# *Une introduction à l'environnement Linux pour des bricoleuses et bricoleurs*

*jb*

*6 mai 2018*

*Ce qu'on peut attendre de ce papier*

Ce document est conçu dans le cadre de l'atelier « Les données, du capteur au Web [1] ».

Il fournit les éléments essentiels pour comprendre, utiliser et modifier les programmes fournis par François.

1. https://konkarlab.linkpc.net/wiki/doku.php?id=start:projets:capteurweb

*Table des matières*

## *Linux koitesse ?*

Linux désigne, par abus de langage [2], un système d'exploitation libre (et donc *open source*).

Par nature, et contrairement à Windows, l'interface humain/machine est le terminal. Le fenêtrage graphique y est conçu comme une surcouche, c'est pourquoi on trouve de nombreux gestionnaires de fenêtrage graphique [3], mais dès lors que l'on veut bricoler et notamment réaliser des projets en électronique numérique, la connaissance de l'interface par terminal devient indispensable.

Nous allons donner ici quelques points de repère essentiels pour y survivre.

2. Voir `https://fr.wikipedia.org/wiki/Linux`

3. La distribution Ubuntu utilise notamment Unity, mais aussi Gnome, KDE, XFCE, etc.

### *Quelques ressources pour débuter*

En préalable, un mot sur la recherche d'information sur Internet. Les géants de l'Internet soutiennent (pour la plupart), par intérêt bien pensé, les logiciels libres mais leur philosophie particulièrement intrusive relativement à la vie privée interroge la notion même de liberté individuelle. C'est pourquoi il apparaît pertinent de se tourner vers des fournisseurs alternatifs [4].

Ainsi, pour la recherche d'information, de nombreux fournisseurs alternatifs aux géants du web existent, Startpage [5] notamment, dont le respect de la vie privée est particulièrement strict.

Concernant Linux, le site d'information le plus complet et le plus accessible au commun des mortels est ubuntu-fr [6].

4. Voir `https://framasoft.org/`

5. `https://www.startpage.com/fra/`

6. `https://doc.ubuntu-fr.org/`

### *Commandes pour s'y retrouver*

Sur Linux, toute commande est documentée. Pour obtenir des informations on pourra

| taper | explication |
|---|---|
| man | taper « man lacommande » affiche le manuel complet de la commande |
| apropos | taper « apropos quelquechose » affiche le résumé de chaque commande relative à ce quelque chose |

### *La gestion de paquets*

La gestion des logiciels comporte son lot de commandes [7] que nous verrons plus loin.

7. Il s'agit des commandes APT (*Advanced Packaging Tool*) de la distribution DEBIAN dont dérive UBUNTU.

Il existe aussi d'excellentes versions graphiques sur ce sujet, notamment l'Ubuntu Software Center ainsi que, plus généralement, le logiciel *synaptic*.

Quelque soit l'outil de gestion utilisé, terminal ou graphique, il est nécessaire de comprendre les principes globaux qui en sous-tendent l'usage.

UTILISATION DU TERME *paquet* plutôt que *logiciel* : le logiciel (sa version binaire adaptée aux caractéristiques de votre ordinateur) s'accompagne généralement

— de sa documentation,

— d'informations techniques et de configuration à répartir dans les différents répertoires systèmes,

— des librairies complémentaires nécessaires à son fonctionnement (équivalent des .dll sous Windows).

Logiciels, commandes, librairies, documentation, fichiers de configuration, tout ceci est empaqueté et se gère de la même manière.

