LIFPCA – Programmation Concurrente et Administration Système

Fascicule de Devoirs Maisons – Corrigés

Les sujets de DMs ont été réalisés au fil des années, par les différents responsables de LIFPCA :
Fabien Ricco, Yves Caniou, Matthieu Moy, Grégoire Pichon, Florence Zara

Table des matières

DM 0 – Fondamentaux du shell

Pour réaliser ce qui suit, vous pouvez : utiliser une machine sous Unix (Linux, MacOS, en installation de base ou en VM), utiliser une VM sur la plate-forme cloud du Département Informatique, ou vous connecter sur `https:shell.univ-lyon1.fr` par exemple.
Dans la suite, l'énoncé suppose que vous êtes connecté en tant que l'utilisateur `etudiant` dans un shell `bash` – ce qui est le cas par défaut en utilisant une VM ASR7 d'Openstack.

1 Configuration du shell bash

Avant de commencer, un peu de vocabulaire :

— Un *shell* est un programme qui vous permet de lancer d'autres programmes. Dans ce cours on considère les shells Unix textuels, donc le rôle se résume à exécuter, en boucle, les étapes :

1. Afficher *l'invite de commande* (« *prompt* »)
2. Lire une ligne de commande au clavier
3. Exécuter la commande

— `bash` est le nom du shell Unix le plus utilisé aujourd'hui. D'autres shells sont `zsh` (qui a plus de fonctionnalités), `ksh` qui est un ancêtre de `bash`, `tcsh`, *etc.*

— Un *terminal* est un dispositif permettant d'accéder à un ordinateur. Historiquement, un terminal a d'abord été un ensemble clavier + écran (voire machine à écrire). Dans ce document, nous utilisons le mot « terminal » pour « terminal virtuel », c'est à dire une fenêtre graphique dans laquelle l'ordinateur lit et affiche du texte. En général, on ouvre un terminal pour exécuter un shell dedans (c'est fait par défaut).

— `xterm`, `gnome-terminal`, `konsole`, `eterm` ou encore `rxvt` sont des noms de terminaux classiques sous Linux. Vous pouvez également accéder à un shell en vous connectant en mode console (ctrl-alt-F1 par exemple sur la plupart des distributions, mais les ordinateurs des salles de TP de l'université sont configurés pour que ça soit le serveur graphique : il faut donc aller sur ctrl-alt-F2 (et suivant) – taper aft-F1 pour revenir à l'affichage graphique –. Le passage en mode console peut s'avérer pratique quand l'ordinateur répond moins et que l'on souhaite stoppper ou tuer des applications, voire simplement savoir ce qu'il se passe sur la machine).

— Il existe des systèmes d'invite de commande [1] pour avoir un shell avec certaines fonctionnalités/présentations configurées par défaut (certains ne peuvent être utilisés que pour un shell donné, donc à vérifier avant son installation).

  Parler de *terminal*, c'est parler de *tty*, abréviation de *teletype* (nous aurons l'occasion d'avoir un aperçu en regardant le résultat de la commande `w` par exemple). Pour avoir la configuration actuelle, ou définir une configuration (ajouter un raccourci clavier par exemple), on peut utiliser la commande `stty` : ainsi, si dans le résultat de `stty --all` on voit `Werase = ^W`, on sait que taper les touches `Ctrl` et `W` permettent d'effacer le mot précédent le curseur.

1. Un article assez fourni a été publié en novembre 2023 : `https://linuxfr.org/news/comparaison-critique-de-systemes-d-invite-de-commande`

1.1 Les variables d'environnement

Q1. En tant qu'utilisateur `etudiant`, exécutez les commandes :

```
TOTO=tutu
echo "$TOTO"
```

La première ligne crée une variable du shell nommée `TOTO` et de valeur tutu. La seconde affiche la valeur de la variable `TOTO` (donc la chaîne de caractères « tutu »).

Q2. Pour vous convaincre que les espaces sont significatifs en shell, essayez aussi :

```
TOTO = tutu
```

Ça ne marche pas (tentative d'exécuter la commande `TOTO` avec les arguments = et `tutu`).

Q3. Lancez un nouveau shell avec la commande `bash`. Vous aurez peut-être l'impression que rien ne s'est passé, en réalité :

- Le premier shell a créé un nouveau processus (primitive `fork()`)
- Le nouveau processus exécute la commande `bash` (primitive `exec()`)
- Le premier shell attend que le second termine (primitive `waitpid()`)

Les commandes que vous entrez maintenant sont donc exécutées par le nouveau shell.

Q4. Exécutez `echo "$TOTO"`. La commande devrait afficher une chaîne vide. Pourquoi ?

Correction

La variable `TOTO` est définie dans le premier shell, mais elle n'est pas héritée par le second (techniquement, l'appel à `exec()` réinitialise les variables).

Q5. Quittez le second shell (`exit`), et vérifiez que `echo "$TOTO"` affiche bien `tutu` maintenant que vous êtes revenu au premier shell.

Q6. Exécutez la commande `export TOTO` pour transformer `TOTO` en variable d'environnement. Une variable d'environnement est transmise aux processus fils (contrairement à ce qui vient de se passer quand nous avons affiché la valeur de `TOTO` dans le deuxième shell ci-dessus). Trouvez un moyen de vérifier que c'est bien le cas.

Correction

Il suffit de refaire la même manipulation que ci-dessus :

```
bash
echo "$TOTO"
```

Cette fois-ci la valeur tutu devrait être affichée.

Q7. La commande `env` permet d'afficher les variables d'environnement du shell courant. Vérifiez que vous retrouvez bien la définition de `TOTO`.

