



C2R1

# Tutorat Git

Version du September 30, 2014

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Histoire . . . . .	2
1.1.1	Versionnage de code? . . . . .	2
1.1.2	Outils de versionnage . . . . .	2
1.2	Installation du client git . . . . .	3
1.2.1	Windows . . . . .	3
1.2.2	Linux . . . . .	3
1.2.3	Mac OS . . . . .	3
1.3	Choix du serveur git . . . . .	3
<b>2</b>	<b>Manipulation basique de git</b>	<b>4</b>
2.1	Début du projet . . . . .	4
2.1.1	Créer le dépôt . . . . .	4
2.1.2	Récupérer un dépôt existant . . . . .	5
2.1.3	README . . . . .	5
2.2	Premier commit . . . . .	5
2.2.1	Récupérer les modifications depuis le serveur . . . . .	5
2.2.2	Envoyer les modifications vers le serveur . . . . .	5
2.2.3	diff/ignore . . . . .	6
2.3	Historique . . . . .	6
<b>3</b>	<b>Manipulation avancée</b>	<b>7</b>
3.1	Plus d'informations sur le fonctionnement de git . . . . .	7
3.2	branch . . . . .	7
3.2.1	créer sa branche . . . . .	7
3.2.2	fusionner . . . . .	8
3.2.3	résoudre un conflit . . . . .	10
3.2.4	savoir qui a fait quoi . . . . .	10
3.3	Supprimer un commit . . . . .	10
3.4	Se positionner sur un commit . . . . .	11
3.5	Contribuer à un autre repository . . . . .	11

# Partie 1

## Introduction

### 1.1 Histoire

#### 1.1.1 Versionnage de code?

Lorsqu'on travaille à plusieurs sur un projet, il est fréquent de devoir travailler au même moment. C'est ce à quoi servent les gestionnaires de version. Ils permettent à chaque développeur de travailler localement, puis de l'envoyer aux autres développeurs une fois leurs modifications finie.

#### 1.1.2 Outils de versionnage

Il existe de nombreux outils de versionnage de code. Les 3 plus connus sont sans doute Git, Mercurial et SVN.

##### **SVN (subversion)**

Ce fut le plus utilisé pendant longtemps. Développé par la fondation Apache, il s'agit d'une amélioration d'un programme nommé CVS (très peu utilisé aujourd'hui). Le plus gros problème de SVN est qu'il s'agit d'un système centralisé. Un serveur contient donc le code, des clients travaillent dessus. Il n'y a qu'un seul versionning. C'est le gestionnaire de version utilisé notamment par Apache, freeBSD et sourceforge.

##### **Mercurial**

Contrairement à SVN, il s'agit d'un système décentralisé. Chacun possède son propre *repository* et publie son code sur le *repository* public. Une autre différence avec SVN est qu'il utilise la notion de *changeset*. C'est à dire qu'il préfère garder en mémoire les changements appliqués que les versions des fichiers. Il est utilisé notamment par Mozilla (Firefox, Thunderbird, ...), Facebook, et adblock plus.

##### **Git**

Git possède très peu de différences avec Mercurial, mais l'histoire a fait qu'il c'est plus imposé. C'est en partie grâce à Linus Torvalds qui en a fait la pub et des projets comme github que nous allons utiliser dans le reste du tutorat.

C'est le gestionnaire de version utilisé par l'équipe de développeurs du noyau linux.

Pour plus d'informations sur les différences entre mercurial et Git, vous pouvez vous référer à ces articles : <http://importantshock.wordpress.com/2008/08/07/git-vs-mercurial/>, <http://www.rockstarprogrammer.org/post/2008/apr/06/differences-between-mercurial-and-git/>.

## 1.2 Installation du client git

### 1.2.1 Windows

Lancez le programme que vous pouvez trouver sur cette page : <http://msysgit.github.io> ou <http://www.git-scm.com/>

### 1.2.2 Linux

`apt-get install git` ou `yum install git` selon la distribution

### 1.2.3 Mac OS

`sudo port install git-core +svn +doc +bash_completion +gitweb`

## 1.3 Choix du serveur git

Pour la suite de ce tutorat, il vous faut créer un compte github : <https://github.com/>. Vous pouvez aussi vous intéresser à gitlab : <https://about.gitlab.com/>

## Partie 2

# Manipulation basique de git

## 2.1 Début du projet

### 2.1.1 Créer le dépôt

Pour créer le dépôt, il faut initialiser un dossier. Ceci se fait avec la commande `git init` (cette étape est optionnelle avec github). Ensuite, il faut créer le dépôt sur le serveur.

Sur Github, ceci se fait avec un formulaire, accessible en cliquant sur le bouton *create repository*, à côté de la liste des dépôts déjà existants.

