

# Chapitre 1 : Environnement

## Construction et maintenance de logiciels

Guy Francoeur

basé sur du matériel pédagogique d'Alexandre Blondin Massé, professeur

**UQÀM** | **Département d'informatique**

# Table des matières

1. Matériel et droits
2. Environnement Unix
  - grep
  - sed
  - exemples
  - extras
3. Environnements de développement
4. Le Markdown (.md)
5. Introduction à Git
6. Développer sous Git
7. Git - fonctions avancées

# Table des matières

1. Matériel et droits
2. Environnement Unix
3. Environnements de développement
4. Le Markdown (.md)
5. Introduction à Git
6. Développer sous Git
7. Git - fonctions avancées

Le présent matériel de cours c'est inspiré du matériel pédagogique écrit par A. Blondin Massé, Professeur à l'UQAM.

# Table des matières

## 1. Matériel et droits

## 2. Environnement Unix

grep

sed

exemples

extras

## 3. Environnements de développement

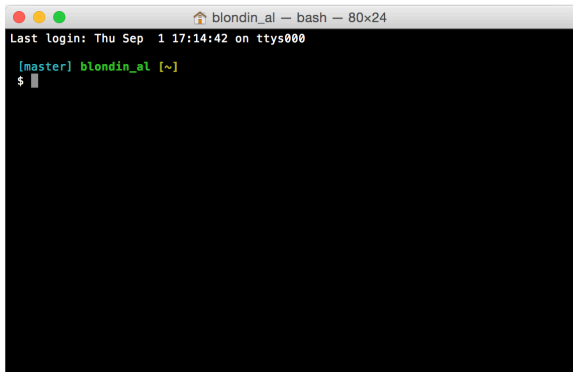
## 4. Le Markdown (.md)

## 5. Introduction à Git

## 6. Développer sous Git

# La console

- ▶ Dans ce cours, nous travaillerons avec le **terminal** ou la **console**.
- ▶ Le terminal agit comme **intermédiaire** pour lancer vos commandes dans le système d'exploitation **Unix**.



```
blondin_al - bash - 80x24
Last login: Thu Sep 1 17:14:42 on ttys000

[master] blondin_al [~]
$
```

# Commandes agissant sur les fichiers

- ▶ Résumé (**cheatsheet**) :

<https://ubuntudanmark.dk/filer/fwunixref.pdf>

- ▶ Commandes **fréquentes** :

Commande	Description
ls	Liste les fichiers dans un répertoire
ls -alhs	Liste tout (-a) détaillée (-l) humain (-h) size -s
cd	Change de répertoire
mkdir	Crée un répertoire
rm	Supprime un fichier (irréversible)
rm -rf	Supprime un répertoire (récursivement)
cp	Copie un fichier
cp -r	Copie un dossier (récursivement)
mv	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	Affichage interactif des processus actifs
ps	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
time	Affiche le temps d'exécution requis par un processus

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

# Le programme grep

- ▶ Identifie des **motifs** dans un texte;
- ▶ Basé sur les **expressions régulières**;
- ▶ Permet de faire une **recherche rapide** dans + fichiers;
- ▶ Documentation : **man grep**.
- ▶ Très utile lorsque **combiné** à d'autres programmes.

```
$ grep -R 'Statut' -A5 --color *
```

- ▶ -R récursif dans la structure de répertoire;
- ▶ -A5 la ligne qui correspond et les 5 suivantes.

# Le programme sed

- ▶ À l'instar de grep, il est basé sur les expressions **régulières**;
- ▶ Permet de faire un **“find and replace”**;

```
$ sed 's/int/long/g' fichier.c > nouveau.c
```

- ▶ Remplacer les **int** par des **long** :
- ▶ **s** signifie qu'on fait une **substitution**;
- ▶ **/** est un séparateur;
- ▶ **g** signifie qu'on fait la substitution **globalement**.

