ;
6. A l'aide de la commande `stat` affichez la taille des blocs de votre répertoire courant ;
7. Combien de caractères faut-il ajouter au fichier `essai` pour faire augmenter son nombre de blocs ? Essayez .On pourra par exemple utiliser le raccourci C-u sous emacs. Ex : "C-u 100 d" insère 100 fois le caractère d.

EXERCICE 6 ► **Fichiers et allocation de blocs**
Réalisez les expérimentations suivantes :

1. A l'aide de la commande `stat`, regardez le nombre de liens physiques sur le fichier `foo` de l'exercice 6.2. Créez un nouveau lien physique sur `foo` et relancez `stat` ;
2. Créez un répertoire `RepEssai` puis lancez la commande `stat` sur ce répertoire. Expliquez la valeur du champ `Link` ;
3. Ajoutez un sous répertoire `SousRep` dans le répertoire `RepEssai` puis vérifiez la valeur de `Link` (pour `RepEssai`). Comment expliquez vous ce changement ?
4. Observez la taille du répertoire `RepEssai` (retournée par `stat`) avant et après y avoir créé un fichier `fic`.
5. Pouvez-vous faire augmenter la taille de `RepEssai` uniquement en augmentant la taille de `fic` ? Vérifiez votre réponse ;
6. Faites augmenter la taille de `RepEssai` en y créant un certain nombre de fichiers. On pourra utiliser le `for` avec indice de boucle de `bash`.

TP 7

Système de fichiers 2 : programmation en C

Objectifs

— Utiliser les fonctions de manipulation des fichiers en C.

Important Les fonctions utilisées ici sont celles de manipulation de fichiers "bas niveau", *i.e.* "système". En particulier, chaque écriture/lecture fait un accès disque. La librairie ANSI C (voir cours) fournit des fonctions de plus haut niveau de manipulation des fichiers, que vous utiliserez dans le cas de manipulation de fichiers dans vos programmes de haut niveau (cours d'Algo).

7.1 Préparation : commande `tee` et manipulation de fichiers

EXERCICE 1 ▶ **Commande `tee`**
En utilisant tee (man tee) :

1. Écrire le résultat de la commande `ls` sur la sortie standard et dans un fichier en même temps.
2. Écrire le résultat de la commande `ls` dans trois fichiers de noms différents.
3. Écrire le résultat de la commande `ls` et dans le même temps, chercher un fichier précis dans la liste rendue par `ls`.

Sur la page du cours, récupérer les exemples de manipulation de fichiers C par appels systèmes (POSIX) et désarchiver l'archive `tgz` dans un répertoire spécialement créé pour l'occasion. Dans la suite travailler dans ce répertoire.

EXERCICE 2 ▶ **Commande `cat` (rappel)**
À l'aide de la commande `cat`, créer un fichier nommé `test1.txt` contenant un message personnel. Quels sont les droits de ce fichier ?

EXERCICE 3 ▶ **Manipulation d'un fichier existant**
Le fichier source C `fich1.c` fournit un exemple de manipulation de fichier en lecture :

1. Compiler, exécuter.
2. Comprendre ce que fait ce programme. Vous avez accès au manuel de `open` à avec `man 2 open`.
3. Que se passe-t-il si le fichier `test1.txt` n'existe pas ?
4. Que se passe-t-il si le fichier contient plus de N caractères ?
5. Changer (avec chmod) les droits de `test1.txt` (écriture seulement, pour l'utilisateur). Prévoir ce qui va se passer, tester.

EXERCICE 4 ▶ **Write**
Le fichier source C `fich2.c` fournit un exemple de manipulation de fichier en écriture :

1. Exécuter dans un terminal la commande `file test1.txt`.
2. Compiler le fichier, exécuter le binaire.
3. Regarder le contenu du fichier `test1.txt` après l'exécution du programme C dans lequel on a modifié le nombre d'octets écrits (dernier paramètre de la fonction `write`) et ce que retourne alors la commande `file`.
4. Que se passe-t-il si le fichier n'existe pas ?