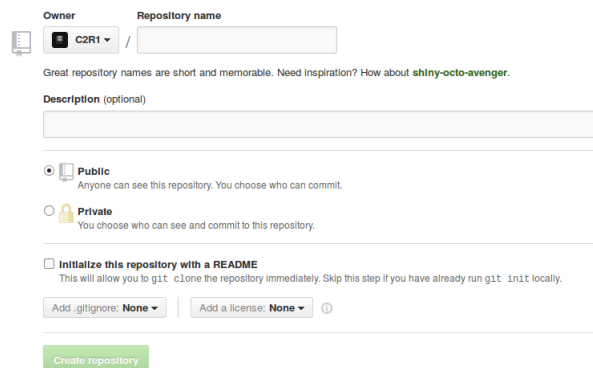
The image shows the GitHub 'Create repository' form. At the top, there's a section for 'Owner' (C2R1) and 'Repository name' (a text input field). Below this is a note: 'Great repository names are short and memorable. Need inspiration? How about shiny-octo-avenger.' Then, there's a 'Description (optional)' text input field. Below that, there are two radio button options: 'Public' (selected) and 'Private'. The 'Public' option has a subtext: 'Anyone can see this repository. You choose who can commit.' The 'Private' option has a subtext: 'You choose who can see and commit to this repository.' Below these is a checkbox labeled 'Initialize this repository with a README' and a subtext: 'This will allow you to git clone the repository immediately. Skip this step if you have already run git init locally.' At the bottom, there are two dropdown menus: 'Add .gitignore: None' and 'Add a license: None'. Finally, there is a green 'Create repository' button.

Figure 2.1: Formulaire de création de dépôt

Nous avons alors un formulaire de création de dépôt (cf figure 2.1), qui nous demande :

- le nom du dépôt (*Repository name*)
- une description (optionnelle)
- s'il faut initialiser le dépôt (à ne pas cocher si vous avez fait `git init`).

### 2.1.2 Récupérer un dépôt existant

Pour récupérer un dépôt existant, il faut connaître l'*url* du dépôt. Par exemple `https://github.com/C2R1/Tutorats.git`. Vous trouverez cette adresse sur la page du dépôt sur github, sous le nom *HTTPS clone URL*.

il faut alors aller dans le dossier où l'on veut cloner le dépôt, et effectuer la commande `git clone <URL>`.

Ainsi, pour notre exemple : `git clone https://github.com/C2R1/Tutorats.git`.

### 2.1.3 README

Le README est un fichier de présentation du projet. Généralement, on y décrit son installation, les fonctionnalités à venir, comment contribuer, la licence, ...

## 2.2 Premier commit

Une fois que l'on a le dépôt cloné sur son ordinateur, on peut commencer à coder. Cependant, les modifications ne se font pas en temps réel sur le serveur. Ainsi, il faut connaître quelques commandes pour récupérer les modifications faites par les autres développeurs, et pouvoir leur envoyer nos modifications.

### 2.2.1 Récupérer les modifications depuis le serveur

Pour récupérer les modifications, il faut utiliser la commande `git pull`.

### 2.2.2 Envoyer les modifications vers le serveur

Pour envoyer les modifications au serveur, il faut tout d'abord dire à git les fichiers qui ont été ajoutés depuis la dernière modification. Ceci se fait avec la commande `git add <fichiers>`. Par exemple, pour ajouter tous les nouveaux fichiers, on peut utiliser `git add *`.

Ensuite, il faut "enregistrer" les modifications localement. On dit alors qu'on *commit* les changements. La commande est donc `git commit <fichiers>`, en spécifiant les fichier à *commit*. Pour enregistrer les modifications de tous les fichiers, on utilise l'option `-a` (pour *all*), et ainsi : `git commit -a`.

Durant un commit, il faut expliquer les changements effectués. C'est l'utilité du *commit message*. On peut spécifier ce message en utilisant l'option `-m <message>`, soit : `git commit -m <message>`.

Si l'on ne spécifie pas de message avec l'option `-m`, git lancera un éditeur de texte (`vi` par défaut) pour écrire le message. En cas de message vide, le commit ne peut se faire.

Enfin, il faut envoyer ses modifications au serveur. C'est l'utilité de la commande `git push`.

Par exemple, pour "dire" au serveur qu'on a ajouté le fichier `c2r1.dtc`, on utilisera la suite de commande :

```
>> git add c2r1.dtc
>> git commit -am "ajout du fichier c2r1.dtc"
>> git push
```

Soit :

- On “avertit” git qu’on ajoute le fichier,
- On *commit* les changements de tous les fichiers (a), avec un message (m),
- On envoie les modifications au serveur.