# Exemples

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

$ touch tp1.c

# 1. permission pour l'utilisateur
# 2. permission pour le groupe
# 3. permission pour les autres

$ chmod u=rwx,g=rw,o=r tp1.c

$ chmod 777 # rwx user, group, other
$ chmod 755 # rwx pour user, rx pour groupe et les autres
$ chmod +x script.sh # change les droits d'exécution
```

Il existe de nombreux programmes UNIX très **pratiques** :

- ▶ **grep** : recherche de motifs;
- ▶ **sed** : modifie en fonction d'expressions régulières;
- ▶ **cat** : liste le contenu d'un fichier vers la sortie standard;
- ▶ **echo** : affiche une chaîne vers **stdout**;
- ▶ **tree** : affiche la structure et les fichiers;
- ▶ **find** : recherche l'existence d'un fichier dans le système;
- ▶ **time** : affiche des statistiques sur l'exécution du processus;

# Installation de logiciels

- ▶ Un développeur doit 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 10 : **OneGet**, **Chocolately**.

# Table des matières

1. Matériel et droits
2. Environnement Unix
3. Environnements de développement
4. Le Markdown (.md)
5. Introduction à Git
6. Développer sous Git
7. Git - fonctions avancées



# 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 :

- ▶ UltraEdit ou Notepad++ (Windows);
- ▶ nano (Linux);
- ▶ SublimeText (multiplateforme);
- ▶ Emacs et ses dérivés (multiplateforme);
- ▶ Vi/Vim et ses dérivés (multiplateforme).

## Éditeurs de texte (2/2)

- ▶ Dans le cours, l'éditeur **préfér**é sera **nano**, mais vous êtes libre d'utiliser celui de **votre choix**.
- ▶ **N**'utilisez **pas** les éditeurs :
  - ▶ Windows : **Notepad**;
  - ▶ Mac OS : **TextEdit**;
- ▶ Dans tous les cas, assurez-vous que vos fichiers sont enregistrés au format **iso-8859**;
- ▶ Tout fichier ayant un problème d'**encodage** sera considéré comme non valide;
- ▶ Sauvegarder les fichiers avec l'option **UTF8 with noBOM**.

- ▶ 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.

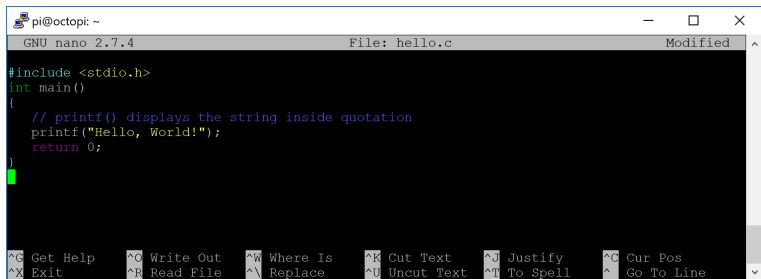
# Configuration

- 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
```

- ▶ Un très simple **éditeur de texte**;
- ▶ La première version de nano a été écrite par Chris Allegretta en 1999;
- ▶ Son ancêtre est **pico**, qui n'est pas gratuit;
- ▶ La configuration de GNU nano se fait à l'aide du fichier **.nanorc**;
- ▶ Les fichiers de configuration sont dans **/usr/share/nano/**
- ▶ `$ cd ; cat /usr/share/nano/c.nanorc » .nanorc`

# GNU nano



```
pi@octopi: ~  
GNU nano 2.7.4 File: hello.c Modified  
#include <stdio.h>  
int main()  
{  
    // printf() displays the string inside quotation  
    printf("Hello, World!");  
    return 0;  
}
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos  
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^\_ Go To Line

- ▶ **ctrl** + **x** sortir (demande pour la sauvegarde);
- ▶ **ctrl** + **o** sauvegarder;
- ▶ **ctrl** + **w** rechercher dans le fichier;
- ▶ **ctrl** + **k** effacement de la ligne au complet.



# Table des matières

1. Matériel et droits
2. Environnement Unix
3. Environnements de développement
4. Le Markdown (.md)
5. Introduction à Git
6. Développer sous Git
7. Git - fonctions avancées

- ▶ 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

```
./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.

...

## 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);
- ▶ Markdown quick reference cheat sheet;
- ▶ Extension pour GitLab.

# Table des matières

1. Matériel et droits
2. Environnement Unix
3. Environnements de développement
4. Le Markdown (.md)
5. Introduction à Git
6. Développer sous Git
7. Git - fonctions avancées

- ▶ 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**.

