

TD Mise en œuvre et utilisation de GIT

Version 1.2



Sommaire

PRESENTATION.....	3
INSTALLATION	3
SOUS LINUX	3
SOUS WINDOWS.....	3
SOUS MAC OS X	3
CONFIGURATION.....	4
DEBUT DU TRAVAIL AVEC GIT	4
CREATION D'UN NOUVEAU DEPOT	5
CLONAGE D'UN DEPOT EXISTANT	5
MODIFICATION DU CODE ET COMMITS.....	6
ETAPE PRELIMINAIRE	6
SUITE DU TRAVAIL.....	7
EFFECTUER UN COMMIT	9
CORRIGER UNE ERREUR	12
TRAVAIL AVEC DES BRANCHES	13
CREER UNE BRANCHE.....	13
AUTRES FONCTIONNALITES	16
TAGGER UNE VERSION	16
RECHERCHER DANS LES FICHIERS SOURCE	17
IGNORER DES FICHIERS (.GITIGNORE)	17
CREATION D'UN DEPOT SERVEUR	18
TRAVAIL AVEC UN SERVEUR.....	19
TELECHARGER LES NOUVEAUTES.....	19
ENVOYER VOS COMMITS.....	19
ANNULER UN COMMIT PUBLIE	20
CONCLUSION	21

Présentation

Git présente les particularités suivantes :

- Il est très rapide ;
- Il sait travailler par branches (versions parallèles d'un même projet) de façon très flexible ;
- Il est assez complexe : un certain temps d'adaptation est nécessaire pour bien le comprendre et le manipuler ;
- Il est à l'origine prévu pour Linux, et est donc d'emploi plus facile pour les développeurs confirmés... travaillant de préférence sous Linux.

Il existe de nombreux sites web collaboratifs fondés sur Git : GitHub est probablement le plus connu car employé par de nombreux projets, dont jQuery, Symfony, Ruby on Rails...

Installation

Il existe des versions Git pour Linux, Windows et Mac OS X. Git est plus agréable à utiliser sous Linux et sensiblement plus rapide, mais il reste néanmoins utilisable sous Windows.

Sous Linux

Avec un gestionnaire de paquets, c'est très simple :

```
sudo apt-get install git-core gitk
```

Cela installe 2 paquets :

- git-core : c'est git, tout simplement. C'est le seul paquet vraiment indispensable ;
- gitk : une interface graphique qui aide à mieux visualiser les logs. Facultatif.

Sous Windows

Pour utiliser Git sous Windows, téléchargez et installez Git-1.9.5-preview20141217.exe. Il existe des versions plus récentes, mais celle-ci est en français. Lancez ensuite Git en cliquant sur Git GUI. Pour lancer la console, choisissez Git Bash.

Sous Mac OS X

Il y a plusieurs façons d'installer Git sous Mac OS X. Le plus simple est de se baser sur cet installeur pour Mac OS X. Vous allez télécharger une archive .dmg. Il suffit de l'ouvrir pour la monter, ce qui vous donnera accès à plusieurs fichiers. Ouvrez tout simplement l'archive .pkg qui se trouve à l'intérieur, ce qui aura pour effet d'exécuter le programme d'installation :

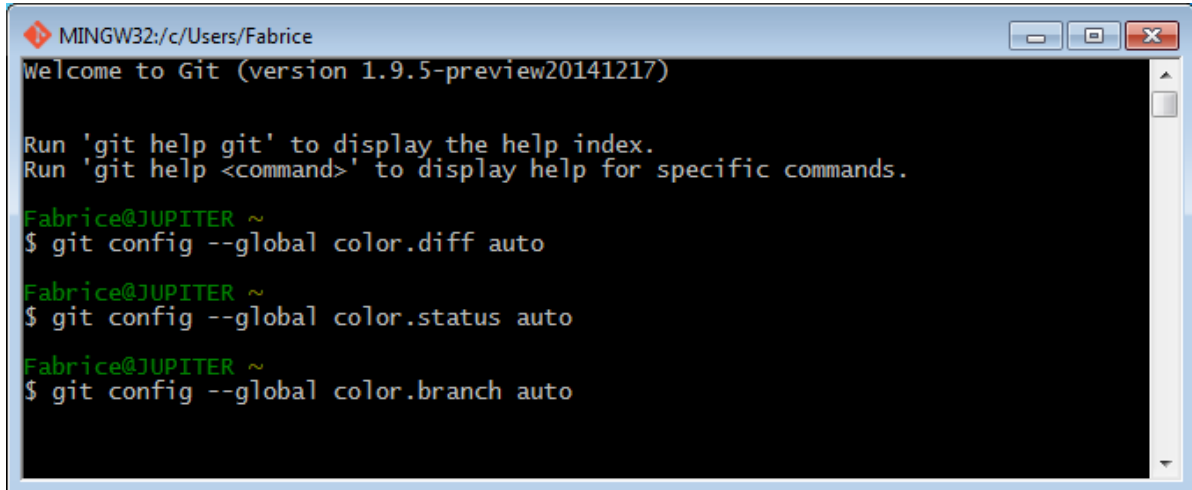
Suivez les étapes en laissant les valeurs par défaut (c'est suffisant), et vous aurez installé Git !

Il vous suffit ensuite d'ouvrir un terminal et vous aurez alors accès aux commandes de Git comme sous Linux.

Dans ce qui suit, nous travaillerons avec la version Windows. Comme vous le verrez, si le GUI est très pratique certaines opérations ne peuvent être effectuées que depuis la console (ou bien je n'ai pas su trouver comment faire dans le GUI...).

Configuration

La première étape consiste à activer les couleurs dans la console GIT : cela sera inutile avec l'interface graphique, mais rend les messages plus lisibles dans la console :



```
MINGW32: c:/Users/Fabrice
Welcome to Git (version 1.9.5-preview20141217)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Fabrice@JUPITER ~
$ git config --global color.diff auto

Fabrice@JUPITER ~
$ git config --global color.status auto

Fabrice@JUPITER ~
$ git config --global color.branch auto
```

De même, il faut configurer votre nom (ou pseudo) et votre email :

```
git config --global user.name "votre_pseudo"
git config --global user.email votre_email
```

Vous pourriez également effectuer cette étape depuis l'interface graphique en choisissant Edition > Option. Editez ensuite le fichier de configuration `.gitconfig` situé à la racine de votre répertoire personnel. Ajoutez à la fin une section alias :

```
[color]
    diff = auto
    status = auto
    branch = auto

[user]
    name = votre_pseudo
    email = moi@email.com

[alias]
    ci = commit
    co = checkout
    st = status
    br = branch
```

Ces alias permettent de raccourcir certaines commandes de Git. Ainsi, au lieu d'écrire `git checkout`, vous pourrez écrire si vous le désirez `git co`.

Début du travail avec Git

Pour commencer à travailler avec Git, trois possibilités s'offrent à vous :

- créer un nouveau dépôt vide, si vous souhaitez commencer un nouveau projet ;
- Cloner un dépôt existant, autrement dit récupérer la totalité de l'historique des changements d'un projet.
- Ouvrir un dépôt existant.

Un **dépôt** représente une copie du projet. Chaque ordinateur de développeur travaillant sur le projet possède donc une copie du dépôt. Chaque dépôt regroupe tous les fichiers du projet ainsi que leur historique.

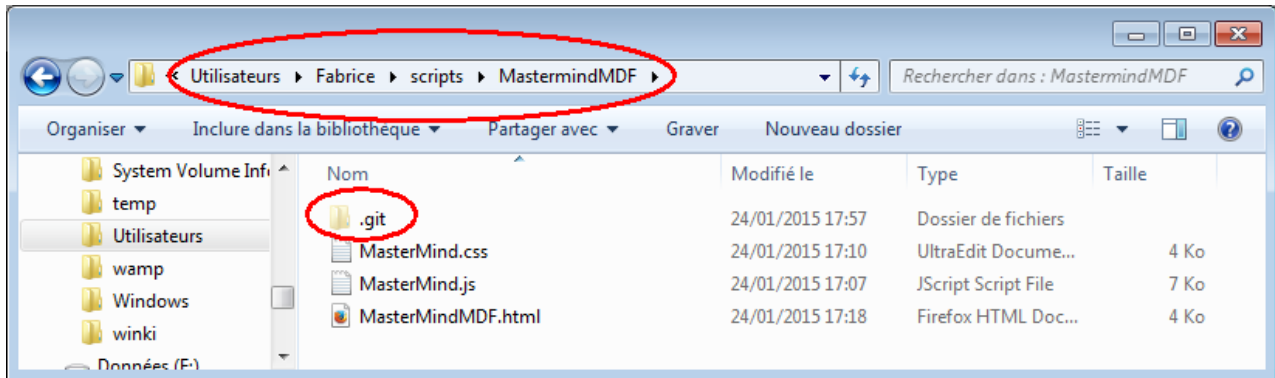
Création d'un nouveau dépôt

Depuis la console, déplacez-vous jusqu'à l'emplacement de votre projet sur le disque. Créez-le si nécessaire. Par exemple, /users/votrenom/mastermindMDF pour héberger le jeu Mastermind (ça devrait vous rappeler quelque chose...). Ensuite, initialisez un dépôt Git tout neuf dans ce dossier avec la commande :

```
git init
```

Avec l'interface graphique, cliquez sur Créer un nouveau dépôt, naviguez jusqu'au répertoire pertinent et cliquez sur Créer.

C'est tout ! Dans les deux cas, vous venez de créer un nouveau projet Git dans le dossier où vous vous trouvez. Un dossier caché `.git` vient d'être créé. Il faudra ensuite créer les fichiers source du projet et les faire connaître à Git en faisant des *commits*, ce que nous verrons plus tard.