EXERCICE 5 ▶ **Read/Write**
Le fichier source C `my_cp2.c` fournit un exemple de manipulation de 2 fichiers :

1. Compiler, et tester comme ceci :

   ```
   ./nomdubinaire test1.txt nomquinexistepas.txt
   ```

2. Vérifier qu'une copie du fichier `test1.txt` est réalisée, et expliquer pourquoi.
3. (man) Que signifient les options passées à `open` pour l'ouverture du deuxième argument de la ligne de commande (`argv[1]`) [1] *On pourra remarquer l'utilisation du "ou" bit à bit pour "ajouter" les flags entre eux.* Tester (avec un fichier existant comme deuxième argument, par exemple).
4. Comment est détectée la fin de la copie ? (man 2 `read`). Remarquer le test du `while`, qui fonctionne parce que l'affectation de n a pour valeur la nouvelle valeur de n.
5. (optionnel) Imprimer les valeurs numériques des différents flags et vérifier que le "ou" bit à bit fait bien son travail. On pourra s'aider d'une calculette binaire/décimal.

EXERCICE 6 ▶ **stdin, stdout, et utilisation de pipe**
Chercher dans le cours quels sont les descripteurs de fichiers (file descriptors) de `stdin` et `stdout`, puis écrire un programme C qui :

1. Lit des chaînes de caractères d'au plus 20 caractères sur l'entrée standard, séparés par des "entrées" (avec `read`).
2. Écrit les mêmes chaînes sur la sortie standard (avec `write`).

Comment ensuite utiliser le pipe du shell (`pipe`) pour afficher la sortie du programme dans l'ordre alphabétique des lignes ?

7.2 Simulation de la commande tee

Dans cet exercice [2] , pour les E/S sur les fichiers, vous utiliserez les fonctions système POSIX (`open, read, write`, ...) et non les fonctions ANSI C (`fopen, fprintf, fscanf`, ...).

Dans le programme que vous allez réaliser, l'analyse des arguments (et des options) doit être réalisée dans la fonction `main` à l'aide de `argc/argv`. La commande `tee` sera réalisée dans une fonction dédiée que vous appellerez avec les bons paramètres dans le `main`.

N'oubliez pas de tester les valeurs de retour des appels système pour vérifier que tout s'est bien passé !

1. Récupérer le fichier à remplir sur la page web du cours.
2. Programmer en C un clone du programme `tee` qui lit sur l'entrée standard et écrit sur la sortie standard, qui se lancera avec `./mytee`.
3. Ajouter l'écriture dans un fichier passé en paramètre.
4. (optionnel) Implémenter l'option `-a` de `tee`.

7.3 Pour aller plus loin

Un exercice supplémentaire (simulation d'un shell en C) est disponible sur la page web du cours.

7.4 Les codes du TP

1. Les curieux pourront aller voir une liste des "flags" disponibles ici `http://www.gnu.org/software/libc/manual/html_node/File-Status-Flags.html#File-Status-Flags`.
2. Source : TP de J. Forget (LIFL/Polytech'Lille).

```
/* Code source fich1.c − LG 2013 pour CCI − Lyon1 */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

const int N = 20; // buffer size

int main(){

  char afilename[12]="test1.txt";
  char buff[N];

  //tentative d'ouverture du fichier.
  //ouverture en lecture seulement.
  int f2=open(afilename, O_RDONLY);

  if (f2 == −1) { printf("open : failed\n"); exit(1) ; }
  else {
    int nbreal=read(f2,buff,N); // lecture de 100 chars au max
    printf("j'ai lu %d chars\n",nbreal);
    printf("Voici les chars lus : %s\n", buff);

    close(f2);// bien fermer proprement.
  }
  return 0;
}
/* Code source fich2.c − LG 2013 pour CCI − Lyon1 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>

const int N = 20; // buffer size

int main(){

  char afilename[12]="test1.txt";


  //tentative d'ouverture du fichier.
  //ouverture en écriture seulement
  // en supposant que le fichier existe.
  int f2=open(afilename, O_WRONLY);

  if (f2 == −1) { printf("open for write: failed\n"); exit(1) ; }
  else {
    char* mon_msg="oh le joli fichier";
    write(f2,mon_msg,strlen(mon_msg));
  }
  return 0;
}
/* Code source my_cp2.c − LG 2013 pour CCI − Lyon1 */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

const int MAX=1000
```



```
int main(int argc, char *argv[]) {
  int f1, f2,n;
  char buf[MAX];

  //vérification du nombre de paramètres.
  if (argc != 3){
    fprintf(stderr, "erreur parametres\n"); exit(1);
  }

  //ouverture readOnly du premier argument
  f1=open(argv[1], O_RDONLY);

  if( f1 == −1){ /* mode lecture */
    perror("ouverture fichier source impossible"); exit(2);
  }

  //ouverture du deuxième argument en écriture et ?
  f2=open(argv[2], O_WRONLY|O_CREAT|O_APPEND, S_IRWXU)

  if( f2 == −1 ){//700
    perror("creation fichier destinataire impossible"); exit(3);
  }

  //lecture, ecriture.
  while ((n = read(f1, buf, MAX)) > 0){
    write(f2, buf, n);
  }

  return 0;
}
```