LES PAQUETS SONT STOCKÉS DANS DES *dépôts*. Sous Ubuntu[8], il y a plusieurs types de dépôts :

— les dépôts officiels, répartis en quatre sections
  — *main* géré par la fondation Ubuntu, ne contient que des logiciels libres à strictement parler
  — *restricted* géré par la fondation Ubuntu, *a contrario* contient des logiciels dont la licence n'est pas libre
  — *universe* dépôt à activer manuellement, géré par la communauté, contient des logiciels libres
  — *multiverse* dépôt à activer manuellement, géré par la communauté, contient des logiciels non libres
— autres dépôts
  — *backports* pour obtenir la *dernière* version disponible des logiciels
  — *proposed* pour essayer des paquets encore en développement
  — *commercial* paquets de la société Canonical
  — *portage* pour le portage sur certains processeurs spécifiques

L'appel aux dépôts peut se configurer au travers des gestionnaires graphiques de paquets. Mais cette configuration peut aussi s'effectuer directement en modifiant (en tant que super utilisateur) le fichier /etc/apt/sources.list

8. Sous Debian, les dépôts sont gérés de manière différente, avec une dénomination spécifique, mais tout autant rigoureusement structurés.

Universe et Multiverse sont facultatifs mais chaudement recommandables.

Les dépôts non officiels sont à configurer seulement pour un usage très spécifique.

Configuration des dépôts avec /etc/apt/sources.list

LA GESTION DES PAQUETS AVEC LE TERMINAL s'effectue en tant que super utilisateur soit avec le logiciel *aptitude* soit au travers de commandes *apt* et notamment la commande « apt-get » qui permet :

| taper | explication |
|---|---|
| apt-get update | La commande update permet de resynchroniser un fichier d'index répertoriant les paquets disponibles et sa source. Ces fichiers sont récupérés aux endroits spécifiés dans /etc/apt/sources.list. |
| apt-get upgrade | La commande upgrade permet d'installer les versions les plus récentes de tous les paquets déjà présents sur le système en utilisant les sources énumérées dans /etc/apt/sources.list. Les paquets installés dont il existe de nouvelles versions sont récupérés et mis à niveau. L'*upgrade* doit naturellement être précédé d'un *update*. |
| apt-get install | La commande install est suivie par un ou plusieurs noms des paquets à installer. Tous les paquets requis par le(s) paquet(s) que l'on veut installer sont aussi récupérés et installés (on parle de *dépendance*). Le fichier /etc/apt/sources.list est utilisé pour retrouver les paquets désirés. |
| apt-get purge | Cette commande est suivie par un ou plusieurs noms de paquets à désinstaller. Elle va aussi effacer les fichiers de configuration correspondants. |

*Quelques commandes essentielles*

| taper | explication |
|---|---|
| cat | suivi du nom d'un fichier, permet de l'afficher sur la sortie standard |
| ls -l | affiche d'une manière détaillee le contenu (nom caché) du répertoire courant |
| ls -lA | affiche d'une manière détaillée la liste des fichiers cachés (commançant par « . » du répertoire courant) |
| mkdir | suivi du nom du répertoire à créer |
| cd | suivi du nom complet du répertoire où on veut désormais travailler |
| pwd | affiche le chemin complet du répertoire actuel |
| rmdir | suivi du nom du répertoire à supprimer |
| ps -a | affiche tous les processus en cours sauf ceux non associés à un terminal |
| y killall | suivi du nom du processus à tuer, permet d'arrêter un processus en cours (identifié préalablement avec la commande *ps*) |
| whereis | suivi du nom du programme pour afficher les répertoires où il est installé |
| chown -R | (commande super utilisateur) suivi de *utilisateur :groupe*, change le propriétaire et le groupe d'appartenance d'un fichier et modifie récursivement l'ensemble des fichiers et répertoires en dépendant. |
| chmod | pour modifier les droits d'une fichier. Pour lancer cette commande il faut soit être propriétaire du fichier soit être super utilisateur (commande *sudo*). Par exemple pour rendre un fichier exécutable, on entrera la commande « chmod +x » suivi du nom du fichier. On peut ainsi accorder (par +) ou empêcher (par -) la lecture (r), l'écriture (w) et l'éxécution (x) d'un fichier donné. |

PETITE PRÉCISION SUR L'USAGE DE LA COMMANDE CD :

— cd suivi du nom d'un répertoire comprend qu'il s'agit d'un sous-répertoire du répertoire courant

— cd suivi de .. comprend qu'il s'agit du répertoire au-dessus du répertoire courant

— cd suivi de ~ comprend qu'il s'agit du répertoire HOME

*Un mot sur les redirections*

Nous allons aborder ici l'usage le plus simple des redirections.