A retenir

- `VARIABLE=valeur` : affectation de valeur à la `VARIABLE`.
- `echo "$VARIABLE"` : Affichage de la valeur de la variable.
- `export VARIABLE` : faire de `VARIABLE` une variable d'environnement.
- `export VARIABLE=valeur` : raccourcis pour `VARIABLE=valeur; export VARIABLE`.

Q8. Rappel de MUNIX en L1 : quelle est la différence entre l'utilisation de `$TOTO`, `${TOTO}` et `"$TOTO"` ?

Correction

La notation avec `{` et `}` doit être utilisée dans le cas où un nom de variable est le préfixe d'un autre : un exemple simple est celui où l'on récupère les arguments donnés en paramètre d'une commande (`argv[]` en C) qui sont respectivement `$1 $2 $3 ... $10` *etc.* Ainsi utiliser `${10}` permet de s'assurer que ce n'est pas la valeur de `$1` qui sera utilisée concaténée au caractère '0' (`$0` est le nom du programme exécuté).

1.2 La variable d'environnement `PATH`

La variable d'environnement `PATH` définit les répertoires dans lesquels le système va chercher les exécutables quand on lance une commande. C'est une liste de répertoires séparés par des « deux points » (`:`).

Q1. Affichez le contenu de cette variable. Vérifiez que les répertoires classiques `/bin/`, `/usr/bin/` s'y trouvent.

Q2. Exécutez la commande `PATH="$HOME"/test1:"$PATH":"$HOME"/test2` puis affichez le nouveau contenu de la variable `PATH`.

Q3. Créez les répertoires `"$HOME"/test1` et `"$HOME"/test2`.

Q4. Créez un fichier `"$HOME"/test1/less` contenant :

```
#!/bin/sh
echo "Je suis test1"
```

Q5. Créez un fichier `"$HOME"/test2/less` contenant :

```
#!/bin/sh
echo "Je suis test2"
```

Q6. Lancez la commande `command -v less` pour voir quelle commande sera exécutée quand vous entrerez `less` en ligne de commande. Vérifiez que la commande se comporte comme prévu. Pour l'instant, les deux fichiers que nous avons créés ne sont pas considérés comme des exécutables car les fichiers n'ont pas le droit `x` (*eXecutable*).

Correction

`command -v less` va chercher dans le `PATH` le premier répertoire qui contient un exécutable `less`, qui doit être `$HOME/test1`. Il existe d'autres commandes faisant la même chose (`which` et `type`). `command -v` est celle qui est dans la norme POSIX.

Q7. Lancez la commande `chmod +x "$HOME"/test1/less "$HOME"/test2/less`. Ré-essayez les commandes `command -v less` et `less`. Si vous ne voyez pas de changement par rapport à la question précédente, exécutez la commande `hash -r`. Que remarquez-vous ?

Correction

Cette fois-ci, les fichiers sont exécutables. La commande `less` sera cherchée dans `"$HOME"/test1/less` puis dans les répertoires du système dont `/usr/bin` où se trouve la commande `less` par défaut, et enfin dans `"$HOME"/test2/less`. Comme la commande est trouvée dans `"$HOME"/test1/less`, la recherche s'arrête et notre premier script est exécuté. Vous devez donc voir :

```
$ command -v less
/home/etudiant/test1/less
$ less
Je suis test1
```

Q8. Fermez votre shell et ouvrez-en un nouveau. Ré-essayez les commandes `command -v less` et `less`. Que remarquez-vous ?

Correction

La variable `PATH` que nous avions positionnée n'est pas persistante d'un shell à l'autre. Notre nouveau shell a donc repris la valeur par défaut qui n'inclut pas les répertoires `"$HOME"/test*/less`.

Q9. De la même manière, créez la commande `"$HOME"/test1/which` (exécutable) en configurant `PATH`. Exécutez la commande `which which`. Que voyez-vous ?

Correction

À priori, le résultat continuera d'être consistant malgré la configuration que vous avez apportée : `bash` contient des commandes `built-in` dont `which`. Lire la manpage de `which` vous apprendra que pour utiliser le binaire `/usr/bin/which`, il faut définir une fonction de même nom (et nous verrons plus loin une autre syntaxe que celle utilisée dans la page man).

1.3 Les fichiers de configuration : `.bashrc`, `.bash_profile`

Les manipulations que nous avons faites jusqu'ici étaient temporaires : leur effet était limité au shell courant. La plupart du temps, quand on modifie une variable d'environnement (comme `PATH`), on souhaite que la modification soit persistante et que tous les shells ouverts dans le futur aient la nouvelle configuration automatiquement. Pour cela, nous allons entrer les commandes comme `PATH=...` non pas dans la ligne de commande interactive, mais dans des scripts utilisés à chaque démarrage du shell. Pour bash, ces fichiers sont `~/.bash_profile`, `~/.bash_login`, `~/.profile` et `~/.bashrc`. Nous nous limiterons à :

— `~/.bashrc` : au démarrage du shell

— `~/.bash_profile` : au démarrage des shells « login » (c'est à dire quand un shell est ouvert suite à une entrée de nom d'utilisateur et mot de passe, par exemple suite à une connexion SSH). Sur certains systèmes (comme Mac OS X), ouvrir un terminal lance un shell « login ».

En pratique, la plupart des éléments de configurations doivent être les mêmes pour les deux types de shell, donc nous allons nous arranger pour que `~/.bash_profile` inclue automatiquement `~/.bashrc`.

