PP1-0: L'environnement de travail

Amélie Ledein & Gabriel Hondet
`nom (at) lsv (dot) fr`

September 16, 2020

Cette session se distingue des autres par son objectif: vous fournir les outils nécessaires pour pouvoir travailler dans de bonne conditions.

L'environnement de travail se décompose en plusieurs outils qu'il sera nécessaire d'utiliser durant votre scolarité, et probablement dans votre vie professionnelle. Notez qu'il n'est possible (ni utile) de maîtriser complètement ces outils. Votre apprentissage se fera progressivement au cours de vos études.

Ce TP exposera les outils utilisés généralement pour le développement (que ce soit la programmation ou l'édition de fichiers LaTeX). Il est important de noter que toutes les indications sont subjectives, et découlent de l'expérience des auteur·e·s. Les préceptes énoncés ne constituent en aucun cas une "approche pour les gouverner toutes".

Nous commencerons par introduire la ligne de commande, puis les éditeurs de texte et enfin les outils spécifiques à OCaml.

1 Communiquer avec un système d'exploitation via la ligne de commande

Sur les systèmes d'exploitation Il existe principalement 3 systèmes d'exploitation: GNU/Linux, Unix et Windows (par Microsoft). Si Windows est "populaire", il est assez peu adapté au développement (en plus d'être à sources fermées). Il est *vivement recommandé de ne pas l'utiliser*. Concernant le choix GNU/Linux v. BSD, la question est idéologique et pratique. D'une part, les licences BSD sont libérales: elles offrent beaucoup de libertés aux utilisateur·rices du programme[1] et sont non ambiguës par leur simplicité (on dit aussi qu'elles sont à *copyleft faible*). De l'autre côté du spectre, les licences GPL (GNU Public License) sont dites à *copyleft fort* et imposent aux développeur·euses qui utilisent des programmes sous cette licence à publier eux·elles même sous cette licence (ou équivalent).

D'un point de vue pratique, GNU/Linux n'est qu'un noyau qui ne se suffit pas à lui même, c'est pourquoi on parle de *distributions* GNU/Linux: une distribution est une collection de programmes (dont un gestionnaire de paquets) qui utilisent le noyau. Les systèmes BSD sont constitués du noyau et d'un ensemble de programmes s'interfaçant avec celui-ci. Le système résultant est en général plus uniforme. Les lecteur·trices intéressé·es et hésitant entre les deux systèmes peuvent se rapporter à `http://www.over-yonder.net/~fullermd/rants/bsd4linux/01`.

La ligne de commande C'est le principal medium de communication avec le système d'exploitation. Si elle est moins intuitive que les interfaces graphiques, elle reste le moyen privilégié pour utiliser les systèmes d'exploitation GNU/Linux ou Unix par un grand nombre de personne pour son efficacité et sa versatilité. Ce qu'on appelle ligne de commande décrit principalement le mode d'interaction textuel. Le programme se chargeant de cette interaction est appelé *shell* (la coquille du noyau).

Pratiquons

- Le *shell* est accessible via un émulateur de terminal. Pour en lancer un, cherchez `xterm`, `rxvt` ou `Terminal`. Vous devriez vous retrouver devant une fenêtre noire avec une inscription de la forme

  ```
  user@host:~ $$
  ```

[1] voir la licence à 3 clauses BSD

Cette inscription est appelée *prompt* et est configurable. Elle apparaît à chaque nouvelle commande.

- Les commandes sont tapées au clavier, à la suite du *prompt*. Par exemple, pour afficher le répertoire courant, entrez `pwd`.
- Voici quelques commandes capitales,
  - `ls` liste le contenu d'un dossier (*list*)
  - `cd D` change le répertoire courant pour `D` (*change directory*);
  - `mkdir D` crée le dossier `D` (*maked directory*);
  - `rm F` supprime le fichier (ou dossier) `F` (*remove*);
  - `mv F` déplace le fichier (ou dossier) `F` (*move*);
  - `cat F` affiche le contenu du fichier `F` dans le terminal (*concatenate*);
  - `man C` affiche la page de manuel concernant `C` (*manual*).
- Les systèmes d'exploitations Unix et GNU/Linux sont assez bien documentés (OpenBSD l'est particulièrement bien). Les principales commandes concernant la documentation sont
  - `apropos S` cherche les pages de manuel qui peuvent parler de `S`;
  - `whatis S` affiche le synopsis de la page de manuel de `S`;
  - `man S` affiche la page de manuel de `S`. On pourra apprendre à lire le manuel avec `man man`.