Pour rediriger l'affichage du résultat d'un programme ou d'une commande vers un fichier, comme par exemple de lister le contenu du répertoire courant dans le fichier voir.txt, on tapera :

```
ls -l > voir.txt
```

Pour transmettre le résultat d'un programme ou d'une commande vers l'entrée d'un autre programme ou commande, comme par exemple obtenir la liste de tous les programmes installés utilisant le terme « libre » dans leur dénomination ou leur résumé, puis utiliser le programme *less* pour l'afficher, on tapera :

```
apropos libre | less
```

*Pour aller plus loin*

D'autres commandes ou logiciels non graphiques sont particulièrement intéressants. En voici quelques uns, pris plus ou moins au hasard :

| commande | explication |
|---|---|
| crontab | pour lancer des programmes et activités à intervalles réguliers |
| rsync | pour réaliser des sauvegardes synchronisées |
| convert | commande du paquet « imagemagick » pour convertir un format d'image vers un autre |
| wget -ct o | suivi d'une adresse internet pour télécharger un fichier de manière non interactive |

### *Python koitesse ?*

Python est un langage libre, ce qui a pour conséquence qu'on trouve gratuitement et librement sur Internet une documentation abondante et de qualité.

Python est un langage puissant, professionnel, conçu pour être aisément maintenable. La communauté autour de ce langage est extrêmement dynamique et son usage se généralise [9] sur de nombreuses niches : ingénierie système, intelligence artificielle, robotique, ingénierie financière, réseau informatique, électronique digitale, analyse de données massives, etc.

Il apparaît donc judicieux, pour sa culture de bidouilleu(se)/(r) ou son avenir professionnel, de s'investir un peu sur ce langage, d'autant que le ticket d'entrée est faible, comme on le verra plus loin.

9. Voir par exemple `https://www.developpez.com/actu/150166/IEEE-Python-devient-le-meilleur-langage-en-2017-`

### *Quelques ressources pour débuter*

L'article sur Wikipedia [10] est naturellement à lire pour les aspects historique et utilisation de Python à travers le monde.

10. `https://fr.wikipedia.org/wiki/Python_(langage)`

Pour bien débuter en Python [11], on ne peut que recommander l'ouvrage très pédagogique de Gérard Swinnen paru chez Eyrolles dont la cinquième édition est librement téléchargeable [12].

G. Swinnen donne sur son site [13] d'autres ressources intéressantes, notamment le livre de cours de Robert Cordeau, librement téléchargeable [14] ne serait-ce que pour sa webographie, son mémento Python 3 et son abrégé Python 3. Le chapitre sur « Le développement dirigé par la documentation » ne pourra être lu qu'avec profit.

11. Toutes les références utilisent la version 3 de Python, et non les antérieures, pour les raisons expliquées dans l'article de Wikipedia.
12. `http://inforef.be/swi/python.htm`
13. `http://inforef.be/swi/python.htm`
14. `https://perso.limsi.fr/pointal/_media/python:cours:courspython3.pdf`

La documentation en français de la *Python Software Foundation* est très complète [15].

15. `https://docs.python.org/fr/3.6/`

La bonne syntaxe de programmation Python est disponible (en anglais) sur le site de la *Python Software Foundation*. Appelée PEP8, sa lecture est un peu abrupte. On conseillera plutôt cette lecture : `http://pep8.org/` en anglais. Enfin, Richard Bronosky l'a mis en pratique [16] sur un programme d'exemple.