Les manipulations ci-dessous sont potentiellement dangereuses donc faites-les sur une VM (ou dans un compte utilisateur créé pour l'occasion). Si vous voulez travailler sur votre machine personnelle, assurez-vous que vous savez ce que vous faites à chaque étape.

Q1. Connectez-vous sur votre VM et faites toutes les manipulations ci-dessous sur ce compte.

Q2. Pour repartir à zéro, supprimez les fichiers `.bashrc` et `.bash_profile` s'ils existent (ATTENTION : à ne pas faire sur votre compte habituel Lyon 1 ou sur votre compte principal de votre machine locale !

Éventuellement, considérez un `mv` plutôt qu'un `rm` – votre `PATH` n'étant plus défini correctement par défaut, il faudra utiliser `/bin/mv` pour revenir à l'état initial).

Q3. Créez le fichier `.bashrc` avec le contenu :

```
echo "chargement de .bashrc"
```

Q4. Créez le fichier `.bash_profile` avec le contenu :

```
echo "chargement de .bash_profile"
```

Q5. Ouvrez une nouvelle connexion. Quel fichier est chargé ?

Q6. Depuis un shell existant, relancez un shell avec la commande `bash`. Quel fichier est chargé ?

Q7. Ajoutez les lignes suivantes au fichier `.bash_profile` :

```
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
```

La signification de ces lignes est : « si le fichier `~/.bashrc` existe, alors le charger ». C'est ce qui nous permettra d'écrire la configuration de l'utilisateur dans `~/.bashrc`, qui sera chargée quoi qu'il arrive.

Q8. Une proposition de configuration commune à tous les utilisateurs est disponible dans `/etc/profile`. Ajoutez les lignes suivantes en début de `~/.bashrc` pour en bénéficier :

```
if [ -f /etc/profile ]; then
    . /etc/profile
fi
```

Q9. Refaites les manipulations ci-dessus pour ouvrir des shells.

Q10. Ajoutez la ligne suivante au fichier `~/.bashrc` :

```
PATH="$HOME"/bin:"$PATH"
```

Q11. Créez le répertoire `~/.local/bin` (pensez à `mkdir -p` !) et placez-y un ou plusieurs fichiers exécutables (par exemple un script, comme nous l'avons fait avant pour `~/test*/less`).

Q12. Exécutez ces exécutables en entrant simplement leur nom. Vous aurez probablement besoin de relancer un shell pour que la modification du `PATH` soit prise en compte (ou de sourcer `~/.bashrc`).

Correction

Petite astuce pour relancer un shell : `exec bash`.

Q13. Supprimez les lignes `echo "..."` des deux fichiers : ces lignes étaient là pour nous aider à comprendre mais c'est une très mauvaise idée de produire du texte sur la sortie standard dans un de ces fichiers.

Q14. Question subsidiaire mais importante : quelle est la différence entre sourcer (`source script`) et exécuter (`./script`) ?

Correction

Sourcer permet de lire le contenu et d'appliquer les configurations au terminal en cours d'exécution. Exécuter va créer un nouvel environnement (`fork()`) et y exécuter le script : les modifications apportées à l'environnement par le script seront donc perdues lorsque le script aura terminé et le nouvel environnement été quitté.
On note qu'on peut sourcer un fichier qui n'a pas de droit d'exécution ; par contre, pour l'exécuter, il faut soit le lui mettre, soit demander explicitement à exécuter le script.

1.4 Un peu de confort et de couleurs !

La variable `PS1` contient une chaîne de caractères qui est affichée comme invite de commande (le *prompt*, que vous avez lue comme `$` dans les exemples de commandes à taper. Sa notation devient `#` lorsque les commandes sont tapées en tant qu'administrateur). Il est très utile d'avoir un prompt bien configuré : il peut l'être en y apportant de la couleur, des effet d'inversion vidéo, mais les codes couleurs peuvent également être utilisés de façon dynamique en fonction des branches `git` dans lesquelles vous évoluez par exemple ! Une bonne configuration permet également d'avoir un prompt « intelligent », comme nous allons le voir après. Configurer le prompt en rouge pour le compte administrateur permet par exemple de prendre un instant de réflexion avant de taper des commandes dangereuses pour le système (comme `hdparm` ou `dd`).

Q1. Essayez par exemple :

```
PS1="Bonjour maitre, que dois-je faire ? "
```

Q2. Essayez d'autres valeurs pour `$PS1` comme :

```
PS1="\h:\w\\\$ "
PS1="\[\e[31m\]rouge\[\e[m\] \[\e[32m\]vert\[\e[m\] "
PS1="\[\033]0;\u@\h:\w\007\]\[\033[01;32m\]\u@\h\[\033[01;34m\] \w \$\[\033[00m\] "
```

Q3. Vous pourrez plus tard générer un joli prompt personnalisé avec un outil comme `http://ezprompt.net/` si vous le souhaitez, et le déployer où vous le voulez.

Q4. Comment rendre une telle modification persistante sur votre compte, ou vos comptes (dont celui de Lyon 1) ?

La complétion dite « intelligente » permet, lorsqu'on appuie sur la touche tabulation (TAB) pendant qu'on entre une commande (donc après avoir tapé le nom de la commande !), de fournir une complétion dépendante de la commande et du contexte. Par exemple, `ls` *[TAB]* (si besoin, répéter *[TAB]* une deuxième fois) proposera des noms de fichiers, alors que `ls~--`*[TAB]* proposera les options de la commande `ls`. Pour l'activer il suffit d'inclure le fichier `/etc/bash_completion` dans un fichier d'initialisation du shell. C'est souvent le cas par défaut, mais si ce n'est pas le cas il suffit d'ajouter des lignes au `~/.bashrc`, généralement :

```
if [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
fi
```

Q5. Mettez en place également la complétion "intelligente". Vérifiez avec l'exemple ci-dessus (`ls`) qu'elle fonctionne.