Clonage d'un dépôt existant

Cloner un dépôt existant consiste à récupérer tout l'historique et tous les codes source d'un projet avec Git. Pour cloner un dépôt depuis la console, il suffit de lancer la commande suivante :

```
git clone urldudépôt
```

où *urldudépôt* est l'adresse Internet (ou réseau) du dépôt concerné. Cela va créer un dossier et y télécharger tous les fichiers source du projet ainsi que l'historique de chacune de leurs modifications !

La connexion au dépôt s'effectue comme ici en HTTP, ou à l'aide des protocoles « `git://` » et « `ssh://` ». Le plus souvent, on utilise SSH car il permet de chiffrer les données pendant l'envoi et gère l'authentification des utilisateurs.

Git compresse automatiquement les données pour le transfert et le stockage afin de ne pas prendre trop de place. Néanmoins, le clonage d'un dépôt peut nécessiter un temps important. Les messages s'afficheront dans la console tout au long du transfert.

Vous pouvez ouvrir le dossier et regarder son contenu : il contient tout le code source du projet. Le seul dossier un peu particulier créé par Git est le dossier `.git`, un dossier caché situé à la racine du projet qui renferme l'historique des modifications des fichiers et la configuration de Git pour ce projet.

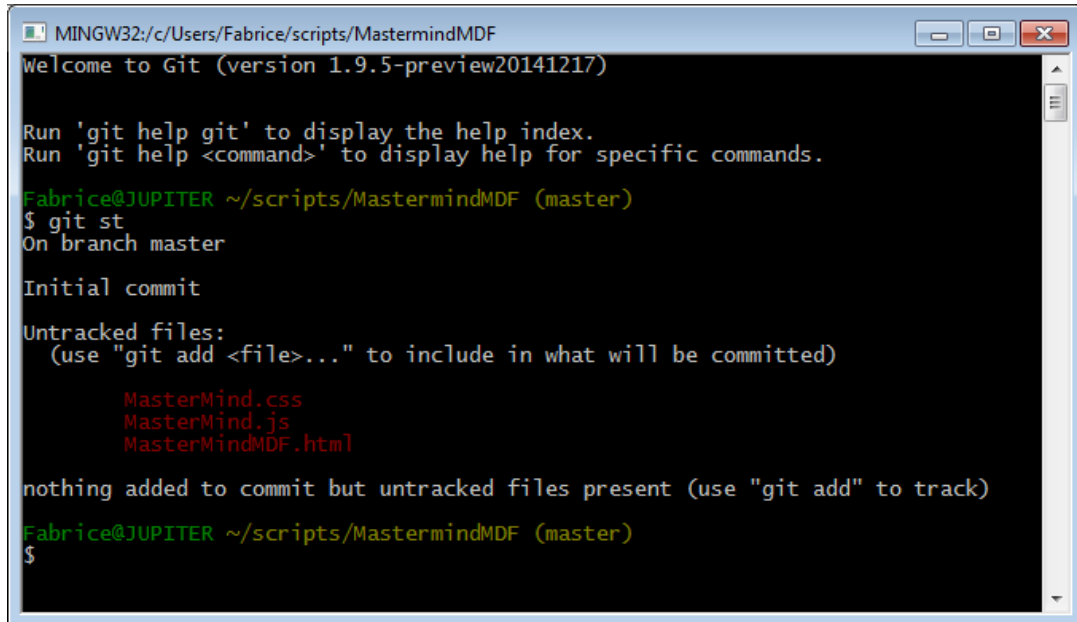
Vous n'aurez en principe jamais à vous rendre dans ce dossier caché, sauf éventuellement pour modifier le fichier de configuration `.git/config` qui s'y trouve.

Avec l'interface graphique, cliquez sur cloner un dépôt existant, naviguez jusqu'à celui-ci puis cliquez sur Cloner, en spécifiant éventuellement le mode de clonage (mais Standard est parfait pour la plupart des situations !)

Modification du code et commits

Etape préliminaire

À ce stade, vous avez créé ou cloné un dépôt Git, pour le projet MastermindMDF. Placez-vous dans le répertoire de base du code et recherchez les fichiers récemment modifiés à l'aide de la commande `git status` (si vous avez défini des alias servez-vous de `git st`) :



```
MINGW32:/c/Users/Fabrice/scripts/MastermindMDF
Welcome to Git (version 1.9.5-preview20141217)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Fabrice@JUPITER ~/scripts/MastermindMDF (master)
$ git st
On branch master

Initial commit

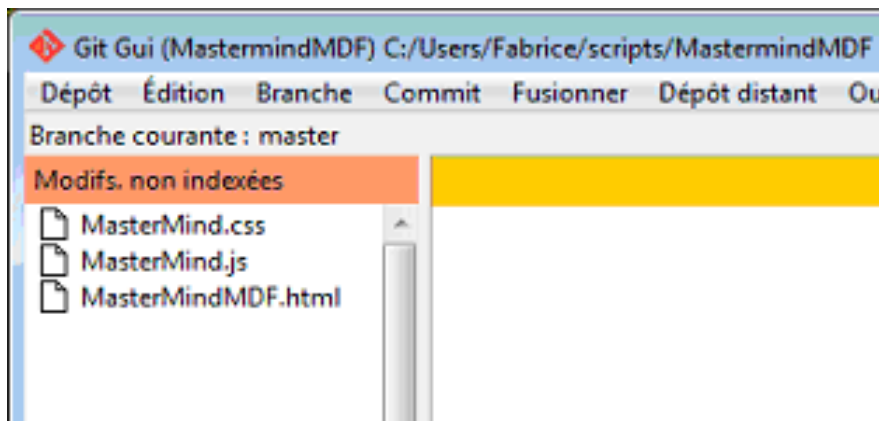
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        MasterMind.css
        MasterMind.js
        MasterMindMDF.html

nothing added to commit but untracked files present (use "git add" to track)
Fabrice@JUPITER ~/scripts/MastermindMDF (master)
$
```

Le message obtenu signale qu'il n'y a rien à commiter, mais qu'en revanche sont présents des fichiers non suivis (*untracked*).

L'interface graphique affiche ces fichiers dans la zone Modifs non indexées :

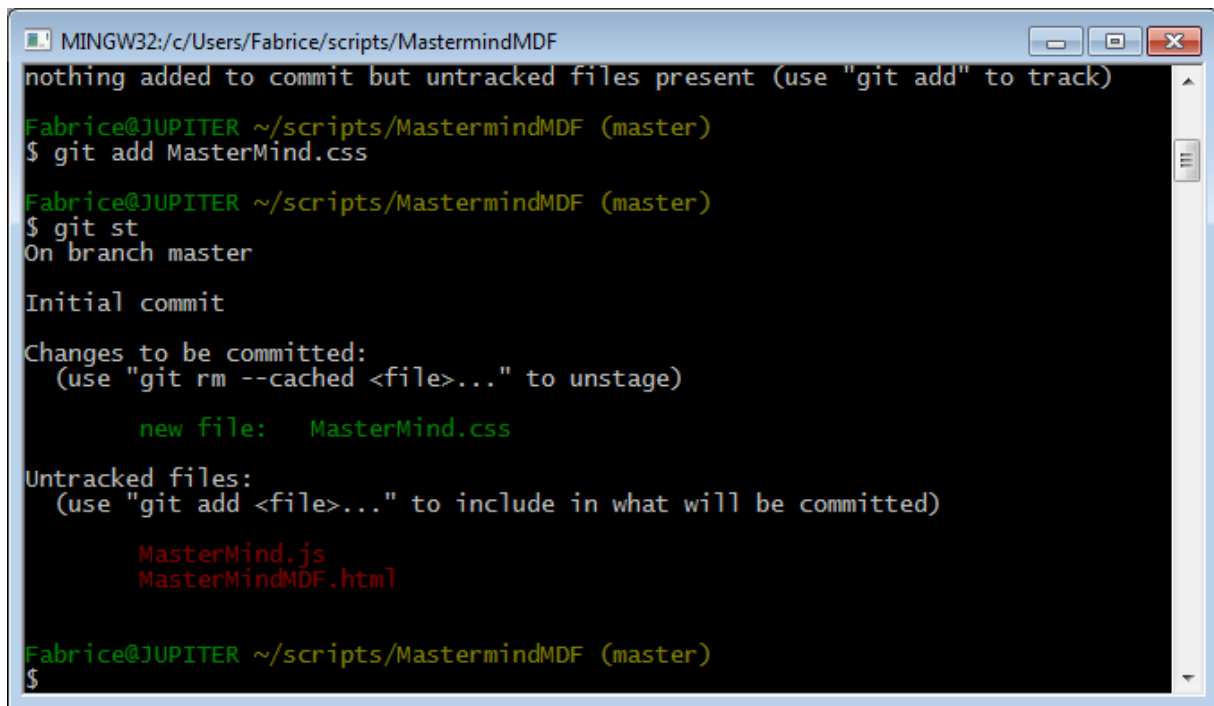


Dans la console, ajoutez les fichiers à suivre :

```
$ git add MasterMind.css
```

Attention, les noms de fichiers sont sensibles à la casse.

