

# INF3135

## Construction et maintenance de logiciels

Alexandre Terrasa

Département d'informatique  
Université du Québec à Montréal

7 septembre 2017  
Chapitre 1 : Introduction

D'après les notes de cours d'Alexandre Blondin Massé

# Table des matières

1. Présentation du cours
2. Environnement Unix
3. Environnements de développement
4. Le format Markdown
5. Le logiciel Git

# Table des matières

1. Présentation du cours
2. Environnement Unix
3. Environnements de développement
4. Le format Markdown
5. Le logiciel Git

# Informations générales

Trimestre	Automne 2017
Titre du cours	Construction et maintenance de logiciels
Sigle	INF3135
Horaire	Jeudi, de 13h30 à 16h30
Salle	PK-6605
Département	Informatique
Enseignant	Alexandre Terrasa, chargé de cours
Coordonnateur	Alexandre Blondin Massé, professeur
Courriel	<a href="mailto:terrasa.alexandre@uqam.ca">terrasa.alexandre@uqam.ca</a>
Page web	<a href="http://info.uqam.ca/~terrasa">http://info.uqam.ca/~terrasa</a>

## Description du cours (1/3)

- ▶ Initier les étudiants à la programmation à l'aide d'un **langage impératif et procédural**.
- ▶ Familiariser les étudiants à la **construction** professionnelle de logiciels et à leur **maintenance**.
- ▶ 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).

## Description du cours (2/3)

- ▶ 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, par ex., gdb).
- ▶ **Assertions** et conception par contrats.
- ▶ **Tests** (unitaires, intégration, d'acceptation, boîte noire vs. boîte blanche, mesures de couverture, outils d'exécution automatique des tests, par exemple, xUnit, scripts).

- ▶ Évaluation et amélioration des **performances** (profils d'exécution, améliorations asymptotiques vs. optimisations, outils).
- ▶ Techniques et outils de base pour la **gestion de la configuration** (par exemple, **make**, **cvs**).
- ▶ Introduction à la **maintenance** de logiciels (types de maintenance, techniques de base, par exemple, remodelage, automatisation des tests de régression).

# Modalités d'évaluation

- ▶ Deux **examens** :
  - ▶ Examen **intra [30%]**;
  - ▶ Examen **final [30%]**;
- ▶ Trois **travaux pratiques** :
  - ▶ TP1 : initiation à C **[10%]**;
  - ▶ TP2 : maintenance d'un logiciel **[10%]**;
  - ▶ TP3 : construction d'un logiciel **[20%]**;
- ▶ **Retard, absence et plagiat** : voir le plan de cours  
[http://info.uqam.ca/~terrassa/cours/INF3135/172\\_INF3135.pdf](http://info.uqam.ca/~terrassa/cours/INF3135/172_INF3135.pdf).



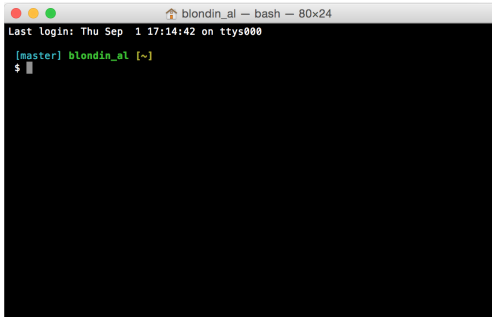
- ▶ **Langage C** : **The C Programming Language**, de Kernighan et Ritchie (disponible à la coop);
- ▶ **Git** : **Git Pro Book**;
- ▶ **Makefile** : **Documentation officielle**;
- ▶ **Markdown** : **Documentation officielle** et **spécialisée pour GitLab**;
- ▶ **StackOverflow**, mais méfiez-vous...

# Table des matières

1. Présentation du cours
2. Environnement Unix
3. Environnements de développement
4. Le format Markdown
5. Le logiciel Git

# La console

- ▶ Le **terminal** (ou **console**) est un outil **essentiel** pour le développeur;
- ▶ Il agit comme une **interface** entre le développeur et les programmes d'un système d'exploitation **Unix**;
- ▶ Il est utilisé pour **exécuter** des programmes et **manipuler** les fichiers.



```
blondin_al - bash - 80x24
Last login: Thu Sep 1 17:14:42 on ttys000
[master] blondin_al [~]
$
```