Correction

Il faut s'assurer que le package `bash-completion` est installé, et faire `source /etc/bash_completion`. Attention la façon de procéder peut être différente sur d'autres distributions que Debian et Ubuntu (par exemple Gentoo..)

Q6. Avec `git` *[TAB]*, regardez la liste des commandes git disponibles. Comment entrer la commande `git commit --amend` en appuyant seulement sur 13 touches (pour 18 caractères) ?

Correction

`git com[TAB]--am[TAB]`

Q7. Exécutez sur des machines différentes `ifconfig` : est-il possible d'utiliser la complétion automatique pour compléter la commande (par exemple `ifc`*[TAB]* ? Pourquoi ?

Correction

Oui, mais elle ne fonctionne que si la commande est exécutable **et** dans `$PATH`. En local, il y aura une erreur car `ifconfig` n'est à priori pas dans `$PATH`, c-à-d `/sbin/` (ou `/usr/sbin/` sur certaines distributions) n'est pas dans le `PATH` de l'utilisateur par défaut.

Petite remarque : on utilise `ifconfig` ici pour l'exercice, car la commande est dépréciée (obsolète) et on lui préférera l'utilisation de la commande `ip` (Voir `https://fr.wikipedia.org/wiki/Ifconfig`).

1.5 Les alias

Les alias permettent de gagner un peu de temps en définissant des raccourcis pour les commandes qu'on utilise le plus souvent.

Q1. Exécutez et commentez pour chaque **ligne** suivante :

```
alias
l
alias l="ls -l --color=auto"
l
alias
```

Q2. Peut-on appeler une commande définie par un alias dans un script ?

Correction

Pour le savoir, il suffit de configurer un alias et de créer un script qui appelle cet alias, par exemple :

```
$ alias yo='echo Happy birthday'
$ echo '#!/bin/sh
yo
' > monscript.sh
$ chmod +x monscript.sh && ./monscript.sh
```

On voit que `yo` est considérée comme une commande introuvable : elle n'existe pas dans le `PATH`, l'alias n'est pas reconnu, même si défini de façon persistante.

1.6 Les fonctions

On peut aussi utiliser des fonctions, par exemple la suivante qui permet d'effacer automatiquement les fichiers postfixés par ~ qu'`emacs` crée comme backups avant l'édition des fichiers voulus.

```
function rmc {
    files=‘ls --color=never .*~ *~ 2> /dev/null‘
    if [ "x${files}" != "x" ] ; then
        for i in ${files}; do
            echo rm ${i}
            rm "${i}"
        done
    fi
}
```

2 Aller plus loin avec des scripts

Des commandes supplémentaires peuvent être pratiques dans les shells, par exemple `grep`, `cut`, `sed`, `tr`, mais aussi `bc` ou `dc` pour faire des calculs, ou encore `awk` qui permet de facilement de faire des `cut` et des opérations mathématiques (`bc` ou `dc`). Ainsi pour calculer une moyenne d'exécution d'un programme qui retourne son temps d'exécution, on peut par exemple utiliser :

```
tmp="scale=6 ; (0"
idxmax=3
for j in $(seq 1 $idxmax); do
    tmp="$tmp+$(./monprogramme)"
done
moy_duration=$(echo "$tmp)/$idxmax" | bc | awk '{printf "%f", $0}')
echo $moy_duration
```

Correction

On utilise ici `bc` pour faire des calculs : on établit dans son script qu'on souhaite des nombres codés sur 6 digits (0.000001 ou 999999) puis l'opération à effectuer. `awk` est ensuite utilisée car elle permet de convertir automatiquement la valeur .004 en 0.004.

Q1. Aller plus loin :

— `lesspipe -h` et `https://www-zeuthen.desy.de/~friebel/unix/less/README`
Exécutez `less filename` où filename est un fichier C, une archive ou `/bin/bash`.
Puis : `export LESSCOLOR=yes; export LESS="-R -M --shift 5"; LESSOPEN="| lesspipe %s"`
et recommencez. Commentaire ?

Correction

Attention, il y a plusieurs variantes de `lesspipe`, celle décrit sur le README pointé gère la coloration syntaxique paramétrée par `\$LESSCOLOR` mais pas celui fourni par défaut par Debian.

— Exécutez `man man` puis

```
man() {
env LESS_TERMCAP_mb=$'\E[01;31m' \
    LESS_TERMCAP_md=$'\E[01;38;5;74m' \
    LESS_TERMCAP_me=$'\E[0m' LESS_TERMCAP_se=$'\E[0m' \
    LESS_TERMCAP_so=$'\E[38;5;246m' \
    LESS_TERMCAP_ue=$'\E[0m' \
    LESS_TERMCAP_us=$'\E[04;38;5;146m' \
    man "$@"
}
man man
```

DM 1 – Rappels Système

1 Retour sur les scripts shell

Q1. On considère le script `convertoMP3.sh` suivant. Que fait-il ?

```
#!/bin/bash

# see http://en.linuxreviews.org/HOWTO_Convert_audio_files

#
# Usage: convertomp3 fileextention
#
if [ "x$1" = "x" ];then
  echo 'Please give a audio file extention as argument.'
  exit 1
fi

for i in *.$1
do
  if [ -f "$i" ]; then
  rm -f "$i.wav"
  mkfifo "$i.wav"
  mpv \
   -quiet \
   -vo null \
   -vc dummy \
   -af volume=0,resample=44100:0:1 \
   -ao pcm:waveheader:file="$i.wav" "$i" &
  dest=`echo "$i"|sed -e "s/$1$/mp3/"`
  lame -V0 -h -b 160 --vbr-new "$i.wav" "$dest"
  rm -f "$i.wav"
fi
done
```

Q2. Pourquoi
`$> convertoMP3 webm`
m'indique `command not found` ?