  De plus, la plupart des programmes proposent une aide succinte via l'option `--help` (vous pouvez essayer `ls --help`).

Notez que les `shell` usuels proposent l'autocomplétion, qui permet de compléter les commandes ou les noms de fichiers. Par exemple, entrez `cd Doc` et appuyez sur la touche tabulation: `Doc` est complété par `Documents`.

Exercice Utilisez les commandes ci-dessus pour créer une arboresscence de fichiers dans laquelle vous sauvegarderez les fichiers générés pour chaque TP.

2 L'éditeur de texte

C'est probablement le programme qui vous servira le plus pour le reste de votre vie. Il faut distinguer deux paradigmes d'édition. Le plus habituel est celui utilisé par les logiciels de traitement de texte: on entre un caractère en appuyant sur la touche du clavier correspondant. Par opposition, dans les éditeurs modaux, la touche "a" du clavier n'écrit pas le caractère "a" dans le fichier: elle correspond à une commande de l'éditeur. Les éditeurs modaux disposent d'une touche qui permet de passer en mode d'insertion, dans lequel on retrouve un fonctionnement classique.

L'édition modale est un peu plus compliquée à appréhender, mais finit en général par être plus agréable: les commandes de manipulation de texte sont plus accessibles. Si fichier n'est écrit qu'une seule fois, il est édité une infinité.

- Emacs et VsCode (développé par Microsoft, mais à sources ouvertes) appartiennent à la première catégorie.
- Vim, NeoVim et Kakoune appartiennent à la deuxième.

On notera qu'on peut rendre Emacs modal, via le mode Evil[2]

En général, le choix se fait surtout entre Emacs et Vim. Ces deux éditeurs ont les avantages d'être très répandus et extrêmement personnalisables.

On peut aussi faire basculer le *shell* en édition modale, avec `set -o vi`.

[2]Ce fichier est édité avec la distribution Doom Emacs qui embarque le mode Evil

Emacs : Pour apprendre à utiliser EMACS (en mode classique), vous pouvez lire le tutoriel (ce qui prend environ 30mn). Pour cela entrez la commande `$ emacs` puis appuyer sur la touche `F1` et ensuite la touche `t`. Cela lancera le tutoriel pour vous, et vous n'aurez plus qu'à le lire. Vous pouvez aussi jeter un coup d'oeil au document suivant http://www.jesshamrick.com/2012/09/10/absolute-beginners-guide-to-emacs/.

Vim : Un tutoriel est également disponible dans VIM; saisir dans le *shell* `$ vimtutor` voire même `$ vimtutor fr` . Le site http://www.openvim.com/ propose un tutoriel interactif.

3 Compiler du OCaml

La compilation est un processus qui transforme un fichier écrit dans un langage A en un autre fichier écrit dans un langage B. La machine que vous utilisez ne reconnaît qu'un seul langage qui est un langage binaire. Lorsque vous voulez que votre machine éxécute du code OCAML, vous allez donc devoir compiler votre fichier écrit en OCAML vers un fichier qui contient du code binaire. Ce programme est appelé `ocamlc`.
Question 0 : Créer un fichier vide avec une extension `.ml` et compilez le. Il est possible d'exécuter le programme généré avec la commande
`$ ./a.out`

Comment nommer le fichier de sortie différemment que `a.out`?

Entrez le contenu suivant dans votre fichier:

```
let _ = Printf.printf "Hello World!\n";
```

et compilez-le puis exécutez-le.

4 Programmer en OCaml

Les éditeurs de textes mentionnés sont tous très extensibles. Si bien que la plupart des langages de programmation disposent d'un mode pour aider à développer. Ces modes proposent en général de la coloration syntaxique, de l'indentation automatique, de la complétion, &c.

OCAML est un langage fortement typé, ce qui permet d'éviter un grand nombre d'erreurs de programmation. En temps normal, il faut appeler le compilateur à chaque fois que l'on veut vérifier le code écrit. Ce qui devient vite fastidieux.