# Les commandes

- ▶ Le terminal sert à **exécuter** des commandes;
  - ▶ Les commandes sont des **programmes**;
  - ▶ **Entrées**: arguments, entrée standard, fichiers...
  - ▶ **Sorties**: sortie standard, sortie d'erreur, fichiers...

```
date -u  
Thu Sep 7 13:30:00 UTC 2017
```

- ▶ Résumé (**cheatsheet**) :  
<https://ubuntudanmark.dk/filer/fwunixref.pdf>

# Commandes agissant sur les fichiers

► Commandes **fréquentes** :

Commande	Description
<code>pwd</code>	Affiche le répertoire courant
<code>ls</code>	Liste les fichiers dans un répertoire
<code>ls -alh</code>	Liste plus détaillée
<code>cd</code>	Change de répertoire
<code>mkdir</code>	Crée un répertoire
<code>rm</code>	Supprime un fichier (irréversible)
<code>rm -rf</code>	Supprime un répertoire (récursivement)
<code>cp</code>	Copie un fichier
<code>cp -r</code>	Copie un dossier (récursivement)
<code>mv</code>	Renomme/déplace un fichier/répertoire

## Autres commandes de fichiers

Commande	Description
find	Trouve toutes les occurrences d'un fichier
cat	Affiche le contenu d'un fichier ou concatène plusieurs fichiers
less	Permet de parcourir le contenu d'un fichier
head	Affiche les premières lignes d'un fichier
tail	Affiche les dernières lignes d'un fichier
pwd	Affiche le répertoire courant
touch	Crée un fichier vide ou s'il existe, modifie sa date au moment présent

Commande	Description
top	Affiche les processus actifs
kill	Tue un processus (fin de tâche)
chmod	Change les permissions d'un fichier ou un dossier
which	Affiche le chemin d'une application
grep	Recherche une expression régulière
date	Affiche l'heure et la date

Commande	Description
ping	Vérifie si un hôte est disponible
whois	Affiche de l'information sur un nom de domaine
dig	Affiche l'information DNS d'un nom de domaine
wget	Télécharge un fichier
curl	Télécharge un fichier
ssh	Ouvre une session vers un serveur SSH



- La commande `man` affiche la documentation d'une commande:

```
man ls
man touch
man ssh
man man
```

- L'option `--help` (ou `-h`) affiche l'aide d'une commande:

```
ls --help
touch -h
```

# Exemples d'utilisation

- ▶ Tous les **résultats** de ces commandes peuvent être **redirigés** dans un fichier. Par exemple

```
ls > contenu-repertoire.txt
```

- ▶ `touch` est utile si on veut forcer une **recompilation** avec un Makefile par exemple.

- ▶ Exemples classiques de changement de **permissions** :

```
chmod 777 # rwx pour tous  
chmod 755 # rwx pour user,  
          # rx pour groupe et tous
```

- ▶ Utilisées en combinaison avec d'autres commandes, on peut écrire des **scripts** simples effectuant des tâches relativement complexes.

# Installation de logiciels

- ▶ Un développeur doit très souvent **installer** des logiciels sur une machine;
- ▶ Cela peut rapidement devenir **complexe**, surtout lorsque certains logiciels **dépendent** d'autres logiciels, en particulier dans les systèmes **Unix**;
- ▶ Heureusement, il existe un type de programme appelé **gestionnaire de paquets** (en anglais, *package manager*), qui facilite le processus :
  - ▶ Linux : **Aptitude**, **Pacman**, etc.;
  - ▶ MacOS : **MacPorts** et **Homebrew**;
  - ▶ Windows : **OneGet** et **Chocolatey**;

- ▶ Il vous sera **difficile** de travailler **efficacement** dans ce cours si vous n'installez pas un système **Unix** sur votre machine personnelle;
- ▶ Les utilisateurs de **Mac OS** pourront utiliser leur machine **telle quelle**.
- ▶ Pour les utilisateurs de **Windows**, il y a deux **solutions** possibles :
  - ▶ **[Recommandé]** Installer un système **Linux** en partition **double** ou comme **unique** système (**Ubuntu** ou **Mint**);
  - ▶ **[Moins recommandé]** Installer une **machine virtuelle**.