Correction

Si vous ne savez pas, retour sur ce qu'est la variable PATH (voire toutes les autres variables d'environnement, ce qu'elles sont et leur utilisation).

Q3. J'ai copié le script dans le fichier `convertoMP3`. Pourquoi la commande
`$>./convertoMP3 m4a`
ne s'exécute pas ?

Correction

Revoir les droits des répertoires et des fichiers : notamment ceux des binaires et scripts, qui se doivent d'être exécutables, et pour un script, il faut qu'il soit aussi lisible (voir l'impact des droits du répertoire sur ceux des fichiers qu'il contient !). Pas besoin ici, mais si vous lisez cette correction, il vaut mieux revoir les droits spéciaux comme +s.

Q4. À quoi sert l'extension `.sh` à la fin du nom du fichier ?

Correction

C'est purement décoratif : ça peut aider l'utilisateur à savoir ce qu'est le fichier, mais l'information sur son contenu est obtenu avec la commande `file`

Q5. Est-ce que `root` doit placer ce script dans `/usr/bin/` ?

Correction

NON ! Ça le serait s'il s'agissait d'une commande utilisable pour tout le système, c-à-d accessible également aux autres utilisateurs : c'est ce qu'il se passe par exemple lors de l'installation de nouveaux *packages* sur la machine. Mais **chacun peut avoir ses propres scripts** pour lui simplifier la vie. L'utilisateur peut créer un répertoire `~/bin` ou `~/.local/bin` pour cela.

Q6. On place `convertoMP3` dans `~/bin/` mais
`$> convertoMP3 wav`
ne fonctionne pas. Pourquoi ?

Correction

Il faut vraiment réviser ce qu'est la variable d'environnement `PATH`, et comment ajouter des informations au `PATH` ; et savoir comment faire pour que cette variable soit définie dans tout nouvel environnement créé (gestion de `~/profile`, `~/bash_profile` et `~/bashrc`).

2 Scripter pour automatiser – et se simplifier la vie

Jean n'a pas la data sur son smartphone. Il sait par ailleurs que tout transfer est mauvais pour le climat [2] et qu'il n'a pas besoin d'avoir des vidéos en 4k : elles consomment beaucoup de bande passante, épuisent sa batterie pour rien, et consomment en plus de l'espace de stockage s'il les télécharge. Mais il prépare un voyage en train, et avoir à disposition quelques documentaires disponibles sur `arte.tv` [3], quelques épisodes de série, en plus d'un bon livre serait agréable. Or il sait qu'il peut télécharger des vidéos avec l'outil `youtube-dl`...

Q1. Quelle est l'option qui permet d'afficher l'ensemble des flux vidéos et audio de l'URL entrée ?

2. `https://www.euractiv.fr/section/climat/news/cat-videos-online-porn-branded-an-ecological-disaster/`
3. notamment les épisodes `https://www.arte.tv/fr/videos/RC-015330/propaganda/`

Correction

Utilisation de `man`, `-F` ou `--list-formats`

Q2. Quelles sont les options qui permettent de ne télécharger que le flux audio, de préciser le type de flux audio ogg vorbis, en choisissant la meilleure qualité audio disponible ?

Correction

Utilisation de `man`, `-x --audio-format vorbis --audio-quality 0`

Q3. Jean appréciant particulièrement avoir des flux audio avec lui, il souhaite créer un raccourci commande pour cette commande : comment fait-il ?

Correction

Il utilise les alias ! Par exemple `alias yda="youtube-dl -x -audio-format vorbis -audio-quality 0"`
placé dans son `~/.bashrc`

Q4. Jean est aussi fan de XKCD. Cette fois-ci, plus de vidéo youtube, mais Jean aimerait télécharger les images en ligne de commande, pour pouvoir scripter les choses plus facilement. Il sait que l'URL de l'image se trouve derrière la chaîne de caractère `Image URL` sur les pages `https://xkcd.com/`< `numero` >. 1) Comment télécharger la page web (HTML) ? La commande `wget` devrait aider, mais comment a-t-on la liste des options disponibles ?
2) Comment avoir la ligne de la page web qui contient l'URL de l'image ?

Correction

`wget URL` pour télécharger n'importe quoi sur le net. Comme à peu près toute commande Unix, on accède à la documentation avec `man wget` dans un terminal (ou on cherche un tuto via son moteur de recherche favori. À noter les options `-r` (récursif) et `--no-parent` pour aspirer une partie ou un site web complet, mais elles ne nous serviront pas ici.

Pour obtenir la ligne, il suffit de faire un `grep`. On peut tenter :
`wget URL | grep "Image URL"`
Ça ne marche pas, car alors, `grep` s'applique à la sortie standard de `wget`, qui est vide, alors que le contenu de la page est envoyé dans un fichier. Seconde tentative :

```
prompt$ wget https://xkcd.com/2418/ --quiet -O - | grep 'Image URL'!
Image URL (for hotlinking/embedding): https://imgs.xkcd.com/comics/metacarcinization.
```

Bingo, ça a marché !