TP 8
Processus (utilisateur)

Objectifs

— Manipuler les processus en mode “utilisateur” sous Linux.

Les deux TPs sui viennent prendront la même forme que le précédent (fichiers). le TP7 vous fera manipuler des processus en mode utilisateur sous Linux, et dans le TP8 vous utiliserez les appels systèmes et les fonctions C pour les processus.

Pour réaliser les exercices[1] de ce TP, vous n'utiliserez pas l'interface graphique (sauf pour éditer des fichiers) mais les terminaux non graphiques. Pour basculer de l'interface graphique vers le premier terminal, vous devez utiliser la combinaison de touches Ctrl + Alt + F1. Il faudra se loguer sur les différents terminaux (2 suffisent ici). ces terminaux sont nommés `tty1`, `tty2`, ... Par contre, pour écrire les différents scripts avec vos éditeurs graphiques préférés, vous pouvez lancer une session graphique et l'utiliser *uniquement à cette fin*.d

EXERCICE 1 ▶ **Un script shell**

Écrire un fichier de commande (script shell), nommé `boucle.sh`, qui réalise une boucle infinie (boucle tant que avec une condition toujours vraie) réalisant à chaque tour une écriture écran (commande `echo`) suivie d'une mise en sommeil (commande `sleep`). Tester ce script.

EXERCICE 2 ▶ **Mode interactif**

Faites les manipulations suivantes (en comprenant ce que vous faites, en vous reportant aux schémas du cours) :

1. Lancez `boucle.sh` en mode interactif dans le terminal `tty1`. Vous n'avez donc plus la main sur ce terminal, il faudra donc travailler ailleurs pour les autres questions.
2. Quel est le PID de ce processus ? (cf commande `ps`)
3. Combien de parents séparent votre processus du processus init ?
4. Tuez `boucle.sh` à l'aide de la commande `kill`. Affichez de nouveau la liste de vos processus.
5. Relancez le programme `boucle.sh` dans le terminal `tty1`.
6. Comment suspendre le processus ainsi créé tout en restant dans le terminal `tty1` ?
7. Comment reprendre l'exécution du processus ainsi suspendu ?
8. Comment tuer ce processus en restant dans le même terminal (deux possibilités) ?
9. Que font les options `-CONT` et `-STOP` de la commande `kill` ? Tester.

EXERCICE 3 ▶ **Passage en mode asynchrone (batch)**

Les différents terminaux que vous manipulez avec les raccourcis Ctrl + Alt + Fn sont associés à des fichiers device du répertoire `/dev`. Ainsi le terminal 1 correspond à `/dev/tty1`, le terminal 2 à `/dev/tty2`, etc. Il est possible d'écrire dans ces fichiers comme on écrirait dans n'importe quel fichier texte.

1. Relancez le programme `boucle.sh` en mode asynchrone à l'aide du symbole "&", dans le terminal `tty1`.
2. Essayez de tuer ou suspendre le processus à l'aide des commandes vues dans l'exercice précédent.
3. Tuez le processus `boucle.sh` à l'aide de la commande `kill` ;
4. Lancez le programme `boucle.sh` en mode asynchrone dans le terminal `tty1` tout en redirigeant la sortie standard vers le terminal `tty2` ? (attention, il faut que vous vous soyiez auparavant logué dans le terminal non graphique numéro 2). Tuez ce processus.

