

Fiche Github

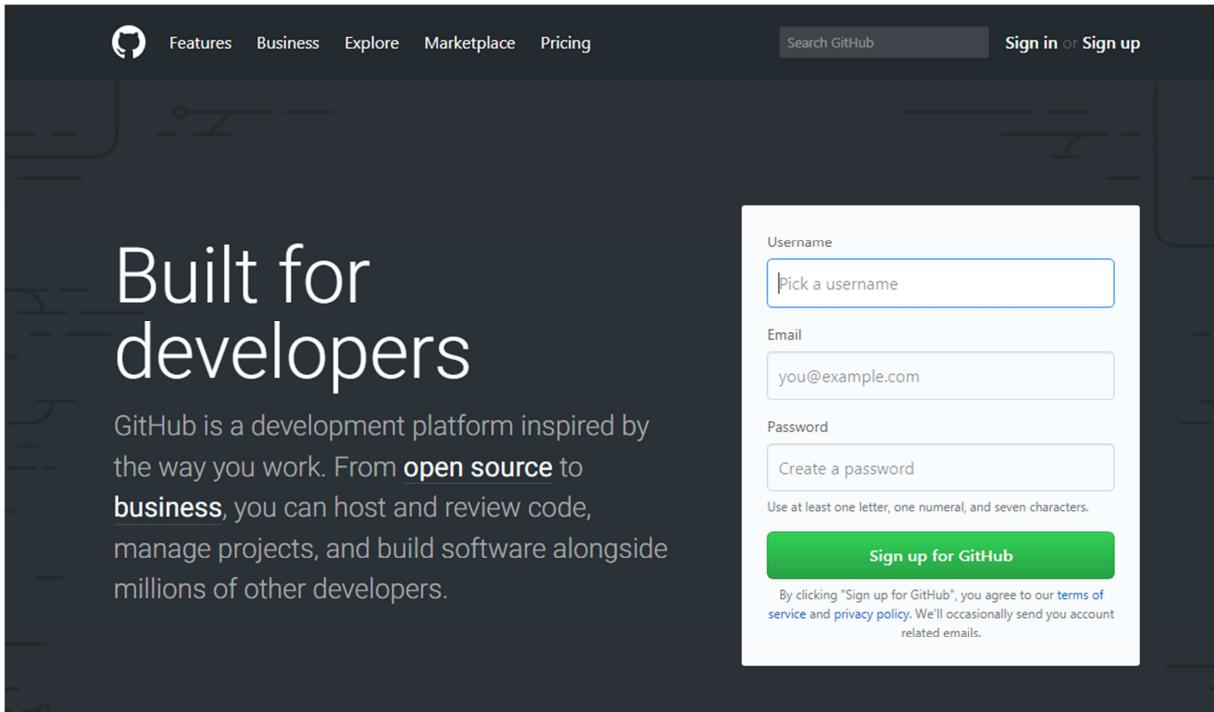
Contenu

Se connecter.....	2
Consulter son Profil	3
Faire un fork (scrum master).....	3
Donner les droits de push aux développeurs.....	4
Protéger la branche Master (scrum master)	5
Git – Configurer le proxy	5
Comment cloner un dépôt (développeurs).....	6
Créer une branche.....	6
Changer de branche	7
Faire un commit	7
« Tirer » les sources – « Pull »	8
Faire un « rebase »	9
Push	11
Faire une Pull Request sur Github (PR)	11
Accepter une Pull Request (scrum master).....	12
Consulter les commits sur github	13
Consulter les branches sur Github	13
Les tickets Github	14