- ▶ **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

- ▶ Créer un nouveau projet : `git init`;
- ▶ Cloner un projet existant : `git clone`;
- ▶ Ajouter à l'index (staging) :
  - ▶ un nouveau fichier, une modification : `git add`;
- ▶ Documenter les modifications à l'index : `git commit`;
- ▶ Consulter l'historique : `git log`;
- ▶ Récupérer des changements à distance : `git pull`;
- ▶ Téléverser les changements → dépôt distant : `git push`;
- ▶ Lister, ajouter ou effacer une branche : *git branch*;
- ▶ Changer la branche active : *git checkout*

# Configuration de Git

- ▶ La configuration de Git est **très simple**;
- ▶ La configuration est gardée dans un **fichier texte** nommé **.gitconfig** généralement stocké dans le dossier **\$HOME**;
- ▶ Le fichier sera créé et rempli grâce a certaines commandes disponibles. Ceci est plus simple qu'éditer le fichier manuellement;
- ▶ Voici les commandes pour certaines configurations :

```
1 $ git config --global user.name "username"
2 $ git config --global user.email "email@domaine.ext"
3 $ git config --global core.editor nano
4 $ git config --global color.ui auto
5 $ git config --global push.default simple
```

# 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**.
- ▶ Dans ce cours, vous devrez utiliser **GitHub** ou GitLab\*, qui offre **gratuitement** un nombre **illimité** de dépôts **privés** et de **contributeurs**.

# Table des matières

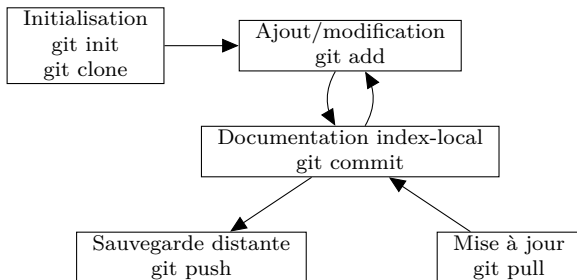
1. Matériel et droits
2. Environnement Unix
3. Environnements de développement
4. Le Markdown (.md)
5. Introduction à Git
6. Développer sous Git
7. Git - fonctions avancées

- ▶ On entend parfois dire **à tort** que le logiciel Git n'est utile que lorsqu'on travaille en **équipe**.
- ▶ C'est pourtant **très utile** :
  - ▶ Permet de **récupérer** une ancienne version;
  - ▶ Fournit naturellement une **copie de sauvegarde**;
  - ▶ Permet de **structurer** le développement;
  - ▶ Constitue un **aide-mémoire** de tout ce qui s'est passé, etc.
- ▶ Lors de vos travaux **TP et LAB**, vous aurez l'occasion de vous familiariser avec le gestionnaire de version de type **Git** et le développement en **solitaire**.

- ▶ Un **gestion de version** Git est simplement un répertoire muni d'un historique Git;
- ▶ Tout l'historique se trouve dans un répertoire caché nommé **.git**, disponible à la racine du projet;
- ▶ Ainsi, si vous supprimez ce dossier caché, vous supprimez par la même occasion tout l'**historique**;
- ▶ Il existe des plateformes qui permettent d'**héberger** des dépôts : GitHub, Bitbucket, GitLab, etc.
- ▶ On communique avec ces plateformes via des connexions **SSH** ou **HTTPS**.



# Flux opérationnel (*workflow*)



Dans le cadre du TP1 :

- Pourquoi devrais-je faire un **git pull**;
- Pourquoi ou quand devrais-je faire un **git push**;

- ▶ Il y a **deux façons** possibles d'initialiser un **répertoire** versionné par Git :
  - ▶ Avec la commande `git init**`, lorsqu'on démarre un **nouveau** projet;
  - ▶ Avec la commande `git clone`, lorsqu'on souhaite récupérer une copie d'un projet **existant**.
- ▶ Lors de vos travaux (labos et pratiques) vous devrez créer un ou des projets (*repository*) de type git;
- ▶ Si vous travaillez sur votre **machine personnelle** et que vous souhaitez récupérer votre projet (repo distant), il suffira de **cloner** le projet.

- ▶ *git commit* documente un état ou des modifications qui sont contenues pour le prochain *push*;
- ▶ *git add* pour ajouter un nouveau fichier ou une **modification** de fichier existant au *staging* (à l'index local);
- ▶ **Avant** de faire un *commit* :
  - ▶ Vérifier l'**état** de votre projet avec *git status*;
  - ▶ Au besoin, vérifiez les modifications avec *git diff*;
- ▶ **Après** avoir fait un *commit* :
  - ▶ Il est ensuite possible de faire *git push*, afin d'envoyer les changements dans le dépôt distant.
- ▶ *git log* résume l'**histoire** du projet;