1. Ce Tp est très largement inspiré d'un TP de J. Forget pour Polytech'Lille

5. Créez un second fichier de commande `boucle2.sh` similaire au fichier de commande `boucle.sh` mais avec un affichage différent. Exécutez ce script, en mode asynchrone, dans le terminal `tty2` en redirigeant la sortie vers le terminal `tty1` et `boucle.sh` lancé à partir du terminal `tty1` en parallèle. [2]
6. Affichez la liste des travaux au moyen de la commande `jobs` et suspendez l'exécution du processus `boucle.sh` [3]. Affichez l'état des travaux pour les deux `tty`.
7. Reprenez l'exécution de `boucle.sh` en arrière plan (asynchrone) avec la commande `bg`. Suspendez l'exécution du processus `boucle2.sh`. Continuez l'exécution de ce processus en avant-plan (interactif) avec la commande `fg` ;
8. Tuez les 2 processus avec la commande `kill` appliquée au numéro de travail de chacun de ces processus.

Vous pouvez réaliser les exercices suivants dans un environnement graphique, et vous reporter à la partie du cours de processus que nous n'avons pas traitée.

EXERCICE 4 ▶ **Priorités**

Expérimentons un peu l'ordonnanceur Linux :

1. Écrivez un petit programme C qui boucle infiniment sur l'incrémentation d'une variable ;
2. Compilez, exécutez et observez l'occupation du processeur (avec `top` ;
3. Ajoutez une deuxième instance concurrente de ce programme et observez l'occupation puis tuez cette deuxième instance ;
4. Votre machine est-elle multiprocesseurs ? Si oui lire le manuel de taskset, et relancez la première instance sur le processeur 0.
5. Relancez une nouvelle (deuxième instance) à l'aide de la commande `nice` (`nice ./binaire`) et observez l'occupation. Si votre machine est multipro il faudra lancer `nice` en combinaison de `taskset` :

   ```
   taskset -c 0 nice ./nombinaire
   ```

6. Quelle est la priorité la plus forte : 0 ou 10 ?
7. Tentez de lancer votre programme avec une priorité plus forte (un nice négatif). Le résultat vous surprend-t-il ?

L'exercice suivant est à sauter si vous êtes en retard.

EXERCICE 5 ▶ **Répertoire** `/proc`

A chaque processus correspond un répertoire dans `/proc` ayant pour nom le PID du processus et contenant divers fichiers stockant les informations utilisées par le système afin de gérer les processus. Reportez-vous au `man` de `proc` pour plus d'informations.

1. Lancez la commande `boucle.sh` en redirigeant la sortie standard vers `/dev/null` ;
2. Retrouvez dans le répertoire `/proc` les informations suivantes liées à l'exécution de `boucle.sh` :
   — Le répertoire de travail, la commande de lancement du processus et les variables d'environnement ;
   — Les fichiers correspondant à chaque descripteur de fichier ouvert ...
   — Le statut du processus.

2. Oui, on peut lancer un processus alors qu'un autre est en train d'écrire sur le même `tty`, même si ce n'est guère pratique.
3. On lance `jobs` à partir du `tty` qui a lancé le processus qui nous intéresse. Mme remarque qu'auparavant, la saisie est "pollulée" par l'affichage créé par `boucle.sh`.

TP 9
Processus (programmation)

Objectifs

— Utiliser les primitives C de création de processus.

Ce TP est fortement inspirée d'un TP de B. Dupouy et S. Gadret pour ENST (`http://www.infres.enst.fr/~domas/BCI/Proc/TPproc.html`).

Attention, sur une machine multi-processeur on peut avoir à forcer l'exécution sur un seul processeur, dans ce cas il faut utiliser la commande taskset, par exemple : `taskset -c 0 nomduprog`

EXERCICE 1 ▶ **Fonctions `fork` et `getpid`**
Sur la page web du cours, récupérer le fichier d'accompagnement pour cet exercice (`fork1.c`), puis :

1. Le compiler, l'exécuter, et comprendre en faisant un dessin pourquoi ce programme qui n'a que 3 appels à `printf` imprime 5 messages sur votre terminal.
2. Décommentez maintenant la ligne suivante derrière l'appel à fork : `if (valeur == 0) sleep (4);` Que se passe-t-il ? Pourquoi ?
3. En utilisant la fonction `getppid` afficher les numéros des processus père.
4. Rajouter un appel à sleep quelque part, et vérifier que les numéros retournés par `getpid()` sont bien des numéros de processus valides (que vous pouvez tuer par exemple).