16. `https://gist.github.com/RichardBronosky/454964087739a449da04`

La conformité au PEP8 peut être testée par un programme à télécharger comme suit :

```
sudo pip3 install pep8
```

Pour tester son programme web.py, on tapera :

```
cd ~/mesprogrammespythonAmoi
pep8 web.py | less
```

Le *pipe* avec *less* est pour le cas bien improbable où notre programme aurait beaucoup de divergences avec les préconisations PEP8.

LES MÉMENTO PYTHON sont nombreux. On trouve celui de Laurent Pointal [17], inclus dans l'ouvrage de Richard Cordeau signalé plus haut. Laurent Pointal fourni aussi beaucoup d'autres éléments sur son site [18].

L'avantage de ceux de L. Pointal est qu'ils concernent tous Python3 (et qu'ils sont en français !). Beaucoup d'autres sont relatifs à des versions antérieures [19] de Python.

LES FORMATIONS SUR LE WEB On trouve aussi des formations très claires [20] ainsi que des MOOC [21].

17. `https://perso.limsi.fr/pointal/_media/python:cours:mementopython3.pdf`
18. `https://perso.limsi.fr/pointal/python:accueil`
19. Comment identifier la bonne version ? Le marqueur le plus simple est que sous Python 3 la fonction print est une fonction et on la tapera *print()* tandis que pour les versions antérieures c'est une commande et on lira *print " version < 3 "*.
20. `https://openclassrooms.com/courses/apprenez-a-programmer-en-python`
21. Sur France Université Numérique (`https://www.fun-mooc.fr/`) les cours Python sont terminés à l'heure où ce document est rédigé. En anglais, on trouve de nombreux cours d'initiation ou de spécialisation sur Python sur le site `https://www.coursera.org/`.

*Premières utilisations de Python*

Sous Ubuntu, le téléchargement de Python[22] ne pose aucune difficulté.

22. Quand ce n'est pas précisé dans le document, la version à privilégier est python 3 et, plus précisément à l'heure actuelle, python 3.6.5

*Par quoi commencer ?* Lors de bidouillages, on reprend généralement le programme de quelqu'un d'autre. Ou bien on essaie d'exécuter une partie d'un programme.

Voyons à quoi pourrait ressembler un programme *vraiment* simple :

```
print("Welcome!")
g = input("Guess the number: ")
guess = int(g)
if guess == 5:
print("You win!")
else:
print("You lose!")
print("Game over!")
```

Si on connaît quelques mots d'anglais, a priori, on peut presque commenter ligne à ligne ce que fait le programme.

Deux ou trois choses sautent aux yeux :

— l'écriture est simple par principe, pas d'accolades dans tous les sens, ni de signe < qu'il faut apparier à un >

— l'affectation de variable se fait par le signe égal **=**

— le test d'égalité est réalisé avec un double égal **==**

— la réalisation du test **si** . . . **alors** est spécifique. Il faut noter le « : » en fin de ligne de test et l'indentation pour l'action à réaliser en fonction de l'embranchement du test. Ne pas réaliser ces deux éléments de syntaxe conduirait à un message d'erreur de l'interpréteur.

Mais comment faire pour exécuter un tel programme ? On peut profiter du fait que Python soit interprété pour l'exécuter ligne à ligne. Pour cela il suffit de lancer un terminal et de taper la commande **python3** puis d'entrer une à une les lignes de programme sans oublier l'indentation et le retour chariot (*enter*) à la fin de chaque ligne.

Pour sortir de l'interpréteur, il faut taper exit(). Voici donc ce que cela donne :

**Taper exit() pour sortir de l'interpréteur Python**

```
jose@debjo:~$ python3
Python 3.6.5rc1 (default, Mar 14 2018, 06:54:23)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> print("Welcome!")
Welcome!
>>> g = input("Guess the number: ")
Guess the number: 4
>>> guess = int(g)
>>> if guess == 5:
...     print("You win!")
... else:
...     print("You lose!")
...
You lose!
>>> exit()
jose@debjo:~$
```

*Comment travailler avec un premier programme ?* Taper quelques lignes de code pour avoir une réaction immédiate de l'interpréteur, on se rend bien compte que c'est insuffisant. On souhaite donc réunir ces lignes dans un programme et faire rejouer ce programme autant de fois que désiré, sans retaper à chaque fois les instructions.

