

INF3135

Construction et maintenance de logiciels

Cours 1: Présentation du cours

Alexandre Blondin Massé

Université du Québec à Montréal
Département d'informatique

Été 2020

Table des matières

Présentation du cours

Environnement Unix

Environnement de développement

Le langage C

Logiciel de contrôle de versions

Présentation du cours

Informations générales

- **Trimestre:** Été 2020
- **Titre du cours:** Construction et maintenance de logiciels
- **Sigle:** INF3135
- **Département:** Informatique
- **Enseignant:** Alexandre Blondin Massé
- **Courriel:** blondin_masse.alexandre@uqam.ca
- **Site personnel:** <http://lacim.uqam.ca/~blondin>
- **Bureau:** PK-4525
- **Coordonnateur:** Alexandre Blondin Massé
- **Site du cours:** <http://lacim.uqam.ca/~blondin/fr/inf3135>
- **Plan de cours:** [cliquer ici](#)

Description officielle (site de l'UQAM):

« Notions de base de la programmation procédurale et impérative en langage C sous environnement Unix/Linux (définition et déclaration, portée et durée de vie, fichier d'interface, structures de contrôle, unités de programme et passage des paramètres, macros, compilation conditionnelle). Décomposition en modules et caractéristiques facilitant les modifications (cohésion et couplage, encapsulation et dissimulation de l'information, décomposition fonctionnelle). Style de programmation (conventions, documentation interne, gabarits). Débogage de programmes (erreurs typiques, traces, outils). Assertions et conception par contrats. Tests (unitaires, intégration, d'acceptation, boîte noire vs. boîte blanche, mesures de couverture, outils d'exécution automatique des tests). Évaluation et amélioration des performances (profils d'exécution, améliorations asymptotiques vs. optimisations, outils). Techniques et outils de base pour la gestion de la configuration. Système de contrôle de version. »

Objectifs du cours

- Maîtriser le développement d'un programme dans un **environnement Unix**
- Apprendre les bases du **langage C**
- **Documenter** convenablement un projet écrit en C
- Gérer la **construction** d'un programme C
- Gérer adéquatement **l'historique** d'un programme
- Apporter des **modifications** à un système développé par d'autres personnes
- Apprendre à utiliser des **bibliothèques C**
- Fournir une **couverture de tests** d'un programme
- Mettre en place des mécanismes d'**intégration continue** et de **déploiement continu**
- Travailler en **équipe** sur la construction d'un programme
- Bien **communiquer** par écrit et à l'oral ses idées à propos du développement d'un programme

Références

Contenu du cours

- *The C Programming Language*, de Kernighan et Ritchie
- *Manuel d'utilisation de Git*: [disponible en ligne](#)
- *Documentation de Make*: [disponible en ligne](#)
- *Documentation de Markdown*: [documentation officielle](#) et [spécialisation pour GitLab](#)

Politiques de l'UQAM

- Règlement 18 sur la tricherie et l'intégrité académique (plagiat): <http://r18.uqam.ca/>.
- Politique 16 contre le harcèlement sexuel: [document pdf](#).

Ressources

Laboratoires

- **Description des labos:** dépôt GitLab
- **Démonstrateurs:** Jocelyn Bédard et Elyes Bejaoui
- **24 juin et 1er juillet:** pas de labo → cours plus long

Liens importants

- **Théorie:** capsules vidéos à visionner **avant** les cours
- **Matériel:** groupe GitLab du cours
- **Support en ligne:** équipe Mattermost plutôt que canal Slack
- **Quiz:** sur Moodle

Modalités d'évaluation

Travaux pratiques (60%)

- **TP1 (seul)**: initiation au langage C (20%)
- **TP2 (seul)**: maintenance d'un programme (20%)
- **TP3 (seul ou équipe)**: construction d'un programme (20%)

Quiz (10% chacun, 40% au total)

- **Quiz 1**: 26 mai
- **Quiz 2**: 23 juin
- **Quiz 3**: 14 juillet
- **Quiz 4**: 11 août

Double seuil

Exceptionnellement, pas de double seuil

Environnement Unix

Terminal et shell

- **Terminal** = interface
- **Shell** = interpréteur de commandes

Interaction

Intermédiaire entre

- l'**utilisateur**,
- le système de **fichiers**
- les **applications**

Particulièrement utile pour des connexions à **distance**

INF1070 Utilisation et administration des systèmes informatiques

- Introduit comme cours **obligatoire** à l'automne 2018
- Répond à un **besoin** des enseignants, des étudiants et de l'industrie
- Permet d'**apprivoiser** la ligne de commande

Commandes de base

- `echo` — afficher un message
- `ls` — lister le contenu du répertoire
- `cat` — afficher un fichier
- `cd` — changer de répertoire
- `exit` — fermer le shell