Q5. Jean dispose maintenant de la ligne qui contient l'URL, mais elle est de la forme suivante :
`Image URL (for hotlinking/embedding): https://imgs.xkcd.com/comics/metacarcinization.png`
Il faut donc découper la ligne pour en extraire l'URL de l'image. Comment faire ?

Correction

Une commande classique est `sed`, qui permet de faire des substitutions d'expressions régulières avec la syntaxe `sed 's/expression-a-remplacer/remplacement/'`. La commande est coupée en 3 lignes pour qu'elle rentre dans la largeur de la page (on a le droit d'entrer un retour charriot derrière un | ou un \), mais vous pouvez aussi l'entrer sur une seule ligne, ça marche aussi :

```
prompt$ wget https://xkcd.com/2418/ --quiet -O - |
    grep 'Image URL' |
    sed 's/Image URL (for hotlinking\/embedding): //'
https://imgs.xkcd.com/comics/metacarcinization.png
```

On peut maintenant utiliser `wget` pour télécharger cette image :

```
prompt$ wget https://xkcd.com/2418/ --quiet -O - |
    grep 'Image URL' |
    sed 's/Image URL (for hotlinking\/embedding): //' |
    xargs wget -O 2418.png
--2021-01-29 19:22:41--  https://imgs.xkcd.com/comics/metacarcinization.png
Resolving imgs.xkcd.com (imgs.xkcd.com)... 151.101.120.67, 2a04:4e42:1d::67
Connecting to imgs.xkcd.com (imgs.xkcd.com)|151.101.120.67|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 60776 (59K) [image/png]
Saving to: ‘2418.png’
100[=======================================>]  59.35K  --.-KB/s    in 0.02s
2021-01-29 19:22:41 (3.44 MB/s) - ‘2418.png’ saved [60776/60776]
```

On peut pousser un peu plus loin et télécharger plusieurs images d'affilée avec une boucle `while` :

```
for i in $(seq 2400 2418); do
    wget https://xkcd.com/$i/ --quiet -O - |
        grep 'Image URL' |
        sed 's/Image URL (for hotlinking\/embedding): //' |
        xargs wget -O $i.png
done
```

Q6. Script : Jean ne se rappelle plus forcément quel est le dernier numéro qu'il a regardé car il ne regarde pas les anime chaque semaine. Il souhaite : Stocker le numéro du dernier anime regardé dans un fichier getit.param, faire une boucle `while` qui téléchargera la page web du prochain épisode non regardé/téléchargé (l'URL de la page web est toujours du genre `https://xkcd.com/<numero>/`), puis récupérera l'URL de l'image pour la télécharger. La boucle `while` s'arrêtera quand `wget` s'arrêtera sur un status d'erreur parce que la page n'existe pas
Écrivez le script de Jean !

DM 2 – Rappels Système

1 Des commandes et une logique à connaître

Q1. À quoi sert `which` ?

Correction

Il y a la page man...

Q2. Quels sont les ensembles d'utilisateurs pouvant exécuter la commande `which` ?

Correction

`which which` qui nous donne a priori `/usr/bin/which` puis `ls -l /usr/bin/which` qui retourne quelque chose de similaire à `-rwxr-xr-x 1 root root 23504 6 déc. 2017 /usr/bin/which`. Ainsi, `root`, le groupe `root` et tous les utilisateurs peuvent exécuter la commande. Celui connaissant ses outils aura directement fait `ls -l $(which which)`

Q3. Quelle est la différence avec `whereis` ?

Correction

Il y a la page man. Il suffit de comparer le descriptif...

Q4. Peut-on utiliser la commande `locate` à la place de `which` ?

Correction

Elle retourne à priori plus d'informations que ce nous voulons avec `which` (pas seulement des commandes). Il faudra donc filtrer/trier toutes ces informations (essayez `locate which`...) ! En plus elle prendra le temps de s'exécuter en conséquence. Par ailleurs, ça nécessite que le ou les packages contenant `locate` et `updatedb` soient installés, et la base de données à jour. Donc non, on ne l'exécutera pas "à la place"...

Q5. Que fait la commande `dmesg` ?

Correction

Celui qui a exécuté la commande directement pour voir le résultat est un fou furieux : c'est l'équivalent de "exécuter `rm *` juste pour voir", bref une "roulette russe". Celui qui n'est pas convaincu peut s'essayer avec la commande `hdparm` et pourra ainsi griller son disque dur pour s'en convaincre...

Q6. Au vu des informations retournées par `dmesg`, que certains pourraient penser sensibles (donc à sécuriser), est-ce qu'un simple utilisateur peut utiliser la commande ?

Correction

Celui qui a exécuté la commande directement ne comprend pas la logique qu'on essaye généralement de mettre dans les exercices, notamment d'examen : il convient ici d'utiliser `which`, *etc.*

Q7. La commande `top` m'indique que `firefox` utilise du CPU alors qu'il est démarré mais que je ne l'utilise pas. Comment faire pour stopper firefox ?

Correction

Rappel d'ASR5 : utilisation de la commande `kill` qui **permet d'envoyer un signal** à un processus dont on connait le `PID` (processus ID).
`top` indique le `PID`, mais on peut l'obtenir aussi avec `ps aux | grep firefox`. Il suffit de faire ensuite `kill -s SIGSTOP PID`.

Q8. Je souhaite avoir deux commandes : `Fst` et `Fc`, la première permet de stopper `firefox`, l'autre permet de lui faire redémarrer son exécution. Comment faire ?