### 2.2.3 diff/ignore

Pour obtenir tous les changements effectués depuis le dernier commit. Voici la forme d’un fichier diff :

```
diff --git a/apps/system/js/sound_manager.js b/apps/system/js/sound_manager.js
index b796d13..3847b93 100644
--- a/apps/system/js/sound_manager.js
+++ b/apps/system/js/sound_manager.js
@@ -686,7 +686,11 @@
     };
 }

- function setVibrationEnabled(enabled) {
+ function setVibrationEnabled(enabled) {
+     //vibrate
+     if ('vibrate' in navigator && enabled) {
+         navigator.vibrate([200, 100, 200]);
+     }
+     setVibrationEnabledCount++;
+     SettingsListener.getSettingsLock().set({
+         'vibration.enabled': enabled

```

Note : On peut aussi obtenir la différence entre 2 commits avec cette commande.

On y trouve donc le numéro du commit, les fichiers modifiés, les lignes supprimées ainsi que les lignes ajoutées.

## 2.3 Historique

## Partie 3

# Manipulation avancée

### 3.1 Plus d'informations sur le fonctionnement de git

Quand vous committez avec git, git enregistre un objet qui contient un pointeur vers vos changements, l'auteur, le message de commit et un lien vers zéro parent si c'est le commit initial, 1 parent pour un commit classique, plus pour un commit résultant d'un merge entre 2 ou plusieurs branches. Un projet sera

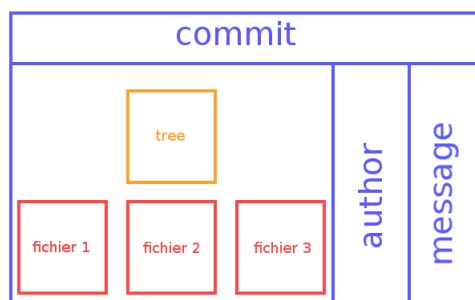


Figure 3.1: Forme d'un objet commit

donc représenté par une suite de ces objets. Avec un pointeur nommé HEAD représentant l'endroit où l'utilisateur se trouve.

### 3.2 branch

#### 3.2.1 créer sa branche

Pour créer une branche (et plus généralement pour gérer des branches) il suffit d'utiliser la commande `git branch`. Une autre méthode plus rapide est d'utiliser `git checkout -b <nom de la branche>` qui va se charger de créer la branche et de se placer dessus. Pour se placer sur une branche qui existe, il faut utiliser `git checkout <nom de la branche>` :



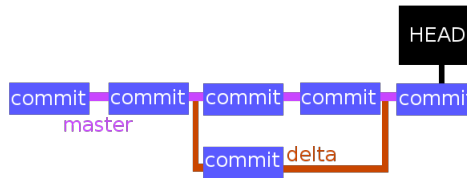


Figure 3.2: Représentation d'un git

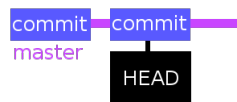


Figure 3.3: Avant git checkout -b

```
>> git checkout -b delta
```



Figure 3.4: Après git checkout -b

### 3.2.2 fusionner

Pour fusionner deux branches, il faut utiliser `git merge`. Exemple : On veut écraser une branche par une autre.

1/ On se place dans le repository. Ici, il contient un dossier git et un fichier README.md

```
>> ls
git README.md
```

2/ On regarde l'historique pour chercher sur quel commit on veut se placer (ici initial commit)

```
>> git log
commit dcf387590eaf281c10ee7f989a690c223d0b125a
Author: AmarOk1412 <amarok@enconn.fr>
Date: Tue Sep 30 13:11:15 2014 +0200
```

```
git log

commit 2986c83edc7ca9f1bcbc565fbe13a14091005b1f
Author: n0m1s <n0m1s@homnomnom.fr>
Date:   Tue Sep 30 13:04:14 2014 +0200
```

creation du depot

...

```
commit f1b4c0ebf59da0af7d782303f52bdae86c3d257b
Author: AmarOk1412 <amarok@enconn.fr>
Date:   Mon Sep 29 22:45:53 2014 +0200
```

Initial commit

3/ On se place sur le commit. Et on créer une branche (ici on la nomme delta).

```
>> git checkout f1b4c0ebf59da0af7d782303f52bdae86c3d257b
>> git checkout -b delta
Basculement sur la nouvelle branche 'delta'
>> git branch
* delta
  master
```

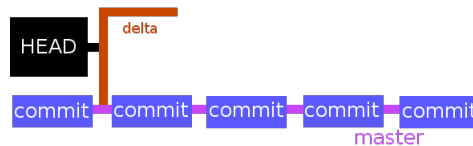


Figure 3.5: On crée une branche

4/ On effectue quelques modifications et on les commit

```
>> echo someDiff > README.md
>> git commit README.md -m "exemple"
[delta e1e999e] exemple
1 file changed, 1 insertion(+), 2 deletions(-)
```