Vérifiez le résultat obtenu par `git st` :

A terminal window titled 'MINGW32:/c/Users/Fabrice/scripts/MastermindMDF'. The prompt is 'Fabrice@JUPITER ~/scripts/MastermindMDF (master)'. The user enters '\$ git add MasterMind.css'. The prompt changes to 'Fabrice@JUPITER ~/scripts/MastermindMDF (master)'. The user enters '\$ git st'. The output shows 'On branch master', 'Initial commit', 'Changes to be committed: (use "git rm --cached <file>..." to unstage)', 'new file: MasterMind.css', 'Untracked files: (use "git add <file>..." to include in what will be committed)', 'MasterMind.js', and 'MasterMindMDF.html'. The prompt returns to 'Fabrice@JUPITER ~/scripts/MastermindMDF (master)' with a '\$' at the end.

```
MINGW32:/c/Users/Fabrice/scripts/MastermindMDF
nothing added to commit but untracked files present (use "git add" to track)
Fabrice@JUPITER ~/scripts/MastermindMDF (master)
$ git add MasterMind.css
Fabrice@JUPITER ~/scripts/MastermindMDF (master)
$ git st
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   MasterMind.css

Untracked files:
  (use "git add <file>..." to include in what will be committed)

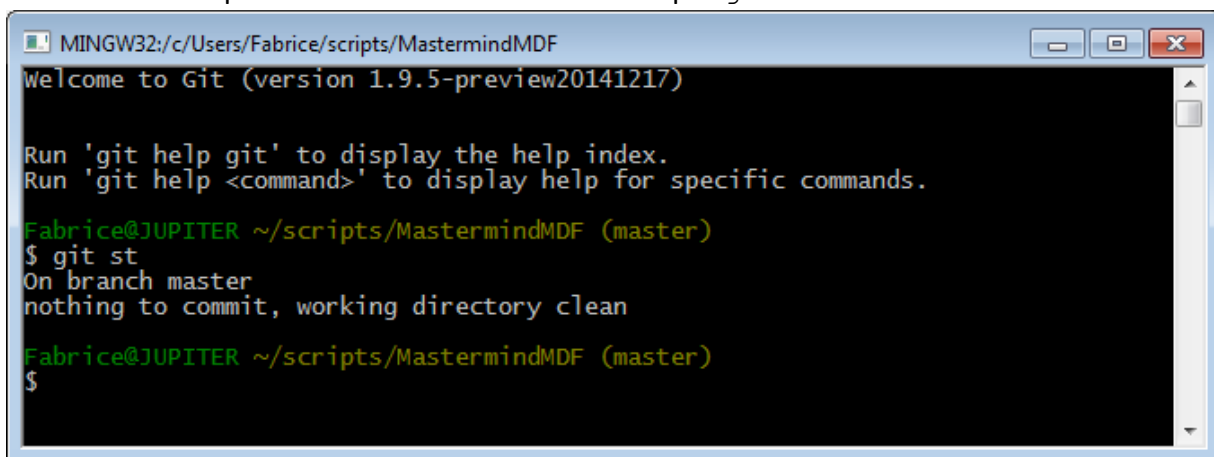
    MasterMind.js
    MasterMindMDF.html

Fabrice@JUPITER ~/scripts/MastermindMDF (master)
$
```

Le fichier css est désormais suivi. Procédez de même avec les deux autres fichiers.

Ensuite, saisissez la commande `git commit -a` pour commiter les trois fichiers (nous reviendrons sur cette commande). L'éditeur par défaut (normalement vim, la version améliorée de vi) s'ouvre pour vous permettre de saisir un message de commit. Saisissez-le (sur la première ligne) puis appuyez sur **ECHAP** pour passer en mode commande et saisissez `:wq` ou `:x` pour quitter l'éditeur.

Dans l'interface graphique, cliquez sur Indexer modifs, saisissez un message de commit, puis cliquez sur Commiter. Les fichiers disparaissent de l'interface. Vérifions par `git st` :

A terminal window titled 'MINGW32:/c/Users/Fabrice/scripts/MastermindMDF'. The prompt is 'Fabrice@JUPITER ~/scripts/MastermindMDF (master)'. The user enters '\$ git st'. The output shows 'Welcome to Git (version 1.9.5-preview20141217)', 'Run \'git help git\' to display the help index.', 'Run \'git help <command>\' to display help for specific commands.', 'Fabrice@JUPITER ~/scripts/MastermindMDF (master)', '\$ git st', 'On branch master', and 'nothing to commit, working directory clean'. The prompt returns to 'Fabrice@JUPITER ~/scripts/MastermindMDF (master)' with a '\$' at the end.

```
MINGW32:/c/Users/Fabrice/scripts/MastermindMDF
Welcome to Git (version 1.9.5-preview20141217)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Fabrice@JUPITER ~/scripts/MastermindMDF (master)
$ git st
On branch master
nothing to commit, working directory clean

Fabrice@JUPITER ~/scripts/MastermindMDF (master)
$
```

Git signale qu'il n'y a rien à faire (*nothing to commit*), que le répertoire de travail est "propre".

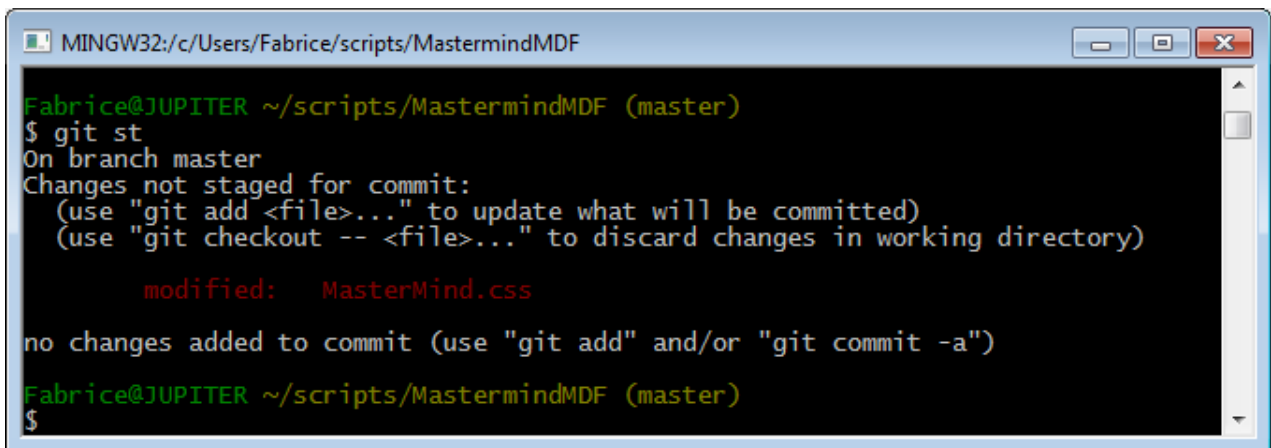
Suite du travail

Le travail avec Git se divise en général selon les étapes suivantes :

- Modification du code source ;
- Test du programme pour en vérifier le bon fonctionnement ;
- Enregistrement des modifications auprès de git à l'aide d'un *commit* ;
- Retour à l'étape 1 pour une autre modification.

Comme vu auparavant, vous enregistrez vos modifications auprès de git à l'aide d'un commit. C'est à vous de juger de l'opportunité de procéder à un commit. Si vous avez travaillé toute une journée sur un code et que vous ne faites qu'un commit à la fin de la journée, sans que cela ne concerne un seul bug ou une seule fonctionnalité, c'est probablement trop peu. Les commits sont là pour « valider » l'avancement de votre projet : faites-les régulièrement. Ni trop, ni trop peu...

Dans la situation suivante, le fichier css a été légèrement modifié. La commande `git status` le révèle clairement :

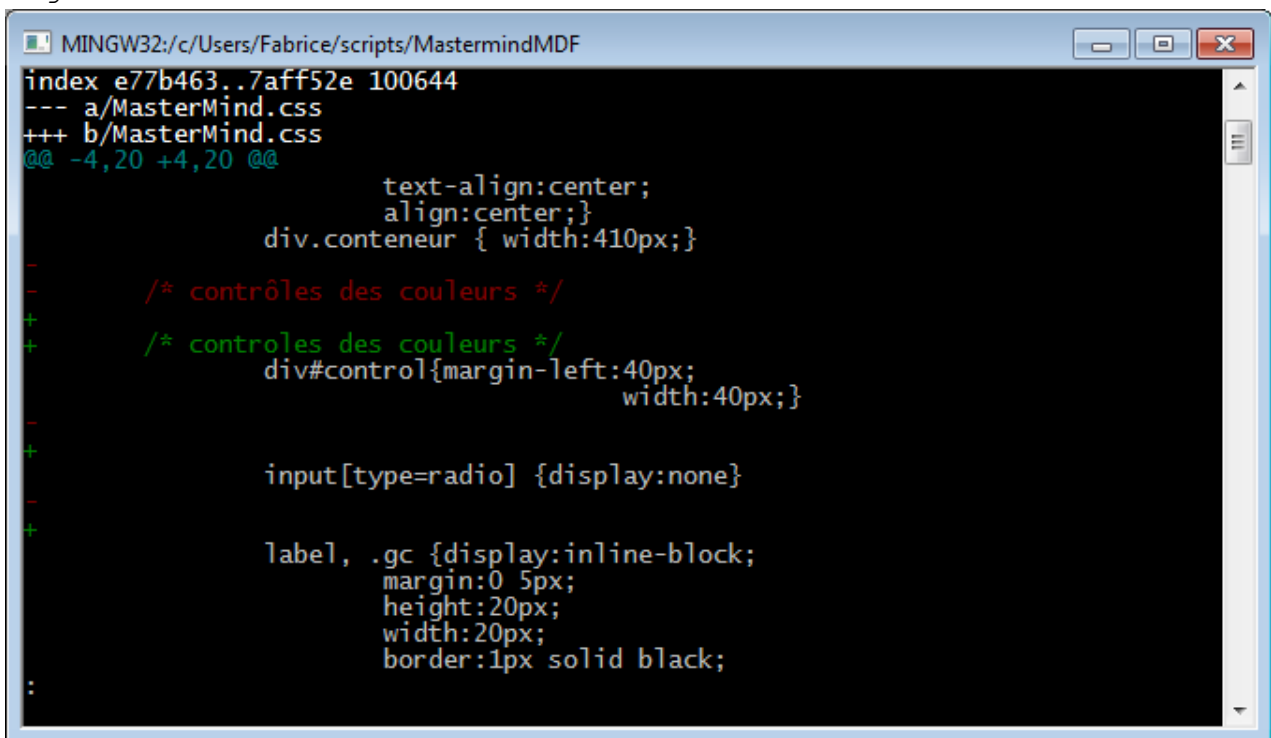


```
MINGW32:/c/Users/Fabrice/scripts/MastermindMDF
Fabrice@JUPITER ~/scripts/MastermindMDF (master)
$ git st
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   MasterMind.css

no changes added to commit (use "git add" and/or "git commit -a")
Fabrice@JUPITER ~/scripts/MastermindMDF (master)
$
```