Correction

Du `ps aux | grep firefox`, du `cut` pour avoir le `PID`, des `kill` bien écrits avec les bonnes options, dans l'alias correspondant.

Q9. On souhaite obtenir les informations du noyau concernant les cartes réseaux. Comment faire ?

Correction

Réseau en anglais, c'est network.
Donc `dmesg | grep net` devrait fonctionner...
Une autre solution est d'utiliser la commande `lspci` qui liste un peu plus de périphériques.

Q10. On souhaite obtenir les informations du noyau concernant les disques durs, voire les espaces de stockage. Comment faire ?

Correction

En anglais, disque s'écrit "drive" et dur, "hard".
On pourrait essayer avec stockage, qui s'écrit "storage", mais... :
Avec un peu de culture générale Linux, on sait que les disques sont des *devices* apparaissant dans `/dev/` par un nom commençant par sd (sda, sdb, *etc.*).
On peut aussi faire un grep sur la métrique de stockage, le GB.

Q11. Je souhaite faire un *backup* de mon répertoire `$HOME/Work`. Comment faire ?

Correction

En utilisant correctement la commande `tar` avec les options `cvf`

Q12. Faire un script `Mybckup` qui permet de réaliser le *backup* précédent. Attention, il faut préserver toutes les sauvegardes !

Correction

Si le fichier existe, il faut d'abord renommer le fichier existant en `bckp1.tar` par exemple. Mais ce fichier existe peut-être lui-aussi ! L'algo : si fichier `bckp1.tar` existe, teste si `bckp2.tar` existe, *etc.* (boucle tant que), et à la fin, on renomme `bckupn.tar` en `bckupn+1.tar` et on redescend jusqu'à `backup1.tar`.
Puis on sauve.

Q13. Que fait la commande `date` ?

Correction

Il y a la page man...

Q14. Quelles options utiliser avec `date` pour obtenir le format `YYYY_MM_DD` ?

Q15. Comment faire son `tar` en utilisant `date` dans le nom du fichier obtenu ?

Q16. Comment mettre un job `cron` en place pour que toutes les semaines, le dimanche soir à 22h, un *backup* soit réalisé ?

Correction

On trouve facilement sur le net les informations pour remplir la `crontab`...

Q17. La machine qu'on vient de configurer pour exécuter une tâche de façon périodique peut être arrêtée à 21h : son propriétaire peut préfèrer lire quelque chose comme Terry Pratchett ou simplement se coucher plus tôt le dimanche soir. Il faudrait pourtant que le *backup* plannifié puisse avoir lieu au plus proche de l'hebdomadaire. Grâce à `anacron`[4], c'est possible : comment ?

Correction

Parmi les choses à retenir ici, le fait qu'`anacron` ne soit pas un démon, doive être lancé par exemple par `cron` – et réponde à notre besoin ;

4. `https://fr.wikipedia.org/wiki/Anacron`

DM 3 – Révision

1 Ordonnancement avec priorités

Nous utilisons un ordonnancement préemptif avec priorité [5] qui se tient à chaque unité de temps sur un système monoprocesseur. Nous allons utiliser un jeu de tâches dans un système préemptible.

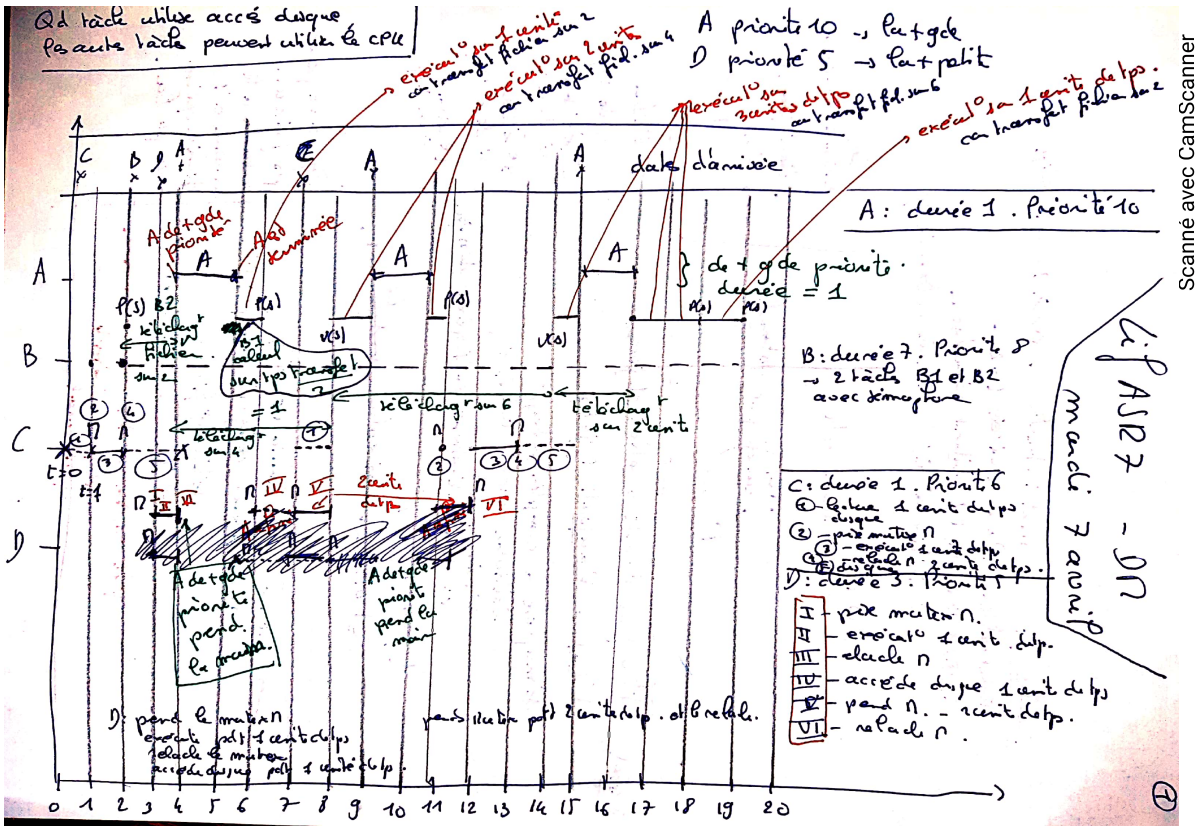
Tâche	Date(s) d'arrivee(s)	Priorité	Durée	Remarque
A	4, 9, 15	10	1	Ponctuelle
B	2	8	7	Ponctuelle. 2 tâches B1 et B2 partagent un sémaphore initialisé à 0 et sont lancées telles que : B1 exécute en boucle { P() et calcule pendant un peu moins du temps de transfert du fichier divisé par 2 } ; B2 exécute en boucle le { téléchargement d'un fichier puis V() } : le premier fichier demande un peu moins de 2 temps de transfert, le deuxième fichier en demande moins de 4, le troisième moins de 6, le quatrième moins de 2.
C	0, 7	6	1	Ponctuelle. À chaque fois, elle commence par faire une lecture disque pendant moins de 1 unité de temps, puis elle prend le mutex `M`, doit s'exécuter pendant moins de 1 unité de temps, relâche le mutex `M` et accède au disque pendant moins de 2 unités de temps.
D	3	5	3	Ponctuelle. Elle commence par prendre le mutex `M`, doit s'exécuter pendant un peu plus de 1 unité de temps et relâche le mutex `M`, accède au disque pendant moins de 1 unité de temps et prend le mutex `M` pendant moins de 2 unités de temps et le relâche.