Aller chercher de l'aide

```
$ man ls
```

```
$ man cat
```

```
$ man man
```

Afficher le contenu de fichiers

- **head** — Affiche les première lignes
- **tail** — Affiche les dernières lignes
- **less** (et **more**) — Affiche le fichier page par page
- **tac** — Affiche un fichier en commençant par la dernière ligne
- **rev** — Inverse chacune des lignes affichée
- **wc** — Affiche le nombre de lignes, mots et octets

Exemples

```
$ wc -l /usr/share/common-licenses/GPL-3
$ head -n 2 /usr/share/common-licenses/GPL-3
$ less /usr/share/common-licenses/GPL-3
```

Répertoires et chemins

- `tree` — affiche l'arborescence d'un répertoire
- `pwd` — affiche le chemin du répertoire courant
- `mv` — déplace/renomme des fichiers (ou répertoires)
- `cp` — copie (ou écrase) des fichiers
- `rm` — supprime des fichiers
- `mkdir` — crée des répertoires (vides)
- `rmdir` — supprime des répertoires vides
- `grep` — cherche les lignes correspondant à un motif dans un fichier
- `find` — rechercher des fichiers dans une hiérarchie de répertoires

Exemple

Suppression d'un répertoire **non vide**:

```
$ rm -rf nom-du-repertoire
```

Droits et utilisateurs

- `chmod` — modifie les droits d'un fichier (ou répertoire)
- `chown` — modifie l'utilisateur propriétaire d'un fichier
- `chgrp` — modifie le groupe propriétaire d'un fichier
- `sudo` — exécute une commande sous un autre utilisateur
- `su` — change d'utilisateur

Exemples

```
$ chmod +x nom-du-script  
$ ls -l  
$ sudo apt install pandoc
```

Autres commandes utiles

- **file** — affiche le type de fichier
- **date** — affiche et configure la date
- **touch** — modifie l'horodatage d'un fichier ou crée un fichier vide
- **stat** — affiche différentes informations sur un fichier

Exemples

```
$ date  
$ file 01-intro.pdf hello.c
```


Gestionnaire de paquets

- Tâche courante: **installer** des logiciels/bibliothèques
- Gestion des **dépendances** entre paquets est complexe
- **Solution**: utiliser un **gestionnaire de paquets**
- En anglais, *package manager*

Exemples

- **Aptitude** (apt): Debian et ses dérivés
- **Pacman** (ArchLinux)
- **RPM** (Red Hat, Fedora, CentOS)
- **MacPorts** (MacOS)
- **Homebrew** (MacOS), etc.

Système Unix

Linux

Idéal

MacOS

Ça va pour le cours, mais attention aux **différences**

Windows

- **[Recommandé]** Installer une distribution **Linux** en partition double ou comme unique système (**Ubuntu**, **Linux Mint**)
- **[Alternative]** Installer une **machine virtuelle** (VirtualBox, VMWare)
- **Linux Subsystem** et autres variantes suffit pour le TP1, mais pas pour les deux autres TP
- Mieux vaut en profiter **dès le début** pour s'habituer à Linux

Environnement de développement

Environnement de développement

- Outil de **base** d'un programmeur
- En anglais, *IDE* = *integrated development environment*
- Quelques exemples:



- Pourtant, de nombreux programmeurs avancés **préfèrent un simple éditeur de texte**
- Certains plaident même que **Unix est un EDD**

Éditeur de texte

Offre très variée

- Notepad/**Notepad++** (Windows)
- TextEdit (MacOS)
- **Gedit** (GNOME), souvent installé par défaut
- **SublimeText** (multiplateforme)
- **Visual Studio Code**
- **Emacs**
- **Vi/Vim** et ses dérivés (multiplateforme) → **préfééré**

Éditeur de texte en ligne de commande

- **Nécessaire** pour certaines manipulations
- Édition de **dialogues** Git (lors d'un rebase **interactif**)
- Modifications de fichiers **distants**

Vim

- Un des plus **anciens** éditeurs de texte
- Un des éditeurs de texte les **plus utilisés**
- Son ancêtre, `vi`, a été créé par Bill Joy en **1976**
- Le nom *Vim* vient de *Vi iMproved*
- Multiplateforme (Linux, MacOS, Windows)

Caractéristiques

- Très **mature**
- Interaction **directe** avec le terminal
- Installé par défaut avec plusieurs distributions Unix
- **Rapide**, en particulier pour la programmation à distance
- Hautement **configurable**
- Orienté **clavier** (GVim permet une utilisation limitée de la souris)
- Courbe d'apprentissage **difficile** pour les débutants
- Une bonne **configuration** est importante!