Se connecter

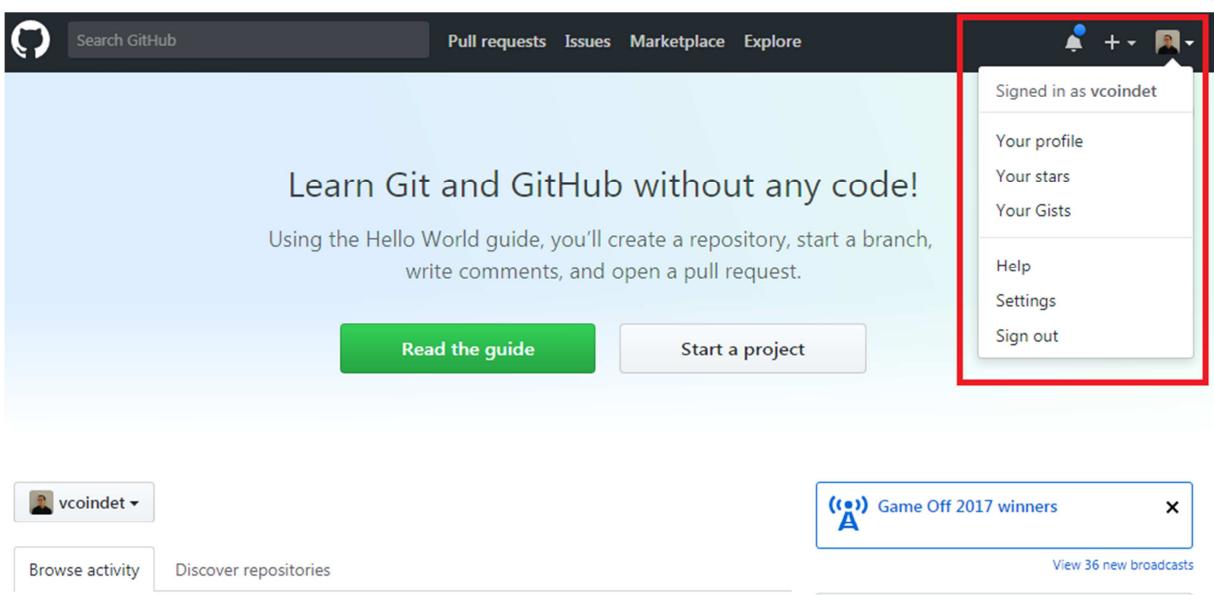
Aller sur <https://github.com/>

Se créer un compte en cliquant sur « Sign up » ou se connecter en cliquant sur « Sign in » si vous avez déjà un compte.



Remplir tous les champs et les soumettre.

Une fois connecté, vous devez arriver sur la page suivante :



Consulter son Profil

En cliquant sur l'icône carrée en haut à droite, il vous est possible d'accéder à votre profil (« Your profile ») et de configurer votre compte github (« Settings » : Modification du nom, changement du mot de passe, changement de l'adresse mail, ajout une clé SSH, etc.)

En allant sur votre profil, vous arrivez sur une page qui ressemble à peu près à ça :

Encadré en rouge ici : ce sont les répertoires ou les dépôts que vous avez cloné sur votre propre compte github. Si ce sont vos premiers pas dans Github, vous n'en avez pas, c'est normal. Pour en avoir, il faut faire un fork !

Faire un fork (scrum master)

Aller sur la page <https://github.com/FormationGPI>

Ici, vous trouverez les projets mis à disposition pour le cours, il y en a quatre (ITOWNS, OPENLAYERS3, LEAFLET, CV). En cliquant sur l'un d'eux (par exemple ITOWNS) vous devez arriver sur la page suivante :

The screenshot shows a GitHub repository page for 'FormationGPI / iTOWNS'. At the top, there's a navigation bar with links for 'This repository', 'Search', 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. On the far right of the header are icons for notifications, adding a repository, and profile. Below the header, the repository name 'FormationGPI / iTOWNS' is displayed, along with a 'Unwatch' button, a star count of 0, and a 'Fork' button which is highlighted with a red box. A progress bar indicates 1 commit, 1 branch, 0 releases, and 1 contributor. Below this, there are buttons for 'Branch: master', 'New pull request', and several file listing buttons: 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The main content area lists files and their details:

File	Description	Last Commit
vcoindet Initialisation du projet	Initialisation du projet	Latest commit ddbd9dc 42 minutes ago
examples	Initialisation du projet	42 minutes ago
README.md	Initialisation du projet	42 minutes ago
README.md	Initialisation du projet	42 minutes ago
index.html	Initialisation du projet	42 minutes ago
itowns.css	Initialisation du projet	42 minutes ago
itowns.js	Initialisation du projet	42 minutes ago
projet_itowns.odt	Initialisation du projet	42 minutes ago
README.md		

En haut à droite, vous pouvez cliquer sur « fork » pour *forker* le projet, c'est-à-dire, rapatrier une copie du répertoire sur votre propre Github . Une fois le bouton cliqué, l'interface de github va vous demander où le forker : choisissez votre propre compte.