Git énumère tous les fichiers modifiés sur le disque. Il peut aussi bien détecter les modifications que les ajouts, les suppressions et les changements de noms. Vous pouvez voir concrètement ce que vous avez modifié à l'aide de `git diff`:



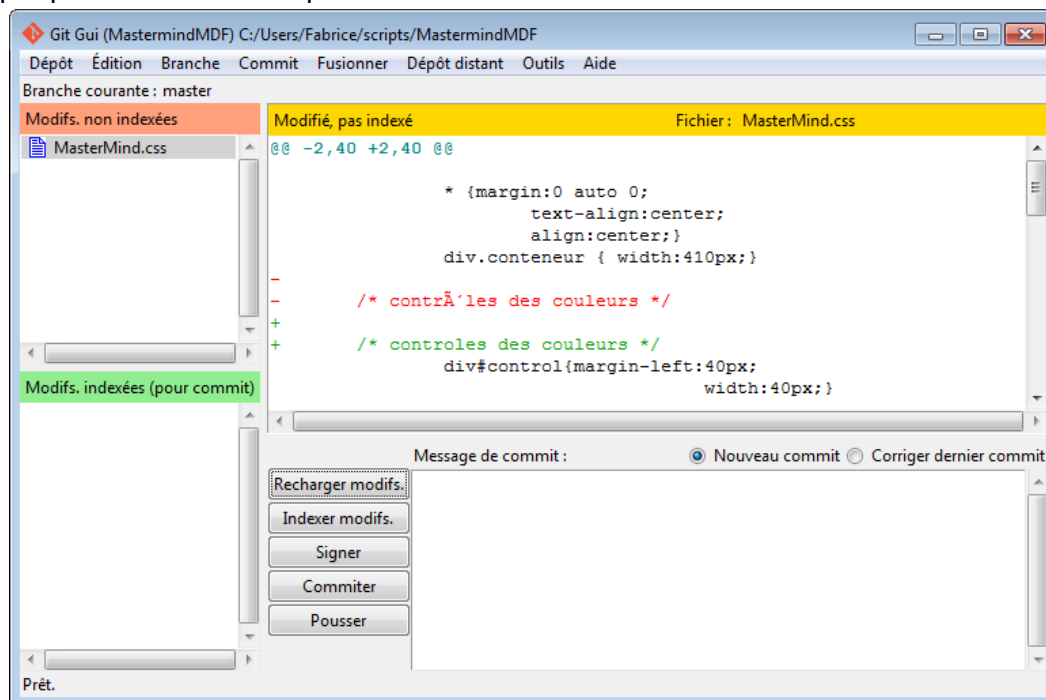
```
MINGW32:/c/Users/Fabrice/scripts/MastermindMDF
index e77b463..7aff52e 100644
--- a/MasterMind.css
+++ b/MasterMind.css
@@ -4,20 +4,20 @@
         text-align:center;
         align:center;}
     div.conteneur { width:410px;}
-
-    /* contrôles des couleurs */
+    /* controles des couleurs */
+    div#control{margin-left:40px;
+                width:40px;}
-
-    input[type=radio] {display:none}
+
+    label, .gc {display:inline-block;
+                margin:0 5px;
+                height:20px;
+                width:20px;
+                border:1px solid black;
+    :

```

Les lignes ajoutées sont précédées d'un « + » tandis que les lignes supprimées sont précédées d'un « - ». Les lignes sont colorées et donc faciles à repérer. Par défaut, Git affiche les modifications de tous les fichiers modifiés. Il n'y en avait ici qu'un, mais s'il y en avait eu plusieurs nous les aurions tous vus. Pour n'afficher que les changements d'un fichier précis, servez-vous de :

`git diff fichier`

L'interface graphique est tout aussi explicite.



Si les modifications vous paraissent bonnes et que vous les avez testées, il est temps de faire un commit.

Effectuer un commit

Préparation

Les fichiers modifiés sont affichés en rouge par `git status` ou apparaissent dans l'interface graphique dans la zone Modifs non indexés. Dans cet état, ils ne seront pas pris en compte par un commit. Vous devez les préparer au commit :

- Dans la console, saisissez `git add nomfichier1 nomfichier2` pour ajouter les fichiers à la liste de ceux devant faire l'objet d'un commit, puis faire un `git commit`. Si vous faites un `git status` après un `git add`, vous les verrez alors en vert ;
- Dans le GUI, choisissez Indexer modifs. Les fichiers concernés passent dans la zone verte Modifs indexés (pour commit).

L'emploi de `git add` (ou de l'indexation) est indispensable lorsque vous venez de créer de nouveaux fichiers que Git ne connaît pas encore. Cela lui permet de les prendre en compte pour le prochain commit.

Commit

Dans la console :

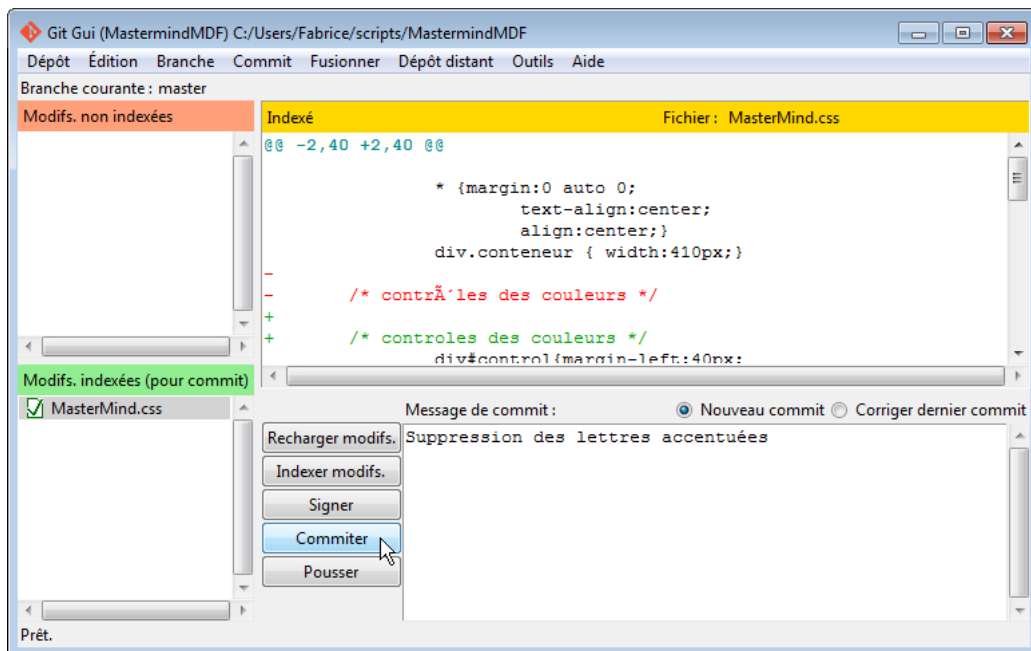
- Saisir `git commit -a` pour « commiter » tous les fichiers qui étaient listés dans `git status` dans les colonnes « *Changes to be committed* » et « *Changed but not updated* » (qu'ils soient en vert ou en rouge) ;
- Ou saisir `git commit nomfichier1 nomfichier2` pour indiquer lors du commit quels fichiers précis doivent être « commités ».

Mieux vaut retenir la seconde solution pour « commiter » tous les fichiers modifiés, et la troisième solution pour ne « commiter » que certains des fichiers modifiés.

Avec le GUI :

- Cliquez sur Commiter.

Lorsque la commande commit est lancée dans la console, l'éditeur par défaut s'ouvre. Vous devez sur la première ligne taper un message qui décrit à quoi correspondent vos changements. De même, avec le GUI, vous devez saisir un message dans la zone « Message de commit ». Un message de commit peut s'étendre sur plusieurs lignes.



Néanmoins, réservez la première ligne pour une courte description synthétique de vos changements. En l'absence de message de commit, celui-ci est annulé.

En règle générale, si vous ne pouvez pas résumer vos changements en une courte ligne c'est que vous avez passé trop de temps à faire des changements sans « commiter ».

Exemple de messages de commit d'une ligne corrects :

- « Ajout d'un dégradé sur le tableau de jeu. » ;
- « Décomposition de la fonction Vérifier en plusieurs fonctions. » ;
- « Résolution du bug #4 : initialisation correcte de la couleur ».