EXERCICE 2 ▶ **Application : père, et deux fils et commande `wait`**
En s'inspirant de l'exercice précédent :

1. Écrire un programme dans lequel le processus père crée deux processus fils. Tous les processus affichent leur PID.
2. Synchroniser correctement les processus pour empêcher le père de mourir avant ses fils.Il faudra deux appels à la fonction `wait`.
3. Vérifier que cette synchronisation est correctement effectuée en retardant la mort des 2 fils.

EXERCICE 3 ▶ **Synchronisation**
Écrire un code à deux processus, 1 père et 1 fils avec :
— Le processus père lit sur l'entrée standard un nom de fichier qu'il stocke dans la variable `fichier` (scanf)
— Le processus fils exécute la commande `wc -l fichier`, qui permet d'afficher le nombre de lignes dudit fichier.
— le père s'arrête après le processus fils.
Il faudra faire attention à la place du scanf et utiliser la fonction `execlp`.

Le dernier exercice est facultatif

EXERCICE 4 ▶ **Communication**
Lire l'excellente page sur la communication inter-processus par tubes (pipe) :

`http://www.lifl.fr/~marquet/ens/sem/tubes/tubes002.html`

tester les codes donnés et faire les exercices.

Bonus : des exos sur les signaux, sur la page web du cours.

Annexe A
Filtres de Mail

A.1 Avec un logiciel de mail “traditionnel”

Vous trouverez des documentations très bien faites, par exemple ici :

`https://support.mozilla.org/fr/kb/classer-vos-messages-en-utilisant-des-filtres`

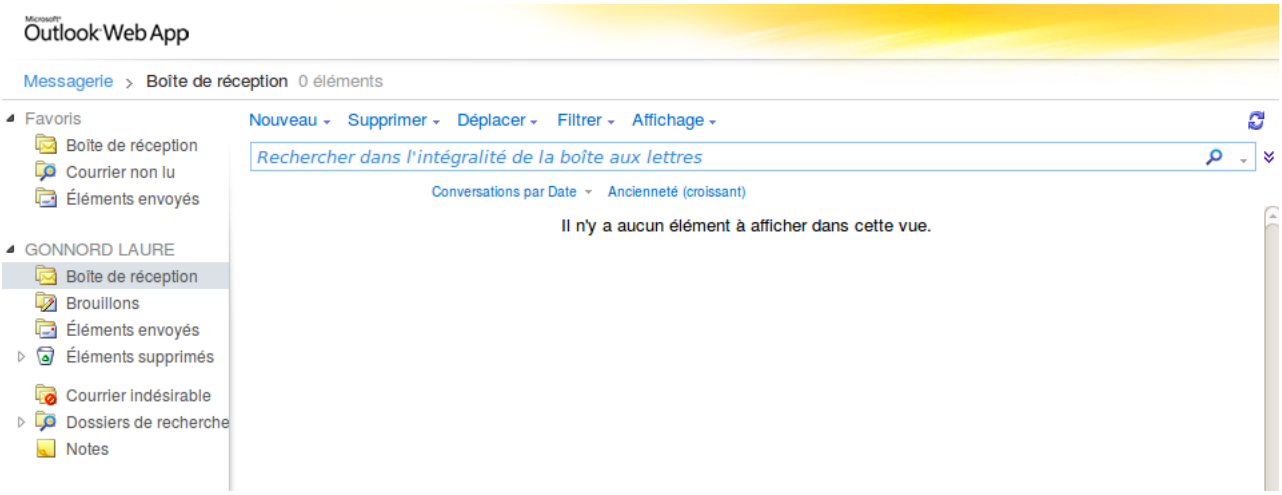
pour Mozilla Thunderbird.