En revenant sur votre profil, le projet a du apparaître dans vos « Repositories ».

Donner les droits de push aux développeurs

Lorsque vous êtes dans votre répertoire (toujours dans Github), cliquez sur « Settings »

The screenshot shows a GitHub repository page with the 'Settings' tab highlighted with a red box in the top navigation bar. Below the navigation bar, there are tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The 'Settings' tab is the active one.

Sur cette nouvelle page, cliquez sur « Collaborators » :

Options

Collaborators

Branches

Webhooks

Integrations & services

Deploy keys

Il vous est alors possible de donner les droits de « push » à d'autres développeurs en renseignant au choix son nom d'utilisateur Github, son nom complet ou son adresse mail.

Cette étape est très importante car sans ça, les développeurs seront seulement capables de cloner le dépôt Git sur leur machine, mais ils ne pourront pas pousser leurs modifications sur le dépôt...

Protéger la branche Master (scrum master)

Pour des questions de sécurité, il est conseillé de protéger sa branche master : c'est-à-dire que seules les personnes ayant les droits peuvent fusionner (*merger*) des branches existantes avec la branche master. Cliquer sur « Branches »

Options

Collaborators

Branches

Webhooks

Integrations & services

Deploy keys

Choisissez de protéger la branche master (n'oubliez pas de cocher « Protect this branch »), puis cliquez sur « Save changes ».

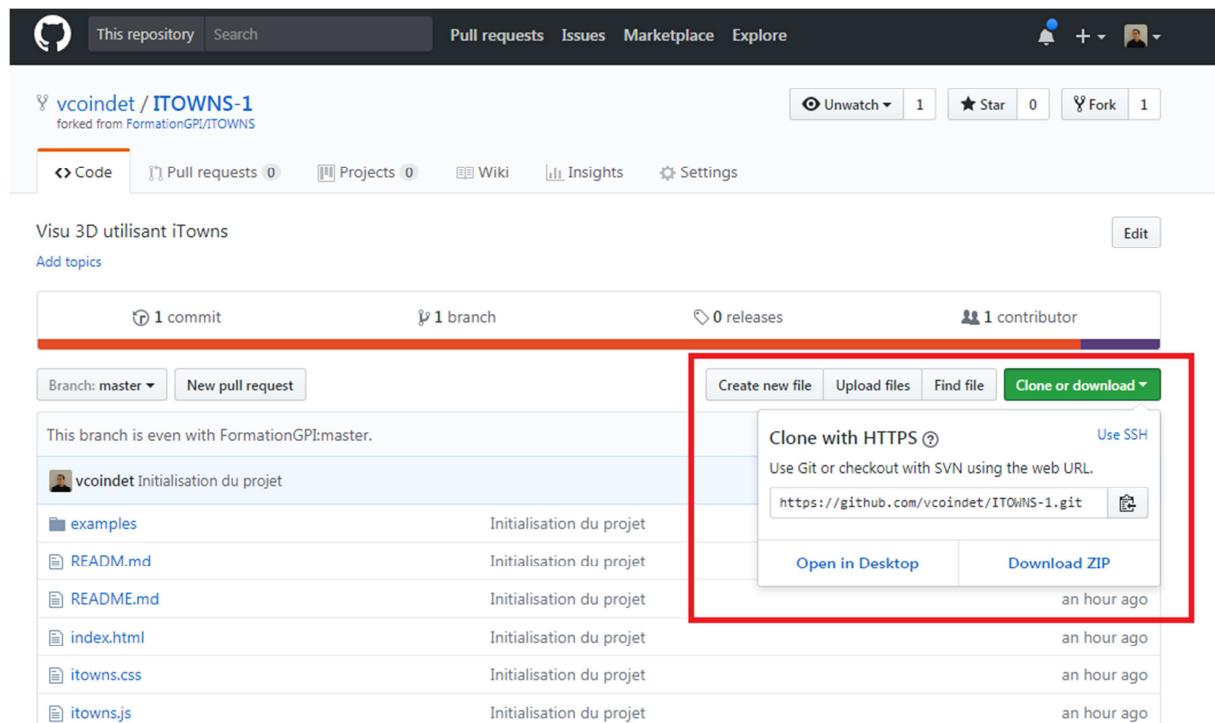
Git – Configurer le proxy

Il peut être nécessaire de configurer le proxy pour Git si jamais ce n'est pas déjà fait. Il vous faut lancer la commande suivante (depuis un terminal, dans n'importe quel répertoire) :

➤ `git config --global http.proxy http://10.0.4.2:3128`

C'est le proxy de l'école (10.0.4.2 :3128).