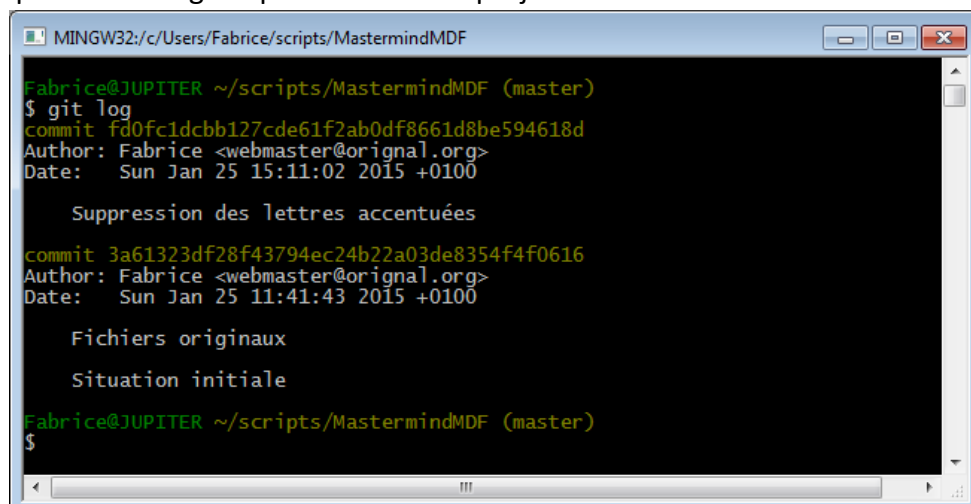
En général, un message de commit est écrit au présent. Une fois le message de commit enregistré, Git va officiellement sauvegarder vos changements dans un commit. Il ajoute donc cela à la liste des changements qu'il connaît du projet.

Un commit avec git est local (sauf s'il est envoyé sur le serveur, comme on le verra plus loin). Personne ne sait que vous avez fait ce commit pour le moment. Cela offre l'avantage de vous permettre de l'annuler si vous constatez avoir commis une erreur (ce qui est impossible avec SVN !).

Annulation d'un commit

Il est fréquent de chercher à comprendre ce qui s'est passé récemment, pourquoi une erreur a été introduite et comment annuler ce changement qui pose problème. C'est même là tout l'intérêt d'un logiciel de gestion de versions comme Git. Pour cela, vous devez consulter les journaux.

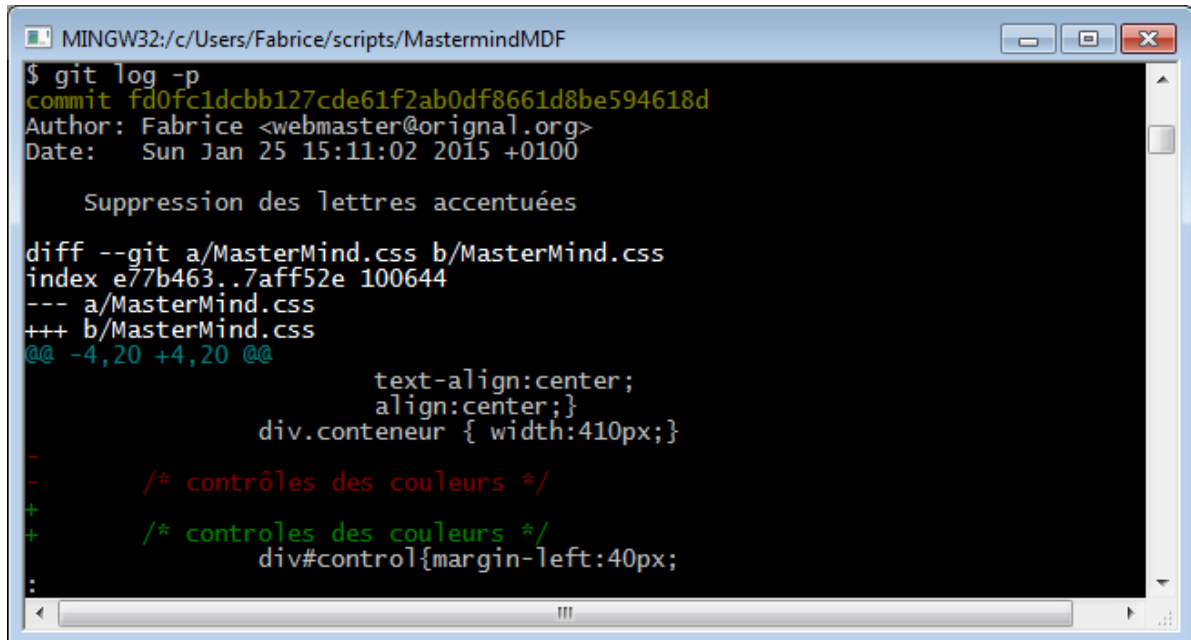
Les journaux (*logs*) permettent de consulter à tout moment l'historique des commits. Vous pouvez ainsi retrouver tout ce qui a été changé depuis le début du projet. Tout commit effectué sera affiché par `git log` :



Chaque commit est numéroté grâce à un long numéro hexadécimal, comme ici 3a61323df28f43794ec24b22a03de8354f4f0616, qui permet de l'identifier.

Vous pouvez parcourir un journal long avec les touches « Page up », « Page down » et les flèches directionnelles, et quitter en appuyant sur la touche « Q ».

Pour avoir le détail des lignes qui ont été ajoutées et retirées dans chaque commit, saisissez `git log -p` :

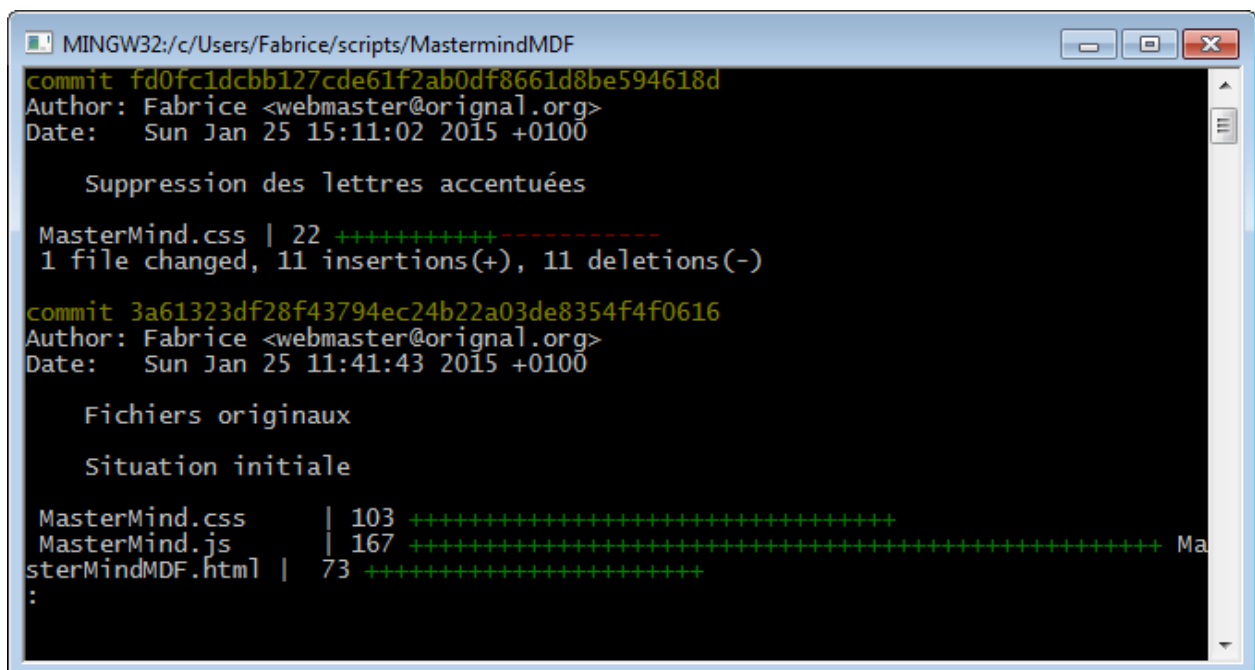


```
MINGW32:/c/Users/Fabrice/scripts/MastermindMDF
$ git log -p
commit fd0fc1dcbb127cde61f2ab0df8661d8be594618d
Author: Fabrice <webmaster@original.org>
Date: Sun Jan 25 15:11:02 2015 +0100

    Suppression des lettres accentuées

diff --git a/MasterMind.css b/MasterMind.css
index e77b463..7aff52e 100644
--- a/MasterMind.css
+++ b/MasterMind.css
@@ -4,20 +4,20 @@
                                     text-align:center;
                                     align:center;}
     div.conteneur { width:410px;}
-
-    /* contrôles des couleurs */
+    /* controles des couleurs */
     div#control{margin-left:40px;
:
```