Q1. Faire l'ordonnancement de ces tâches sur 20 unités de temps.

5. Plus la valeur de priorité est importante, plus la tâche est prioritaire

Correction

Le dessin ci-dessous.



Q2. Quelles sont toutes les observations que vous pouvez faire ?

Correction

Au temps 2, P(s) possible ou non. Sinon se fait en 5 et le calcul démarre immédiatement.
idle sur 0-1, 2-3 13-14.
temps 1-2, C prend mutex.
temps 3-6 puis 7-12, D possède mutex.
temps 11, mutex pris par C et endormie car D le possède.
termine à 19.

Q3. Quel est le temps de réponse (ou latence) de chaque tâche sur l'intervalle demandé (c'est-à-dire le maximum des durées constatées pour chaque tâche) ?

Correction

— A : 1
— B : 19-2=17
— C : max(4-0,15-7)
— D : 12-3

2 Et le travail en collaboration fût

Avec deux amis, c'est séance cuisine chez grand-mère : préparation de Kimchi, du choux chinois pimenté et légèrement fermenté dans de la saumure (eau salée) tant prisé en Corée, avec une recette un peu adaptée ici. Pour cela, chacun a amené un bocal pour stocker sa préparation, et un bol d'ingrédients nécessaires en quantité suffisante à la préparation : un bol de feuilles de choux préparées, un bol de daïkon et de carottes coupés en julienne, un bol de préparation de gingembre/pomme/poire/piment/ail/cébettes.

Il faut éviter le gâchis :

— Les 3 amis se mettent d'accord pour exécuter les mêmes actions, entre eux, et de façon répétée en **étapes**.
— À chaque étape, un ami a besoin de chacun des ingrédients : une fois servi, il place le daïkon et la mixture dans la feuille, la plie et la stocke dans son bocal.
— Et personne ne doit se servir en même temps, car il ne faut pas renverser. Ils demandent donc à la grand-mère d'agir de la façon suivante :
  — Lorsqu'il n'y a rien sur la table, elle prend au hasard 2 des ingrédients et les place sur la table.
  — L'ami qui possède le 3e ingrédient confectionne alors sa feuille avec les 3 ingrédients, et une fois pliée, débarrasse la table en prenant le temps de déposer proprement la feuille pliée dans son bocal.

On va modéliser ceci à l'aide de 4 sémaphores.

Q1. Dessinez un chronogramme de 4 étapes successives : vous représenterez les 3 amis et la grand-mère, et les actions effectuées par chacun. (l'objectif est de voir quel(s) est le(s) point(s) de synchronisation.)

Q2. Donner l'algorithme de la grand-mère (on remarquera que choisir 2 ingrédients est équivalent à choisir un ami donné).

Correction

On initialise T à 1. On initialise le tableau de sémaphores S[3] à 0.
while true { P(T) ; Choisit le xe ami V(S[x]) ; }

Q3. Donner l'algorithme de l'ami i.

Correction

while true { P(S[i]) ; Faire la feuille, la plier. V(T) ; Déposer la feuille proprement dans le bocal }

Q4. De quel type de problèmes classiques vu en cours appartient celui-ci ?

Correction

C'est une variante du diner des philosophes

Q5. Pour que l'après-midi cuisine soit une réussite, il faut que tout le monde puisse préparer une quantité à peu près égale de Kimchi. En conséquence, que doit-on supposer dans les processus décrits avant ?

Correction

Il faut simplement avoir une distribution uniforme lors des tirages au sort effectués par l'arbitre/la grand-mère.