Comment cloner un dépôt (développeurs)



The screenshot shows a GitHub repository page for 'vcoindet / ITOWNS-1'. The top navigation bar includes links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. On the right, there are buttons for 'Unwatch', 'Star', and 'Fork'. Below the header, the repository name 'vcoindet / ITOWNS-1' is shown, along with a note that it's a fork from 'FormationGPI/ITOWNS'. The main content area displays repository statistics: 1 commit, 1 branch, 0 releases, and 1 contributor. A red box highlights the 'Clone or download' button in the top right corner of the repository details section. This section also contains links for 'Create new file', 'Upload files', 'Find file', and 'Clone with HTTPS' (with an option to use SSH). The 'Clone with HTTPS' link is <https://github.com/vcoindet/ITOWNS-1.git>. Below this, there are 'Open in Desktop' and 'Download ZIP' options. The repository listing shows files like 'examples', 'RFADM.md', 'README.md', 'index.html', 'itowns.css', and 'itowns.js', all updated 'an hour ago'.

Demander à votre scrum master l'adresse du fork qu'il a fait sur propre Github, ce sera votre dépôt de référence sur tout le reste du projet.

Lorsque vous cliquez sur « Clone or download », github vous donne l'adresse du répertoire soit sous le protocole HTTPS, soit sous le protocole SSH (ici <https://github.com/vcoindet/ITOWNS-1.git>).

Ouvrir un terminal et placez-vous à l'endroit où vous souhaitez cloner le dépôt. Dans votre terminal tapez la commande suivante :

➤ `git clone https://github.com/vcoindet/ITOWNS-1.git`

Git va alors cloner le répertoire et il vous sera alors possible de travailler dedans.

Attention : nous ne travaillons que très rarement sur la branche Master, il va donc falloir créer une nouvelle branche en local sur votre propre machine pour y faire vos modifications.

Créer une branche

Dans votre terminal, placez-vous dans le répertoire que vous venez de cloner. Pour créer une nouvelle branche il suffit de lancer la commande :

➤ `git checkout -b [name_of_your_new_branch]`

Cette commande crée une nouvelle branche et vous place directement sur celle-ci.

Lorsque vous clonez un projet, vous clonez toutes les branches qui vont avec, pas seulement la branche master.

Changer de branche

Si vous souhaitez vous placer sur une branche existante en local, il vous suffit de lancer la commande :

```
➤ git checkout [name_of_your_branch]
```

Vous pouvez maintenant commencer à coder ! ;)

Faire un commit

Un commit est un message avant validation des modifications. Ces messages permettent de décrire les modifications apportées au code, mais aussi d'avoir un historique de commits qui permet aux utilisateurs de se reporter rapidement aux modifications antérieurs. Exemple d'historique :