A.2 Avec le webmail de l’université

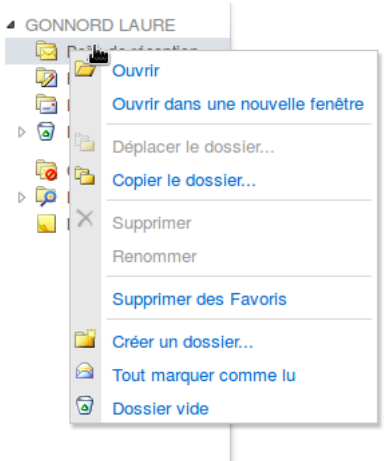
Le webmail de l’université est disponible à l’adresse :

`https://accesbv.univ-lyon1.fr`

La mise en oeuvre des règles de filtrage n’est pas disponible avec l’application “Light”. On fournit ici quelques copies d’écran pour vous aider (Outlook Web App). Après connection (login/mdp de l’université), on obtient :

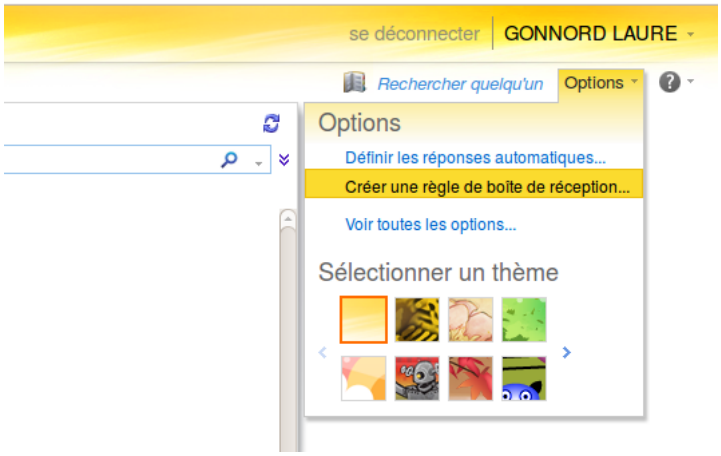


Pour créer des sous-répertoires pour les mails, cliquer sur la boîte principale, et “créer un dossier” :



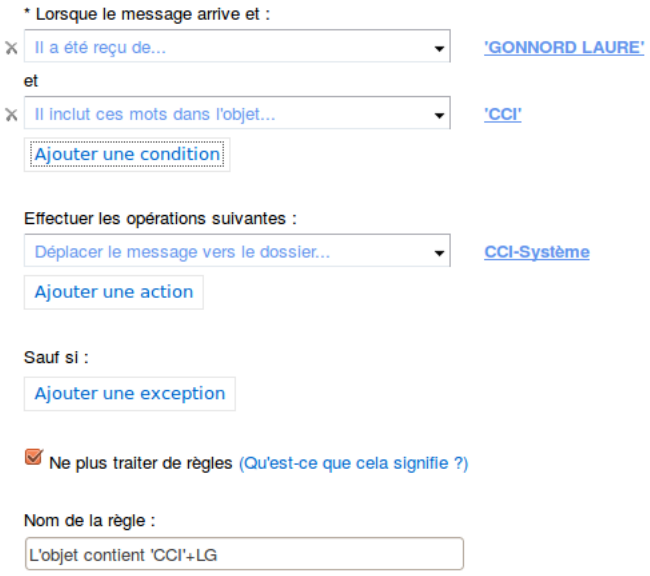
Il est demandé ici de créer les sous-répertoires `CCI` et `CCI-Syst`.

Une règle de filtrage est formée d'une condition (plus ou moins complexe), et d'une action. Grâce à ces règles, nous allons automatiquement mettre des emails dans les boîtes créées précédemment. Cliquer sur "boîte de réception", et chercher dans "options" :

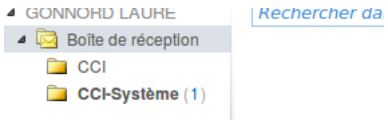


Voici la configuration pour une règle qui met automatiquement les mails contenant `CCI` en Objet dans la boîte aux lettres "CCI" :

Et voici celle qui met automatiquement les mails provenant de Laure Gonnord (avec l'email de l'université) avec un objet contenant `CCI` dans la boîte aux lettres "CCI-Syst" :



Enfin, voici ce qu'on obtient lorsque l'on reçoit un mail qui satisfait une règle :



À partir de maintenant, vous n'avez plus aucune excuse pour ne pas voir les mails de vos enseignants. (Je vous suggère de créer des filtres selon l'expéditeur).