# Table des matières

1. Matériel et droits
2. Environnement Unix
3. Environnements de développement
4. Le Markdown (.md)
5. Introduction à Git
6. Développer sous Git
7. Git - fonctions avancées

# Gestionnaire de versions de type Git

- ▶ En plus de simplifier la **gestion des versions**, un logiciel tel que Git offre d'autres fonctionnalités s'intègre directement dans certains ID;
- ▶ Un exemple, les plateformes **Github**, **Bitbucket** et **GitLab** qui proposent plusieurs services :
  - ▶ Demande d'**intégration** (*pull requests* ou *merge requests*);
  - ▶ Questions ou problématiques soulevées (*issues*);
  - ▶ Tickets (l'outil **trac**);
  - ▶ *Patches* (essentiellement un **diff**), etc.

- ▶ Les plateformes habituelles (GitHub, Bitbucket, GitLab) utilisent toutes ce mécanisme;
- ▶ Une *issue* est essentiellement une discussion, dans laquelle tous peuvent intervenir;
- ▶ Elle peut concerner
  - ▶ une **question**;
  - ▶ la découverte d'un **bogue**;
  - ▶ la demande d'intégration d'une **nouvelle fonctionnalité**;
  - ▶ une proposition d'**intégration**, avec demande de commentaires, etc.

# Comment intervenir?

- ▶ Soyez clair;
- ▶ Il faut utiliser adéquatement le format Markdown;
- ▶ Si c'est un bogue, donner une suite d'étapes permettant de le reproduire;
- ▶ Donner des d'**informations** détaillées :
  - ▶ La **version** testée (numéro ou *commit* exact);
  - ▶ La **plateforme** sur laquelle le bogue a été observé;
  - ▶ Si nécessaire, ajoutez une capture d'écran;
  - ▶ À l'occasion, un vidéo.

## Requête d'intégration (*Pull/merge requests*)

- ▶ Les plateformes habituelles (Github, Bitbucket, GitLab) utilisent toutes ce **mécanisme**;
- ▶ Une **requête d'intégration** est essentiellement un *patch* qu'on applique à un logiciel;
- ▶ Celle-ci est normalement expliquée en détail :
  - ▶ Quel type de modification? Correction d'un bogue? Ajout d'une **fonctionnalité**? de **documentation**?
  - ▶ On peut aussi associer une **personne**.
- ▶ Il y a souvent un processus d'**arbitrage** par les **pairs**, et on peut ensuite faire une **mise à jour**.



# Configurer son environnement pour Git

- ▶ Lorsqu'on commence à utiliser les **branches** sous Git, il est **primordial** de bien configurer son environnement;
- ▶ La première étape consiste à **enrichir** le fichier `.gitconfig` : ajout d'**alias**, visualisation du **graphe** de l'historique, etc.
- ▶ On peut modifier l'**invite de commande** du terminal pour afficher la **branche courante**.
- ▶ De cette façon, les risques d'erreur sont diminuer, mais pas nul.

# Fichier .gitconfig

```
[user]
  name = Alexandre Blondin Massé
  email = alexandre.blondin.masse@gmail.com
[color]
  branch = auto
  diff = auto
  interactive = auto
  status = auto
[alias]
  st = status -s
  co = checkout
  ci = commit
  br = branch
  gr = log --graph --full-history --all --color --pretty=tformat:"%x1b[31m%h%x09%x1b[32m%d%x1b[0m%x20%s%x20%x1b[33m(%an)%x1b[0m"
  puff = pull --ff --ff-only
  stats = shortlog -s -n --all
[core]
  precomposeunicode = true
[merge]
  ff = false
  tool = vimdiff
  conflictstyle = diff3
  prompt = false
[push]
  default = simple
```

Lien vers le fichier : [.gitconfig](#).

# Dupliquer un dépôt (*fork*)

- ▶ Lorsqu'on travaille sur du code **partagé**, une bonne pratique consiste à **dupliquer** le projet (en anglais, *fork*);
- ▶ De cette façon, chacun travaille sur sa **propre copie**;
- ▶ De plus, on peut **contrôler** quel code est intégré/rejeté;
- ▶ Terminologie :
  - ▶ *upstream* : dépôt principal, où le code est intégré;
  - ▶ *origin* : dépôt personnel, où on développe son code.
  - ▶ *local* : tout dépôt qui contient des fichiers, auxquels vous avez accès (ordinateur personnel, ordinateur de bureau, serveurs Java, UQAM).