Message	Date	Author
master <1> fix(core): marked dependency is insecure version	08/01/2018 13:45	Gerald Choqueux
feat(3dtiles): add wireframe support for 3dTiles layer, and add wireframe checkbox to 3...	21/12/2017 15:19	Guillaume Liegard
docs(jsdoc) : adding a home page to the API documentation + adding a new template...	20/12/2017 16:57	vcoindet
feat(wfs): Add a debug UI for geometry layer, to change visibility, opacity and toggle w...	20/12/2017 15:26	Guillaume Liegard
fix(wfs): use bigger integer data type for indices array, for THREE.LineSegments. ↴ Prior...	20/12/2017 10:53	Guillaume Liegard
fix(protocolos): stop parsing unnecessarily xbil buffer	18/12/2017 14:38	Gerald Choqueux
examples(wfs): Place points and bus lines on the ground	12/12/2017 16:51	Guillaume Liegard
refactor(wfs): add contour coordinates to the callback that compute altitude.	12/12/2017 16:42	Guillaume Liegard
fix (renderer): set DEBUG after defines definition	18/12/2017 13:35	Adrien Berthet
2.2.0	15/12/2017 20:21	Augustin Trancart
fix (protocols) : stop using layer's option tileMatrixSet in WMS provider	24/10/2017 13:56	Gerald Choqueux
fix (examples): Wrong zoom min in layer example	13/12/2017 15:11	Gerald Choqueux
example: complete the index one with five different layers	12/12/2017 10:43	Adrien Berthet
fix (example): set correct value on init of GUITools	12/12/2017 09:46	Adrien Berthet
feat (core): complete and use FrameRequesters	05/12/2017 15:30	Adrien Berthet
docs (core): fix error to build view documentation	05/12/2017 15:03	Gerald Choqueux
chore: add out/ to .gitignore	05/12/2017 10:35	Adrien Berthet
fix(pointcloud): don't ignore layer.opacity	15/11/2017 11:05	Augustin Trancart
chore(example): enlarge debug ui in pointcloud.html	15/11/2017 15:56	Augustin Trancart
fix(pointcloud): add internal groups before first update	07/11/2017 14:10	Augustin Trancart
fix(pointcloud): honour object3D for pointcloud layers	06/11/2017 16:36	Augustin Trancart
fix (examples): add comments to panorama.html and remove arrow func	21/11/2017 10:51	pierre-eric
fix (core): add quality settings to tweak panorama refinement behavior	21/11/2017 10:51	pierre-eric
fix (geometry): allow layer to override the default 16 segments value	21/11/2017 10:49	pierre-eric

Sur cet exemple, il est possible de se résigner sur n'importe quel commit, et de ne plus avoir les modifications apportées au code après ce commit. Ce genre de situation peut arriver pour retrouver à quel moment un bug a pu s'insérer et le corriger plus rapidement.

Avant de commiter

Avant de commiter, pour voir quels fichiers ont été modifiés, vous pouvez lancer la commande

```
➤ git status
```

S'il y eu une modification, git vous le signal en affichant le message suivant :

```
D:\COURS\GestionProjetInformatique\Projets\iTownsP\iTownsProject (myBranch2)
λ git status
On branch myBranch2
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:   test.js

no changes added to commit (use "git add" and/or "git commit -a")
```

Ici, git nous signale que le fichier test.js a été modifié. Il faut donc l'indexer afin de préparer le prochain commit. La commande à lancer est :

➤ **git add [nom_du_fichier]**

ou

➤ **git add .**

Cette dernière commande permet d'indexer toutes les modifications d'un coup.

Les modifications sont désormais notées comme validées et indexées. Vous pouvez maintenant commiter :

➤ **git commit --m «message du commit»**

Notez qu'il est important de laisser un message car c'est ce message que les autres développeurs vont lire, il donne une indication sur vos modifications.

Après un nouveau git status, vous devriez avoir :

```
D:\COURS\GestionProjetInformatique\Projets\iTownsP\iTownsProject (myBranch2)
λ git status
On branch myBranch2
nothing to commit, working tree clean
```

« Tirer » les sources - « Pull »

Pendant que vous étiez en train de faire vos propres modifications, il est possible que d'autres développeurs aient apporté des modifications, qu'ils les aient déjà poussées sur le serveur Github et que ces modifications soient déjà fusionnées avec la branche master. Votre branche master locale et la branche master du serveur ne sont donc plus identiques... Imaginez que vous poussiez vos modifications et qu'elles soient fusionnées avec la branche master : toutes les modifications de vos collègues seront écrasées, ce qui est dommage. Pour éviter ce genre de problème, il faut d'abord tirer les sources sur votre propre machine afin de mettre à jour votre branche master à jour.

Afin de prendre en compte les modifications qui ont été poussées sur le serveur distant par d'autres développeurs, il est nécessaire de les rapatrier sur sa propre machine. On fait ce qu'on appelle un git pull

➤ **git pull**

Faire un « rebase »

Dans les bonnes pratiques, avant de pousser ses modifications sur le dépôt Github, il est bien de mettre sa branche à jour avec la branche master. En effet, il se peut que des personnes aient déjà poussé des modifications avant vous, modifications que vous n'avez pas forcément encore pris en compte sur votre machine en local.