Vous pouvez avoir un résumé plus court des commits avec `git log --stat` :



```
MINGW32:/c/Users/Fabrice/scripts/MastermindMDF
commit fd0fc1dcbb127cde61f2ab0df8661d8be594618d
Author: Fabrice <webmaster@original.org>
Date: Sun Jan 25 15:11:02 2015 +0100

    Suppression des lettres accentuées

MasterMind.css | 22 ++++++-----
1 file changed, 11 insertions(+), 11 deletions(-)

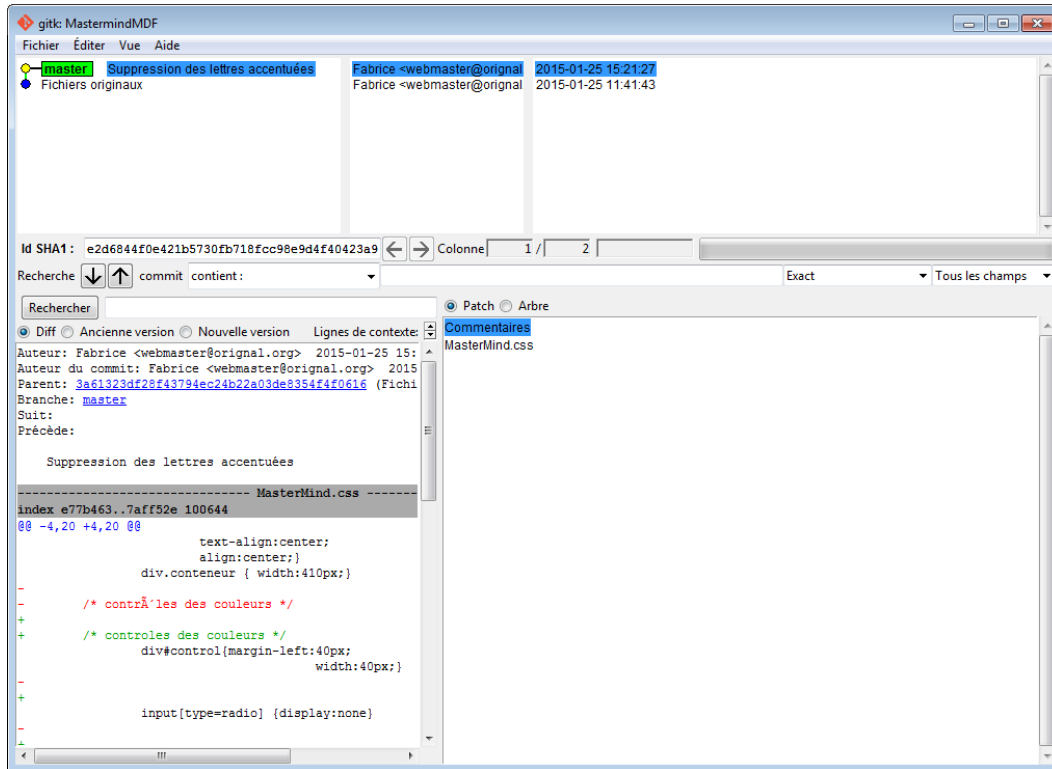
commit 3a61323df28f43794ec24b22a03de8354f4f0616
Author: Fabrice <webmaster@original.org>
Date: Sun Jan 25 11:41:43 2015 +0100

    Fichiers originaux

    Situation initiale

MasterMind.css      | 103 ++++++
MasterMind.js       | 167 ++++++
MasterMindMDF.html |  73 ++++++
:
```

Avec le GUI, choisissez Dépôt > Afficher l'historique de la branche.



Corriger une erreur

Voici différentes méthodes permettant de corriger les erreurs, selon leur ancienneté ou leur importance.

Modifier le dernier message de commit

Si vous avez fait une faute d'orthographe dans votre dernier message de commit ou que vous voulez tout simplement le modifier, vous pouvez le faire facilement grâce à la commande suivante :

```
git commit --amend
```

L'éditeur de texte s'ouvrira à nouveau pour changer le message.

Cette commande est généralement utilisée juste après avoir effectué un commit lorsqu'on se rend compte d'une erreur dans le message. Il est en effet impossible de modifier le message d'un commit lorsque celui-ci a été transmis à d'autres personnes. Avec le GUI, choisissez Commit > Modifier le dernier commit, modifiez le message de commit, puis cliquez sur Committer.

Annuler le dernier commit (méthode douce)

Si vous voulez annuler votre dernier commit :

```
git reset HEAD^
```

Cela annule le dernier commit et revient à l'avant-dernier.

Pour indiquer à quel commit on souhaite revenir, il existe plusieurs notations :

- HEAD : dernier commit ;
- HEAD^ : avant-dernier commit ;
- HEAD^^ ou HEAD~2 : avant-avant-dernier commit ;
- d6d98923868578a7f38dea79833b56d0326fcb1 : indique un numéro de commit précis ;
- d6d9892 : indique un numéro de commit précis (notation équivalente à la précédente, bien souvent écrire les premiers chiffres est suffisant tant qu'aucun autre commit ne commence par les mêmes chiffres).

Avec le GUI, choisissez Commit > Modifier le dernier commit.

Attention : seul le commit est retiré de Git, vos fichiers restent modifiés ! Vous pouvez alors à nouveau changer vos fichiers si besoin est et refaire un commit.

Annuler tous les changements du dernier commit (méthode dure)

Si vous voulez annuler votre dernier commit et les changements effectués dans les fichiers, il faut faire un reset hard. Celui-ci détruira sans demande de confirmation tout votre travail ! Faites donc attention et vérifiez que c'est bien ce que vous voulez faire !

```
git reset --hard HEAD^  /*!\ Annule les commits et perd tous les changements
```

Normalement, Git devrait vous dire quel est maintenant le dernier commit qu'il connaît (le nouveau HEAD).

Annuler les modifications d'un fichier avant un commit

Si vous avez modifié plusieurs fichiers mais que vous n'avez pas encore envoyé le commit et que vous voulez restaurer un fichier tel qu'il était au dernier commit, utilisez `git checkout` :

```
git checkout nomfichier
```

Le fichier redeviendra comme il était lors du dernier commit.

Annuler/Supprimer un fichier avant un commit

Supposons que vous veniez d'ajouter un fichier à Git avec `git add` et que vous vous apprêtiez à le commiter. Vous réalisez que ce fichier est une mauvaise idée et voudriez annuler votre `git add`. C'est facile en procédant comme suit dans la console :

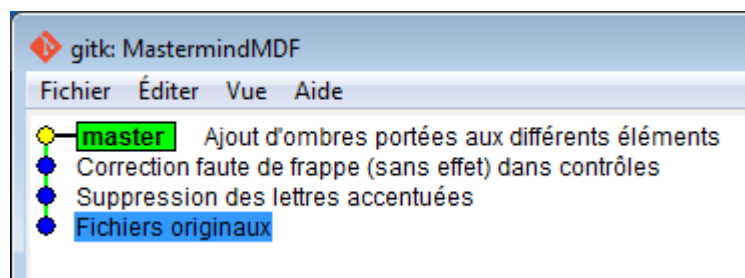
```
git reset HEAD -- fichier_a_supprimer
```

Dans le GUI, sélectionnez le fichier et choisissez Commit > Désindexer.

Travail avec des branches

Les branches sont un des aspects les plus intéressants de Git : c'est un moyen de travailler en parallèle sur diverses fonctionnalités. D'une certaine façon, vous disposez d'une « copie » du code source pour tester vos idées les plus farfelues et vérifier leur bon fonctionnement avant intégration dans le véritable code source du projet.

Dans Git, toute modification effectuée est par défaut appliquée à la branche principale nommée « master ».



C'est le cas pour le moment de mon projet Mastermind : vous voyez les commits effectués au fil du temps.

Créer une branche

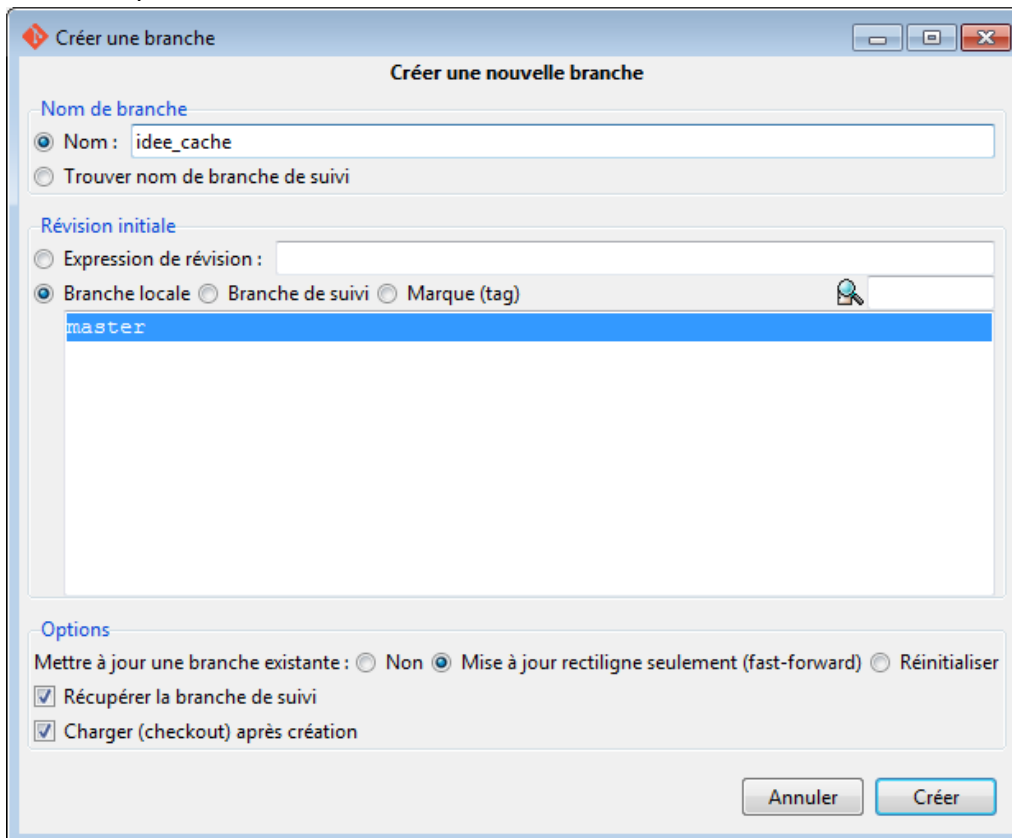
Il me vient toutefois une idée : modifier un peu radicalement l'interface du jeu, pour mettre le code (donc la solution) non plus comme actuellement sous le plateau mais tout en haut du plateau, et le masquer par un « cache » (comme dans le vrai jeu) tant que le joueur n'a pas gagné ou perdu. Cela va demander de modifier au moins deux fichiers (le HTML et le CSS) et probablement aussi le JavaScript... sans que je ne sois sûr pour le moment de la pertinence de cette modification.

Il suffit alors de créer une branche, nommée par exemple « idee_cache », dans laquelle je vais travailler en parallèle.