5/ Sur la branche qu'on souhaite garder on merge master avec une stratégie dite ours (ours This resolves any number of heads, but the resulting tree of the merge is always that of the current branch head, effectively ignoring all changes from all other branches. It is meant to be used to supersede old development history of side branches. Note that this is different from the -Xours option to the recursive merge strategy. )

```
>> git merge -s ours master -m "ours attack"
Merge made by the 'ours' strategy.
```

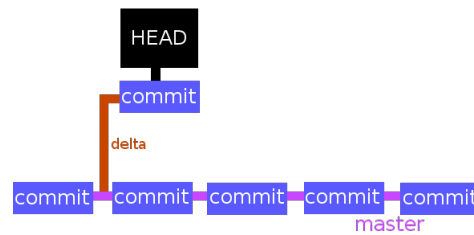


Figure 3.6: On effectue quelques modifications

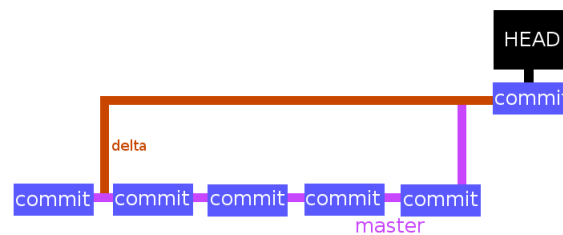


Figure 3.7: Merge avec stratégie ours

6/ On se place sur la branche que l'on souhaite écraser (master ici).

```
>> git checkout master
```

Basculement sur la branche 'master'  
 Votre branche est à jour avec 'origin/master'.

7/ Et on merge delta sur master

```
>> git merge --no-ff delta
```

Message des modifications

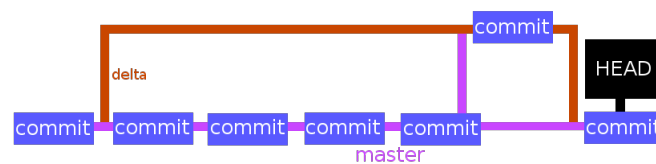


Figure 3.8: On merge nos modifications sur master

8/ Et voilà, delta a écrasé master.

```
>> ls
```

README.md

### 3.2.3 résoudre un conflit

### 3.2.4 savoir qui a fait quoi

Pour savoir qui a fait quoi, il suffit de faire git log. On a alors accès à l'historique. un commit se présente de la forme : "commit commit Author: Auteur <mail> Date: Date message du commit"

## 3.3 Supprimer un commit

Pour supprimer un commit, il suffit de faire git revert du commit. Exemple :

```
>>git log
"commit c610b4bbc6d631b798f158d2b999f0d0d0e108a4
Author: AmarOk1412 <amarok@enconn.fr>
Date: Tue Sep 30 09:26:40 2014 +0200

    paragraph git diff + readme + Contribuer à un autre repo

commit 616b3c6843e6aaa42bcf3a6f680760d94dcee57d
Author: TheMrNomis <n0m1s@homnomnom.fr>
Date: Tue Sep 30 09:23:07 2014 +0200

    titlepage

commit 781c38ff637c10166e6ef28d2574919cb865b7cf
Author: TheMrNomis <n0m1s@homnomnom.fr>
Date: Tue Sep 30 08:41:26 2014 +0200

    plan du tutorat git

commit 1cd1ccccb53846f9763044da4bf50f26a5aebd10
Author: n0m1s <nomis@nosferapti.(none)>
Date: Tue Sep 30 08:25:28 2014 +0200

    started git tutorial

commit f1b4c0ebf59da0af7d782303f52bdae86c3d257b
Author: AmarOk1412 <amarok@enconn.fr>
Date: Mon Sep 29 22:45:53 2014 +0200

    Initial commit
"
```

git revert f1b4c0ebf59da0af7d782303f52bdae86c3d257b va donc nous supprimer le fichier README qui a été créé pendant ce commit.

### 3.4 Se positionner sur un commit

Pour se placer à un état précis, on utilise la commande `git checkout` comme pour se placer sur une branche. Il faut donc faire `git checkout` du commit (`git checkout f1b4c0ebf59da0af7d782303f52bdae86c3d257b` par exemple) pour se placer sur un commit précis. IMAGE

### 3.5 Contribuer à un autre repository

Généralement, il y a deux possibilités : \* Contribuer sans coder, (graphisme, traduction, communication, ouverture de bugs) \* Coder L'ouverture d'issues nécessite de regarder un minimum si l'issue n'a pas déjà été ouverte, de détailler son problème et de bien regarder la version qu'on utilise. Pour programmer, il faut forker le dépôt et le récupérer en local. Puis il suffit de créer sa branche, de réaliser les modifications nécessaires. Quelques fois, il est demandé de créer des tests pour son bout de code. Avant de proposer il faut vérifier son code (norme, lisibilité, ...) Enfin il faut push votre travail sur votre fork et effectuer une pull request. La suite dépend du fonctionnement du projet.