Le programme MERLIN permet aux éditeurs de vérifier le types des expressions entrées dans le fichier, sans avoir à le compiler. Il permet en outre de donner le type d'une expression à la volée.

Il est possible de l'installer avec OPAM, le gestionnaire de paquets pour OCAML. Il est normalement installé, mais il faut le configurer,

1. lancez `$ opam init`

2. Opam devrait proposer d'éditer le `.bashrc` ou `.profile` pour établir les variables nécessaires à OCaml au démarrage. Vérifiez que l'un des deux fichiers contienne une ligne du genre

   ```
   #OPAM configuration
   . $HOME/.opam/opam-init/init.sh > /dev/null 2> /dev/null || true
   ```

   Si ce n'est pas le cas, ajoutez ces lignes au `.profile`

3. Étant donné que `.profile` est lu à l'ouverture de la session, pour charger l'environnement OCaml, il faut ou bien vous déloger et reloger, ou bien charger le `.profile` via `$ . /.profile` ou bien évaluer la configuration que propose opam `$ eval $(opam config env)`

4. Pour s'assurer qu'opam est bien détecté, on peut vérifier que la variable d'environnement `PATH` contient un chemin vers opam. Pour se faire, il faut vérifier que la sortie de la commande `$ echo $PATH` contienne une entrée contenant un `.opam`.

5. On peut maintenant installer les paquets nécessaires, avec `opam install tuareg merlin`.

6. La configuration de l'éditeur peut-être gérée par OPAM avec `opam user-setup install`.

5 Ressources diverses

Cette section est à regarder après avoir fini ce TP.

Dans le prochain TP, on va attaquer la programmation en OCaml. Il se peut qu'une minorité d'entre-vous n'ayant pas fait de CAML-LIGHT soit désemparée face à ce nouveau langage. La toile regorge de ressources diverses et variées pour apprendre OCaml. Je vous invite donc à lire les différentes ressources sur OCaml qui sont citées ci-dessous (ou d'autres selon vos goûts).

5.1 Premiers pas en OCaml

Pour ceux qui découvrent le langage OCaml, je vous invite à commencer par lire les 8 premiers chapitres de ce tutoriel http://mirror.ocamlcore.org/ocaml-tutorial.org/. Sinon, un rapide résumé a été fait par Sylvain Schmitz et est accessible à l'adresse suivante : http://www.lsv.ens-cachan.fr/~schmitz/teach/2012_prog/part1/td1_ocaml/caml.pdf. Une partie des exercices de cette feuille seront repris dans le prochaine TP.

5.2 Aller plus loin avec la ligne de commande

A l'adresse suivante : http://overthewire.org/wargames/bandit/, vous trouverez un petit jeu découpé en niveau qui vous demande de trouver la bonne commande pour passer au niveau suivant. Cela vous fera un bon entraînement pour apprendre à lire le manuel des commandes BASH. Attention, certains niveaux peuvent être corsés.

BASH est en réalité un langage de programmation. Il est possible d'écrire des scripts en BASH. Pour apprendre à programmer en BASH, vous pouvez par exemple lire les premières pages du manuel officiel : https://www.gnu.org/software/bash/manual/bash.pdf.

Comme la ligne de command est le moyen standard d'interagir avec le système, il existe une myriade d'outils pour la modifier. Par exemple, si bash vous est trop austère, essayez fish (friendly interactive shell) ou zsh. `fzf` permet d'avoir une compétion interactive plus "puissante". `fasd` permet de changer de naviguer entre les fichiers en utilisant des raccourcis ou en prenant en compte la fréquence de visite.

5.3 Aller plus loin avec Emacs

- Le manuel d'emacs : https://www.gnu.org/software/emacs/manual/pdf/emacs.pdf
- Une introduction à emacs-lisp : https://www.gnu.org/software/emacs/manual/pdf/eintr.pdf
- Une *cheatsheet* des raccourcis claviers : https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf

5.4 Aller plus loin avec Vim

- Usez et abusez de `:help foo`
- http://vim.rtorr.com ou http://vimsheet.com pour les keybindings
- une cheatsheet pour le vimscript (langage pour configurer vim) https://devhints.io/vimscript