## Se pratiquer à la maison

- ▶ OverTheWire.org propose un jeu sérieux vous faisant manipuler les commandes Unix de base;
- ▶ Bandit: <http://overthewire.org/wargames/bandit/>
- ▶ Le jeu se joue via SSH;
- ▶ Il y a 27 niveaux à débloquer;
- ▶ Pratiquez vous à la maison;

# Table des matières

1. Présentation du cours
2. Environnement Unix
3. Environnements de développement
4. Le format Markdown
5. Le logiciel Git

# Environnements de développement

- ▶ L'outil de base d'un programmeur est son **environnement de développement**;
- ▶ En anglais, **integrated development environment (IDE)**;
- ▶ Quelques exemples :



- ▶ Pourtant, de nombreux programmeurs avancés préfèrent un simple éditeur de texte. Pourquoi ?
- ▶ Autre référence intéressante : Unix comme EDD.

# Éditeurs de texte (1/2)

L'offre d'éditeurs de texte est très variée :

- ▶ Notepad/**Notepad++** (Windows);
- ▶ TextEdit (MacOS);
- ▶ Gedit (Linux);
- ▶ **SublimeText** (multiplateforme);
- ▶ **Emacs** et ses dérivés (multiplateforme);
- ▶ **Vi/Vim** et ses dérivés (multiplateforme).



## Éditeurs de texte (2/2)

- ▶ Dans le cours, l'éditeur **préféré** sera **Vim**, mais vous êtes libre d'utiliser celui de **votre choix**.
- ▶ **N'**utilisez **pas** les éditeurs installés par défaut :
  - ▶ Windows : **Notepad**;
  - ▶ Mac OS : **TextEdit**.
- ▶ Dans tous les cas, assurez-vous que vos fichiers sont enregistrés au format **UTF8**;
- ▶ Tout fichier ayant un problème d'**encodage** sera considéré comme non valide.

- ▶ Un des plus anciens **éditeurs de texte**;
- ▶ En 2009, un **sondage** le plaçait comme l'éditeur de texte le **plus utilisé**;
- ▶ Son ancêtre, **vi**, a été créé par Bill Joy en **1976**;
- ▶ Le nom **Vim** vient de **Vi iMproved**;
- ▶ Supporté sur toutes les **plateformes** habituelles (Linux, MacOS, Windows).

# Avantages/inconvénients

## ► **Avantages :**

- Très **mature**;
- Interaction **directe** avec le **terminal**;
- Installé par **défaut** sur toutes les plateformes **Unix**;
- Extrêmement **rapide**, en particulier pour la programmation **à distance**;
- Hautement **configurable**, etc.

## ► **Inconvénients :**

- Orienté seulement **clavier** (certains dérivés, comme **GVim** permettent une utilisation limitée de la souris);
- Courbe d'apprentissage **difficile** pour les débutants.

# Configuration

- Il est essentiel de configurer Vim (fichier `.vimrc`) :

```
36 " Syntax highlighting
37 syntax on
38
39 " Commentaries
40 nmap <space>c <Plug>CommentaryLine
41 xmap <space>c <Plug>Commentary
42
43 " Colorscheme
44 colorscheme desert
45
46 " Source the vimrc file after saving it
47 augroup vimrc
48   au!
49   au bufwritepost ~/.vim/vimrc source $MYVIMRC
50 augroup END
51
52 " Filetypes
53 au BufRead,BufNewFile *.tikz setfiletype tex
54 au BufRead,BufNewFile *.sage setfiletype python
55
56 " Replace tab by spaces
57 set tabstop=4 | set shiftwidth=4 | set expandtab
58 autocmd filetype tex set tabstop=2 | set shiftwidth=2
59 autocmd FileType make setlocal noexpandtab
```

# Table des matières

1. Présentation du cours
2. Environnement Unix
3. Environnements de développement
4. Le format Markdown
5. Le logiciel Git

- ▶ C'est un format **texte** utilisant certains caractères spéciaux pour le **structurer**;
- ▶ Ces formats sont très pratiques pour rédiger de la **documentation** d'un programme ou d'un système;
- ▶ Quelques formats populaires :
  - ▶ Markdown;
  - ▶ ReStructuredText;
  - ▶ AsciiDoc, etc.;

- ▶ Les fichiers **Markdown** portent généralement l'extension **.md** ou **.markdown**;
- ▶ Ils peuvent facilement être transformés en **HTML**, **PDF**, etc.
- ▶ Un utilitaire très pratique pour cela est **Pandoc**.

- ▶ Par exemple, la commande

```
$ pandoc -s -f markdown -t html exemple.md -o exemple.html
```

produit le fichier **exemple.html**.