LA SOLUTION LA PLUS SIMPLE EST D'UTILISER UN ÉDITEUR DE TEXTE, comme notepad++ sous Windows par exemple. Ce type d'éditeur de texte est suffisamment élaboré pour contenir des modules qui permettent une bonne mise en forme des programmes et mettent en valeur (par des couleurs, du gras, etc.) les verbes, variables, tests et autres éléments du langage traité. On parle dans ce cas de *coloration syntaxique*.

Sous Linux, on trouve des éditeurs de texte du plus sommaire (**nano** par exemple) au plus complexe et complet (**emacs** est considéré comme le *nec plus ultra*). Mais pour **emacs**, le ticket d'entrée est vraiment important même si le coût élevé se trouve justifié. On préférera un éditeur "tout terrain" comme **gedit**.

**Un bon éditeur de texte : gedit**

IL EST TEMPS D'ÉCRIRE SON PREMIER PROGRAMME en passant par un éditeur. On va entrer ces quelques lignes dans gedit

```
print("Bonjour")
nombre = input("Tape un nombre : ")
print("Perdu ! Désolé...")
```

On sauvegarde le fichier en l'appelant « test1 » par exemple. On ouvre un terminal et on lance notre programme test1 :

```
jose@debjo:~$ python3 test1
Bonjour
Tape un nombre : '
```

```
Perdu ! Désolé...
jose@debjo:~$
```

IL EST TEMPS D'OBTENIR LA COLORATION SYNTAXIQUE de son éditeur préféré. Maintenant on va renommer le programme en le suffixant par « .py » pour l'identifier comme un programme python. Cela ne va pas changer grand-chose à son exécution, mais si on le réédite dans gedit, nous verrons une belle coloration syntaxique.

Le suffixe .py permet la coloration syntaxique.

Pour utiliser ce programme, jusqu'à présent il était nécessaire de le lancer d'un terminal en tapant python3 suivi du nom du programme. Et cela ne marchait que lorsqu'on se trouvait dans la bibliothèque (*directory* pour les anglophones) où était sauvegardé ledit programme. La suite du document va expliquer comment, en quelques commandes, transformer notre fichier en un véritable programme exécutable par le système Linux.

IL EST POSSIBLE DE RENDRE SON PROGRAMME EXÉCUTABLE. Pour ne plus avoir à taper python3, on va rendre le fichier test1.py exécutable. Pour cela on tape dans le terminal :

La commande chmod permet de rendre le fichier exécutable.

```
chmod +x test1.py
```

Maintenant, on peut exécuter le fichier en tapant [23] :

```
./test1.py
```

23. On peut même ne taper que les premières lettres du nom de fichier et appuyer sur la barre de tabulation pour que le nom entier soit complété automatiquement.

Mais on s'aperçoit que le système ne sait pas l'exécuter correctement :

```
jose@debjo:~$ ./test1.py
./test1.py: ligne 1: erreur de syntaxe près du symbole inattendu « "Bonjour" »
./test1.py: ligne 1: ‘print("Bonjour")’
jose@debjo:~$
```

En fait, pour que le système sache comment exécuter le fichier, il va falloir le lui indiquer sur la première ligne du programme. On va en profiter pour préciser aussi quel codage de caractères on utilise, afin d'être vraiment portable.