Dans la console, je saisis `git branch` suivi du nom de la branche :

```
git branch idee_cache
```

Avec le GUI, il suffit de cliquer sur Branche > Créer.



La branche créée ici est locale : vous seul y avez accès. Comme vous pouvez le deviner sur la copie d'écran, il est possible de publier une branche pour que d'autres personnes puissent y travailler : nous y reviendrons.

Une fois la branche créée, vous devriez la voir quand vous tapez simplement `git branch` :



Comme vous pouvez le voir, nous sommes passé sur la branche « idee_cache ». Une étoile devant indique que c'est la branche active. C'est parce que j'ai créé la branche à partir du GUI. Si je l'avais fait depuis la console, je pourrais aller sur la branche comme ceci :

```
git checkout idee_cache
```

`git checkout` sert aussi bien à changer de branche qu'à restaurer un fichier tel qu'il était lors du dernier commit. Cette commande a un double usage....

Lorsque vous changez de branche, vous ne changez pas de dossier sur votre disque dur : Git modifie vos fichiers pour qu'ils reflètent l'état de la branche dans laquelle vous vous rendez. Les branches de Git se comportent comme des dossiers virtuels : vous « sautez » de l'un à l'autre avec la commande `git checkout`. Vous restez dans le même dossier, mais Git modifie les fichiers qui ont changé entre la branche où vous étiez et celle où vous allez.

Il est maintenant possible de modifier à loisir les fichiers, de les tester puis de les commiter. Mais en revenant sur la branche « master » (par `git checkout master`) on constate que les commits effectués sur la branche `idee_cache` n'apparaissent pas sur la branche `master`. Les branches sont distinctes.

Vous pouvez même créer une sous-branche à partir d'une branche !

Pourquoi et quand créer une branche ?

Avant de procéder à des modifications sur le code source, posez-vous les questions suivantes :

- Cette modification sera-t-elle rapide ? ;
- Cette modification est-elle simple ? ;
- Nécessite-t-elle un unique commit ? ;
- Est-ce que je vois exactement comment effectuer cette modification d'un seul coup ?

Si la réponse à une seule de ces questions est « non », la création d'une branche est préférable. Créer une branche est très simple, très rapide et très efficace : profitez-en ! Créez une branche pour toute modification susceptible d'être un peu longue. Au pire, si elle se révèle plus courte que prévu, la branche aura été « pour pas grand-chose », mais c'est toujours mieux que de se rendre compte de l'inverse.

Fusionner les changements

Après avoir travaillé sur une branche, si celle-ci donne des résultats satisfaisants, vous devez la « fusionner » vers « master » à l'aide de la commande `git merge` (ou choisir Fusionner > Fusion locale avec le GUI). Vous pouvez fusionner n'importe quelle branche dans n'importe quelle une autre branche, bien que le plus courant soit de fusionner une branche de test dans « master ».

J'ai achevé mon travail avec la branche `idee_cache`. Soit cela ne me plait pas, et je reviens simplement à ma branche `master`, soit elle me convient et je décide de la publier : autrement dit, de fusionner le contenu de la branche `idee_cache` dans la branche principale `master`.

Rendez-vous d'abord dans la branche « master » à l'aide de `git checkout`, puis lancez la fusion avec `git merge` :

```
git checkout master
git merge idee_cache
```

Tous les commits de la branche `idee_cache` se retrouvent maintenant dans `master`. Le travail a été effectué de façon parallèle sans gêner la branche principale, pour finir appliqué à `master`.

Lorsque vous récupérez les nouveaux commits depuis le serveur avec `git pull`, cela revient en fait à appeler deux commandes différentes : `git fetch`, qui s'occupe du téléchargement des nouveaux commits, et `git merge`, qui fusionne les commits téléchargés issus de la branche du serveur dans la branche de votre ordinateur !

La branche `idee_cache` ne sert désormais plus à rien, on peut la supprimer :

```
git branch -d idee_cache
```

Git vérifie que votre travail dans la branche `idee_cache` a bien été fusionné dans `master`. Sinon, il vous en avertit et vous interdit de supprimer la branche (vous risqueriez sinon de perdre tout votre travail dans cette branche !).

Pour supprimer une branche contenant des changements non fusionnés (par exemple parce que votre idée à la base était une erreur), utilisez l'option `-D` (lettre capitale) :

```
git branch -D idee_cache /\ Supprime la branche et perd tous les changements.
```

Mettre de côté le travail en cours avant de changer de branche

Avant de changer de branche, vous devez avoir effectué un commit de tous vos changements. En clair, un `git status` ne devrait afficher aucun fichier en cours de modification.

Si vous avez des changements non « commités » et que vous changez de branche, les fichiers modifiés resteront comme ils étaient dans la nouvelle branche (et ce n'est en général pas ce que vous voulez !).

Pour éviter d'avoir à faire un commit au milieu d'un travail en cours, tapez :

```
git stash
```

Vos fichiers modifiés seront sauvegardés et mis de côté. Maintenant, `git status` ne devrait plus afficher aucun fichier (on dit que votre *working directory* est propre).

Vous pouvez alors changer de branche, faire vos modifications, committer puis revenir sur la branche où vous étiez.

```
git checkout master
(modifier des fichiers)
git commit -a
git checkout mabranche
```

Pour récupérer les changements que vous aviez mis de côté dans « mabranche », tapez :

```
git stash apply
```

Vos fichiers seront alors restaurés et se retrouveront donc l'état dans lequel ils étaient avant le git stash !

Les branches partagées

Il est possible de travailler à plusieurs sur une même branche. En fait, c'est déjà ce que vous faisiez en travaillant sur la branche master. Utilisez `git branch -r` pour lister toutes les branches connues du serveur :

```
$ git branch -r
  origin/HEAD
  origin/master
```

origin est le nom du serveur depuis lequel vous avez cloné le dépôt. Vous pouvez en théorie suivre directement les branches de plusieurs personnes (souvenez-vous, Git fonctionne un peu comme le *peer-to-peer*), mais on travaille plutôt avec un serveur pour suivre les changements.

Outre HEAD qui est un peu particulier, on constate que le serveur ne possède qu'une seule branche, master. Le serveur ne connaît donc que l'historique de la branche principale.

Si le serveur possède une autre branche, par exemple *origin/idee_cache*, et que vous souhaitez travailler dessus, il faut créer une copie de cette branche sur votre ordinateur qui va suivre (*track*, traquer) les changements sur le serveur.

```
git branch --track branche_locale origin/brancheserveur
```

Par exemple, vous pouvez créer une branche *idee_cache* locale qui sera connectée à la branche *idee_cache* du serveur :

```
git branch --track idee_cache origin/idee_cache
```

Lorsque vous ferez un pull depuis la branche *idee_cache*, les changements seront fusionnés dans la branche *idee_cache* locale. Il est donc capital de savoir dans quelle branche vous vous trouvez avant de faire un pull. Un pull depuis une branche ne met à jour que cette branche locale.

Autres fonctionnalités

Git permet d'accomplir bien d'autres choses souvent fort utiles.

Tagger une version

Il est possible de donner un alias à un commit précis pour le référencer sous ce nom. C'est utile par exemple pour dire « À tel commit correspond la version 1.3 de mon projet ». Cela permettra à d'autres personnes de repérer la version 1.3 plus facilement. C'est justement le rôle des tags.

Pour ajouter un tag à un commit :

```
git tag NOMTAG IDCOMMIT
```

Par défaut, les tags ne sont pas envoyés lors d'un push : vous devez préciser l'option `--tags` pour que ce soit le cas :

```
git push --tags
```

Pour supprimer un tag créé :


```
git tag -d NOMTAG
```

Rechercher dans les fichiers source

Git connaissant tous les fichiers source de votre projet, il est facile de faire une recherche à l'intérieur de tout votre projet. Par exemple, pour connaître les noms des fichiers qui contiennent dans le code source le mot `TODO`, il suffit d'écrire :

```
git grep "TODO"
```

Git accepte les expressions régulières pour les recherches.

Pour connaître les numéros des lignes qui contiennent le mot que vous recherchez, utilisez le paramètre `-n` :

```
git grep -n "TODO"
```

Ignorer des fichiers (.gitignore)

Pour ignorer un fichier dans git, créez un fichier `.gitignore` (à la racine) et indiquez-y les noms des fichiers à ignorer, à raison d'un par ligne. Le caractère générique `*` et les chemins sont autorisés :

```
project.xml
dossier/temp.txt
*.tmp
cache/*
```

Aucun de ces fichiers n'apparaîtra dans `git status`, même s'il est modifié, et ne paraîtra donc pas dans les commits. Il est précieux d'employer ceci par exemple sur les fichiers temporaires, qui n'ont aucune raison d'apparaître dans Git.

Création d'un dépôt serveur

Si vous souhaitez mettre en place un serveur de rencontre pour votre projet, il suffit d'y créer (`git init`) ou d'y cloner (`git clone`) un dépôt en ajoutant l'option `--bare`. Cela a pour effet de créer un dépôt qui contient uniquement le dossier `.git` représentant l'historique des changements. C'est suffisant, car les fichiers source ne sont jamais modifiés directement sur le serveur.

```
git init --bare // À exécuter sur le serveur.
```

Pour se connecter au serveur, la meilleure méthode consiste à utiliser SSH. Ainsi, si vous voulez cloner le dépôt du serveur sur votre ordinateur, vous pouvez écrire quelque chose comme :

```
git clone ssh://utilisateur@monserveur.domaine.com/cheminversledepotgit // À exécuter sur votre machine.
```

Il faudra bien entendu vous identifier en entrant votre mot de passe (sauf si vous avez autorisé votre clé publique).