- ▶ Dans le cours, **tous** vos projets devront être documentés à l'aide d'un fichier **README.md**.

# Fichier README.md

```
# Travail pratique 1
```

```
## Description
```

Ce programme permet d'afficher en format ASCII une montagne dont les parties concaves sont remplies d'eau. Il est possible de préciser les caractères représentant la terre et l'eau lors de l'affichage.

Le projet a été réalisé dans le cadre du cours INF3135 Construction et maintenance de logiciel de la session d'automne 2016 à l'Université du Québec A Montréal.

```
## Auteur
```

Alexandre Blondin Massé

```
## Fonctionnement
```

Pour générer un exécutable du programme, il suffit d'entrer la commande

```
make
```

Puis on lance le programme à l'aide de la commande

```
./tpl <terre> <eau> <hauteurs>
```

où '<terre>' est un caractère représentant de la terre, '<eau>' est un caractère représentant l'eau et '<hauteurs>' est une suite de nombres naturels décrivant les hauteurs, colonne par colonne, de la montagne, séparées par des virgules.

```
...
```



## Travail pratique 1



### Description

Ce programme permet d'afficher en format ASCII une montagne dont les parties concaves sont remplies d'eau. Il est possible de préciser les caractères représentant la terre et l'eau lors de l'affichage.

Le projet a été réalisé dans le cadre du cours INF3135 Construction et maintenance de logiciel de la session d'automne 2016 à l'Université du Québec A Montréal.

### Auteur

Alexandre Blondin Massé

### Fonctionnement

Pour générer un exécutable du programme, il suffit d'entrer la commande

```
make
```

Puis on lance le programme à l'aide de la commande

```
./tp1 <terre> <eau> <hauteurs>
```

où **<terre>** est un caractère représentant de la terre, **<eau>** est un caractère représentant l'eau et **<hauteurs>** est une suite de nombres naturels décrivant les hauteurs, colonne par colonne, de la montagne, séparées par des virgules.

- ▶ Dans le cadre du cours, vous utiliserez minimalement les **éléments** suivants :
  - ▶ Les **sections** et les **sous-sections**;
  - ▶ Les **hyperliens**;
  - ▶ Les **listes** à puce et les **énumérations**;
  - ▶ Les **extraits** de code;
  - ▶ Les **images**, etc.
- ▶ Référence rapide (cheatsheet);
- ▶ Extension pour GitLab.

# Table des matières

1. Présentation du cours
2. Environnement Unix
3. Environnements de développement
4. Le format Markdown
5. Le logiciel Git

# Logiciel de contrôle de versions

- ▶ Permet de **stocker** un ensemble de **fichiers**;
- ▶ Conserve en mémoire la **chronologie** de toutes les modifications effectuées;
- ▶ Offre des services de **partage** des fichiers entre plusieurs **développeurs**;
- ▶ Est utilisé pour conserver les différentes **versions** du code source d'un projet;
- ▶ Permet également de gérer différentes **branches** dont les évolutions sont temporairement **indépendantes**.
- ▶ Garantit dans une certaine mesure l'**intégrité des fichiers**, car il est toujours possible de **revenir en arrière**.

# 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**;
- Son utilisation est **obligatoire**, notamment pour la **remise des travaux**.

# Naissance de Git

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

# Commandes les plus courantes

Quelques opérations **courantes** de Git :

- ▶ **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`, etc.

# Configuration de Git

- La configuration de Git est **très simple**;
- Elle se fait à l'aide d'un **fichier texte** généralement stocké dans le dossier **home** :

```
1 [user]
2   name = Alexandre Blondin Massé
3   email = alexandre.blondin.masse@gmail.com
4 [color]
5   branch = auto
6   diff = auto
7   interactive = auto
8   status = auto
9 [alias]
10  st = status -s
11  co = checkout
12  ci = commit
13  br = branch
```



# Hébergement de dépôts Git

- ▶ Lorsqu'on manipule un dépôt Git, la plupart des opérations se font **localement**;
- ▶ Cependant, il est très pratique de pouvoir **partager nos modifications**;
- ▶ Pour cela, il existe des **sites** dédiés à l'hébergement de tels projets :
  - ▶ **Github**;
  - ▶ **Bitbucket**;
  - ▶ **GitLab**, etc.
- ▶ Dans ce cours, vous devrez utiliser **GitLab**, qui offre **gratuitement** un nombre **illimité** de dépôts **privés** et de **contributeurs**.