Les deux premières lignes de tout programme python sous Linux

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
print("Bonjour")
nombre = input("Tape un nombre : ")
print("Perdu ! Désolé...")
```

On peut maintenant relancer le programme en tapant./test1.py dans le terminal et vérifier qu'il s'exécute tout à fait correctement.

Enfin, il reste à ne plus être tenu de se positionner dans le répertoire du programme [24].

Pour cela il faut indiquer au système d'exploitation quels chemins il doit suivre pour aller chercher les programmes à exécuter. Voyons déjà ceux qu'il a par défaut (les chemins sont séparés par « : » comme ci-dessous) avec la commande *echo* :

Mettre le programme sur le bon chemin (PATH)

```
jose@debjo:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
jose@debjo:~$
```

Par convention, sous Linux, **il est préconisé d'appeler ses répertoires d'exécutables « bin »**. Dans mon cas, je veux mettre mes programmes dans

/home/jose/Documents/2018AcquisitionDonnees/bin Voyons donc tout ça :

```
jose@debjo:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
($HOME est une autre variable système)
jose@debjo:~$ echo $HOME
/home/jose
(maintenant la commande qui tue)
jose@debjo:~$ export PATH=$PATH:$HOME/Documents/2018AcquisitionDonnees/bin
(on vérifie)
jose@debjo:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/home/jose/Documents/
2018AcquisitionDonnees/bin
jose@debjo:~$
```

Le programme test1.py étant dans le répertoire susdit, même si je suis sur mon répertoire de base [25], le système le reconnaîtra :

```
jose@debjo:~$ test1.py
Bonjour
Tape un nombre : 659302
Perdu ! Désolé...
jose@debjo:~$
```

Hélas, ce chemin ne durera que le temps que dure la session de travail, comme dit le poète.

COMMENT FAIRE POUR NE PAS S'EMBÊTER AVEC LE PATH. Une astuce permet de ne pas avoir à tripatouiller les fichiers cachés, dans Ubuntu et toute la famille Debian [26], pour rendre le chemin vers nos exécutables permanent.

24. Et donc ne plus être tenu de faire précéder le nom du programme de « ./ » pour l'exécuter.

25. Petite précision, sur Linux le répertoire par défaut de l'utilisateur est /home/monidentifiant Lorsqu'on lance une commande de changement de répertoire (commande « cd » pour *change directory*) il suffit de taper
cd ~/Documents
et le système l'interprète en /home/monidentifiant/Documents

26. Et donc sans doute Mint

Il existe en effet un répertoire par défaut [27] pour les programmes que l'on écrit soit même.

MAIS CE RÉPERTOIRE, IL FAUT LE CRÉER. Pour ce faire, soit on passe par son gestionnaire de fichiers [28] qui permet facilement cette opération, soit on passe courageusement par le terminal et on tape :

```
mkdir bin
```

On a ainsi créé le répertoire « bin » sous notre répertoire principal. Lors de notre prochaine connexion, ce nouveau répertoire apparaîtra automatiquement dans le chemin des répertoires exécutables.

Il suffit donc d'y mettre tous nos programmes exécutables pour qu'ils puissent être lancés dès qu'on tape leur nom dans un terminal.

*Et voilà*, comme disent les Anglo-saxons.

*Utiliser un environnement de développement* Au-delà des éditeurs, on peut utiliser des environnements de développement. Sous Windows, il y a l'excellent **pyscripter**.

Sous Linux, on peut utiliser un gestionnaire simple, comme **thonny**. Cependant le gestionnaire **geany** est bien plus complet tout en restant facile d'abord.

D'autres existent (eclipse d'IBM, SPE, Eric, etc.) mais sont plutôt recommandés pour des utilisations intensives et professionnelles.

**Un bon environnement de développement : geany**

27. Par répertoire par défaut, on entend un répertoire qui est inscrit dans le chemin des répertoires d'exécutable dès le lancement de la session de travail Linux.

28. Sous Linux, il y a foultitude de gestionnaires de fichiers. Sous l'environnement XFCE utilisé par XUBUNTU, le gestionnaire par défaut est *Thunar*, sous Gnome c'est *Nautilus*, etc. Les nostalgiques utilisent *midnight commander* qui se lance à partir d'un terminal virtuel en tapant *mc*.