Génération d'une paire de clés SSH

Pour accéder à vos dépôts Git, vous devez créer et installer des clés SSH. Pour ce faire, il est recommandé d'employer OpenSSH et de générer des clés avec l'outil `ssh-keygen` inclus dans Git, même s'il reste possible de se servir de PuTTY. Ouvrez la console (Git Bash), puis saisissez la commande :

```
ssh-keygen -t rsa
```

Vous êtes invité à saisir un emplacement et une phrase mot de passe. Acceptez l'emplacement par défaut (en principe `C:\Documents and Settings\username\.ssh\` ou `C:\Users\username\.ssh`) en appuyant sur Entrée, puis saisissez une phrase forte..

Ouvrez ensuite le fichier `id_rsa.pub` (situé à l'emplacement défini à l'étape précédente) dans un éditeur de texte. Il contient votre nouvelle clé publique, qui devrait ressembler à ceci :

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAYyA8wePstPC69PeuHFtOwyTecByonsHFAjHbVnZ+h0dpomvLZxUtbknNj3+
c7MPYKqKBOx9gUKV/diR/mIDqsb405MlrI1kmNR9zbFGYAAwIH/Gxt0Lv5ffwaqsz7cECHBbMojQGEz3IH3twEvDf
F6cu5p
00QfP0MSmEi/eB+W+h30NGdQLJCziLDlp409jAfXbQm/4Yx7apLvEmkaYSrb5f/pfvYv1FEV1tS8/J7DgdHUAWo6g
yGUUSZ
JgsyHcuJT7v9Tf0xwiFWOWL9WsWXA9fCKqTeYnYJhHlqfinZRnT/+jKz0OZ7YmXo6j4HymS3RCOqenIX1W6gnIn+e
QIkw==
Mac Pro
```

Vérification de la connexion au dépôt distant

Commencez par vérifier le bon fonctionnement de la connexion à votre dépôt distant. Saisissez la commande suivante dans la console :

```
ssh urldepotdistant
```

À l'authentification, ou lorsque vous tentez de vous connecter au dépôt Git, vous devriez voir un message de ce type :

```
The authenticity of host 'accountname.beanstalkapp.com (204.232.132.2)' can't be
established.
RSA key fingerprint is 30:9a:97:f3:19:4f:d1:6e:28:76:9e:e7:d1:df:2c:31.
Are you sure you want to continue connecting (yes/no)?
```

Vous pouvez saisir `yes` et appuyer sur `Entrée`, ce qui ajoutera le nom d'hôte de votre compte au fichier `known_hosts`. Cette étape n'est à accomplir qu'une seule fois, sauf modification de votre clé publique ou de votre nom d'hôte.

Si l'authentification réussit, vous verrez un message analogue à :

You were successfully authenticated as [emailaddress] in accountname.beanstalkapp.com.

Problèmes potentiels avec Windows 7

Des problèmes ont été signalés lors de la génération de clés SSH sous Windows 7. Essayez alors de la générer sous Windows XP, puis copiez-les à l'emplacement de stockage par défaut des clés (en principe `C:\Documents and Settings\username\.ssh\` ou `C:\Users\username\.ssh`).

Travail avec un serveur

S'il existe un dépôt sur un serveur, vous pouvez partager votre travail avec d'autres personnes,.

Télécharger les nouveautés

La commande `git pull` télécharge les nouveautés depuis le serveur, c'est-à-dire les nouvelles modifications effectuées par d'autres personnes et qui y ont été placées.

Deux cas sont possibles :

- soit vous n'avez effectué aucune modification depuis le dernier pull, dans ce cas la mise à jour est simple (on parle de mise à jour fast-forward) ;
- soit vous avez fait des commits en même temps que d'autres personnes. Les changements qu'ils ont effectués sont alors automatiquement fusionnés aux vôtres.

Si deux personnes modifient en même temps deux endroits distincts d'un même fichier, les changements sont intelligemment fusionnés par Git.

En certaines rares occurrences, deux personnes modifient la même zone de code en même temps. Dans ce cas, Git signale un conflit : il ne peut décider quelle modification doit être conservée ; il vous indique alors le nom des fichiers en conflit. Ouvrez-les avec un éditeur et recherchez une ligne contenant « <<<<<<<<< ». Ces symboles délimitent vos changements et ceux des autres personnes. Supprimez ces symboles et gardez uniquement les changements nécessaires, puis faites un nouveau commit pour enregistrer tout cela.

Envoyer vos commits

Vous pouvez envoyer vos commits sur le serveur qui sert de point de rencontre entre les développeurs.

Avant d'envoyer vos commits, mieux vaut toujours consulter le journal local pour vérifier ce qui va être envoyé :

```
git log -p
```

Vérifiez que tout est conforme et qu'il n'y a pas d'erreur évidente. Une fois les commits envoyés, il sera trop tard pour les corriger. Assurez-vous que personne n'a fait de push avant vous depuis votre dernier pull. Le mieux est d'effectuer un pull avant de faire un push.

Vous envoyez vos commits avec la commande `git push` depuis la console, ou en choisissant `Dépôt distant > Pousser` dans le GUI.

Le changement vers le serveur doit être de type fast-forward, car un serveur est incapable de régler des conflits.

S'il est recommandé de procéder régulièrement à des commits, ce n'est pas le cas des push. Il devrait être exceptionnel de faire plus d'un push par jour. Effectuez trop rapidement vos push vous priverait de la possibilité d'annuler ou modifier un commit : un push est irréversible. Réfléchissez-y donc à deux fois avant d'effectuer un push.

Annuler un commit publié

Puisqu'il est impossible d'annuler un commit envoyé, la seule solution consiste à pousser un autre commit qui effectue exactement l'inverse du commit fautif. Les lignes ajoutées seront supprimées dans ce commit, et inversement. Pour ce faire :

1. Examinez votre journal avec `git log` (ou via le GUI) ;
2. Repérez le numéro du commit à annuler ;
3. Dans la console, saisissez `git revert`, qui va créer un commit « inverse » :

```
git revert numéroducommit
```

Vous devez fournir l'ID du commit à `revert`, mais comme déjà évoqué il suffit de spécifier les premiers chiffres s'ils sont uniques (les 4-5 premiers chiffres devraient suffire).

Vous êtes invités à saisir un nouveau message de commit (un message par défaut est proposé). Enregistrez, vérifiez que tout va bien puis publiez la rectification avec `git push`.

Ajouter ou supprimer une branche sur le serveur

Il est possible d'ajouter des branches sur le serveur. Voici comment procéder :

```
git push origin origin:refs/heads/nom nouvelle branche
```

Vous pouvez ensuite créer une branche locale qui suivra la branche du serveur avec `git branch --track`, comme nous l'avons vu précédemment.

L'inverse est possible : créer une branche locale puis la copier sur le serveur. Pour ce faire, vous devez créer la branche sur le serveur comme expliqué juste au-dessus, mais plutôt que de « tracker » les modifications, modifiez le fichier `.git/config` comme ceci :

- copiez le bloc `[branch "master"]` ;
- remplacez les occurrences de « master » par le nom de votre branche ;
- enregistrez les modifications.

Faites ensuite un `git pull` et un `git push`. Normalement vos modifications devraient être mises à jour sur le serveur.

Pour supprimer une branche sur le serveur :

```
git push origin:heads/nom branche a supprimer
```

À noter que les branches suivies à distance (celles visibles avec `git branch -r`) ne seront pas automatiquement supprimées chez les autres clients. Il faut qu'ils les suppriment manuellement à l'aide de la commande suivante :

```
git branch -r -d origin/nom branche a supprimer
```

Sites de dépôts

Il existe un certain nombre de sites qui permettent de stocker à distance des dépôts GIT : l'expérience montre ne effet qu'il vaut mieux employer ces sites que de tenter de créer un serveur GIT sur Internet. Cela reste envisageable sur un réseau local, mais au prix d'efforts certains. Mieux vaut donc, au moins pour des individus, recourir à un des sites suivants :

- [BitBucket](#) (parrainé par [Atlassian](#)) : Propose de faire des dépôts privé gratuit (jusqu'à 5 utilisateurs sur le même dépôt. Nombre illimité de dépôts publics/privés, gratuit pour 5 utilisateurs, blocs de 1 Go de stockage pour les données de l'utilisateur final
- [Github](#) (parrainé par [GitHub Enterprise](#)) : web-service gratuit qui utilise le logiciel libre [Git](#) et est développé en [Ruby on Rail](#). Nombre illimité de dépôts publics et nombre illimité de collaborateurs publics. Le code de Github est propriétaire. Pas de [quota disque](#)

- [Gitorious](#) : le site web gitorious est un logiciel libre, installable sur son propre serveur. Hébergement gratuit pour les projets open source qui utilisent Git , multiples utilisateurs. Le code de Gitorious est sous la "GNU Affero General Public License". Pas de quota disque
- [Repo.or.cz](#) : Fournit juste un hébergement git. La taille du dépôt ne doit pas dépasser 400 Mo, possibilité de faire du "mirror mode". Parrainé par [UPC](#) en République tchèque
- [Savannah](#) : Héberge les logiciels libres qui fonctionnent sur des systèmes libres et sans dépendance propriétaire.

Conclusion

Git est un outil très complet. Tellement complet qu'il peut parfois se révéler complexe. Comme tout bon outil UNIX, il fait ce qu'on lui demande sans confirmation, ce qui peut être dangereux entre les mains d'un débutant. Commencez donc à l'employer avec prudence et apprenez petit à petit à découvrir ses autres fonctionnalités : vous allez être surpris !

SVN, produit bien implanté, commence toutefois à se faire vieux. La bonne nouvelle est qu'il est possible de migrer facilement à Git avec l'outil `git-svn`. Celui-ci sert également de pont entre un serveur SVN et Git, ce qui permet d'utiliser Git de votre côté même si le projet utilise officiellement SVN !