Le langage C

Bref historique

- **Années 70**: Naissance du langage, créé par **Ritchie** et **Kernighan**
- Origine du langage fortement liée à celle d'**Unix** → 90% du système Unix écrit en C
- **1978**: Publication du livre « The C Programming Language », par Kernighan et Ritchie
- **1983**: ANSI forme un comité pour **normaliser** le langage
- **1989** Apparition de la norme **ANSI-C**
- **1999**: Révision du standard (ISO C99)
- **2011**: Révision du standard (ISO C11)

Caractéristiques du langage

- **Bas niveau**: près du langage machine, contrôle fin de la mémoire, très efficace
- **Déclaratif**: le type des variables doit être déclaré
- **Flexible**: espacement, indentation
- **Structuré**: organisé en blocs (accolades)
- **Modulaire**: division en fichiers, compilation séparée
- **Flexible**: peu de vérification, pointeurs typés mais non contrôlés
- **Spécifique**: pour bien faire une tâche, adapté aux petits programmes et aux bibliothèques
- **Portable**: mais avec parfois un certain effort
- **Simple**: spécification assez courte
- **Verbeux**: il faut écrire beaucoup de code

Exemple

Fichier maj.c:

```
#include <stdio.h>
#include <ctype.h>

int main(void) {
    char c;
    while ((c = getchar()) != EOF) {
        putchar(toupper(c));
    }
    return 0;
}
```

Question

Que fait ce programme?

Cycle de compilation

Édition du programme source (.c)

À l'aide d'un éditeur de texte ou d'un EDD

Compilation (.c \rightarrow .o)

- Indique les erreurs de **syntaxe**
- **Ignore** les fonctions et les bibliothèques invoquées

Édition de liens (*linking*)

- Fichiers .o assemblés pour former un binaire exécutable
- **Unix**: extension .out (par défaut), **Windows**: extension .exe

Exécution du programme

Invocation: ./<nom exécutable>

Compilation « directe »

- Combinaison de la **compilation** et l'**édition des liens**
- Produit par **défaut** un exécutable nommé `a.out`:

```
$ gcc maj.c  
$ ls -l a.out
```

- **Invocation** de l'exécutable:

```
$ ./a.out
```

- On peut **nommer** l'exécutable avec l'option `-o`:

```
$ gcc -o maj maj.c  
$ ./maj
```

Compilation et édition des liens

- Parfois préférable de **séparer** compilation et édition des liens
- Projets de plus **grande envergure**
- On compile **chaque fichier** séparément
- Par **défaut** l'extension `.c` devient `.o`

```
$ gcc -c maj.c
```

```
# Équivalent
```

```
$ gcc -o maj.o -c maj.c
```

- Puis on produit l'**exécutable**:

```
$ gcc -o maj maj.o
```

- Et on **invoque** l'exécutable:

```
$ ./maj
```

Logiciel de contrôle de versions

Logiciel de contrôle de versions

- Permet de stocker un ensemble de **fichiers**
- Conserve en mémoire la **chronologie** de toutes les modifications effectuées
- Permet de **partager** des fichiers entre plusieurs personnes
- Permet de conserver différentes **versions** du code source d'un projet
- Permet également de gérer des versions **parallèles** (branches)
- Garantit l'**intégrité** des fichiers

Liste de logiciels connus

Nom	Type	Accès
Bazaar	distribué	libre
BitKeeper	distribué	propriétaire
CVS	centralisé	libre
Darcs	distribué	libre
Git	distribué	libre
Mercurial	distribué	libre
Subversion	centralisé	libre

- Dans ce cours, nous utiliserons **Git**
- **Obligatoire** pour la remise des travaux

Naissance de Git

Dates

- **2002**: Linus Torvalds utilise BitKeeper pour versionner Linux
- **6 avril 2005**: La version **gratuite** de BitKeeper est supprimée
- Torvalds décide de créer son **propre** logiciel: **Git**
- **18 avril 2005**: Git supporte l'opération de fusion de fichiers
- **16 juin 2005**: Utilisé pour conserver l'historique de Linux
- **Fin juillet 2005**: Junio Hamano devient le développeur principal

Nom - Extrait de Wikipedia

« Git » is British English slang for an unpleasant person.

*« I'm an egotistical bastard, and I name all my projects after myself.
First Linux, now git. »*

— Linus Torvalds

Commandes les plus courantes

- **Créer** un nouveau projet: `git init`
- **Cloner** un projet existant: `git clone`
- **Sauvegarder** l'état courant du projet: `git commit`
- **Versionner** un nouveau fichier: `git add`
- **Ajouter** un fichier pour le prochain commit: `git add`
- **Consulter** l'historique: `git log`
- **Récupérer** des changements à distance: `git pull`
- **Téléverser** des changements à distance: `git push`

Apprendre les commandes

- Plusieurs commandes vues en **laboratoire**
- Vous devrez aussi en apprendre par **vous-même**
- Beaucoup d'**options** sont disponibles:

```
$ git remote --help
```

Hébergement de dépôts Git

La plupart des commandes Git se font de façon **locale**

- Chaque clone d'un dépôt est **autonome**
- L'historique est **purement local**

Cependant, il est pratique de pouvoir **partager** nos modifications

Pour cela, il existe des sites dédiés à l'**hébergement** de tels projets:

- Github
- Bitbucket
- GitLab

Dans ce cours, vous devrez utiliser le **GitLab** du département d'informatique