Il faut donc faire ce qu'on appelle un « rebase », les étapes à suivre sont :

➤ **git add .**
➤ **git commit --m « message »**
➤ **git pull**
➤ **git rebase --i origin/master**

Une fois cette dernière commande lancée, vous arrivez sur l'écran suivant :

The screenshot shows a terminal window titled 'Cmder'. The command entered is:

```
pick 8e24f5e test
pick e541e92 test2
pick 62896c4 test

# Rebase c2da0ff..62896c4 onto c2da0ff (3 commands)
#
# Commands:
# p, pick = use commit, but edit the commit message
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
~ README
~ README
~ commit -m "First commit"
~ git remote add origin git@github.com:tinegna/tinegna.git
~ push -u origin master
~ 
~ this error.

<actionProjetInformatique/Projets/iTownsp/iTownspProject/.git/rebase-merge/git-rebase-todo [unix] (17:15 17/01/2018)1,1 All
<GestionProjetInformatique/Projets/iTownsp/iTownspProject/.git/rebase-merge/git-rebase-todo" [unix] 22L, 694C
```

The terminal shows a list of commits being picked for a rebase. It includes a detailed explanation of the rebase command and its options. The bottom of the terminal shows the command history and the current file being edited.

Git vous propose de choisir les commits à prendre en compte, en principe on les garde tous. Plusieurs options sont possibles pour agir sur les commits :

- **pick** : on prend en compte le commit tel quel
- **reword** : on souhaite modifier le message du commit
- **edit** : on souhaite modifier complètement un commit (pour le couper en deux par exemple)
- **squash** : le commit se fusionné avec le commit précédent

- **fixup** : comme squash, mais utilise le message du commit précédent
- **exec** : lorsqu'on souhaite ajouter une ligne de commande
- **drop** : on supprime un commit.

Dans la pratique, on utilise fréquemment pick, edit et squash.

Une fois que vous avez choisi ou non d'agir sur les commits, taper « :wq » (wq pour write and quit, comme sur l'éditeur de texte VI). Si vous avez choisi de modifier vos commits, git vous demandera de redéfinir vos noms de commit à ce moment-là.

Git va alors commencer le rebase, si tout se passe bien, vous n'aurez rien à faire et le message suivant s'affichera :

```
D:\COURS\GestionProjetInformatique\Projets\iTownsP\iTownsProject (myBranch)
λ git rebase -i origin/master
Successfully rebased and updated refs/heads/myBranch.
```

Dans la pratique, tout ne se passe pas forcément bien lors d'un rebase et vous aurez le message suivant :

```
D:\COURS\GestionProjetInformatique\Projets\iTownsP\iTownsProject (myBranch)
λ git rebase -i origin/master
Auto-merging index.html
CONFLICT (add/add): Merge conflict in index.html
error: could not apply 8e24f5e... test

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".

Could not apply 8e24f5e... test
```

Git a repéré des conflits entre votre code et celui de la branche master. Il faut donc régler ces conflits à l'aide d'un éditeur de texte. Là où il y a des conflits, git a inséré des lignes comme les suivantes :

```
64  <<<<< HEAD
65  =====
66  .....      .....      lalalalalala
67  >>>>> 8e24f5e... test
```

« HEAD » + le numéro du commit.

C'est à vous de juger ce qui doit être pris en compte pour régler le conflit.

Une fois les conflits réglés, pensez bien à enregistrer vos fichiers, puis lancez les commandes suivantes :

- **git add .**
- **git rebase --continue**

Une fois que le rebase est terminé, vous devez avoir ce message :

```
D:\COURS\GestionProjetInformatique\Projets\iTownsP\iTownsProject (HEAD detached at c2da0ff)
└ git rebase --continue
Successfully rebased and updated refs/heads/myBranch.
```

Vous êtes maintenant prêt pour pousser vos modifications sur le serveur distant.

Il est toujours possible d'abandonner un rebase en lançant la commande :

➤ **git rebase --abort**

C'est un retour à la case départ, comme si vous n'aviez jamais lancé de rebase.

Push

Pour pousser votre code (ou votre branche), il suffit de lancer la commande :

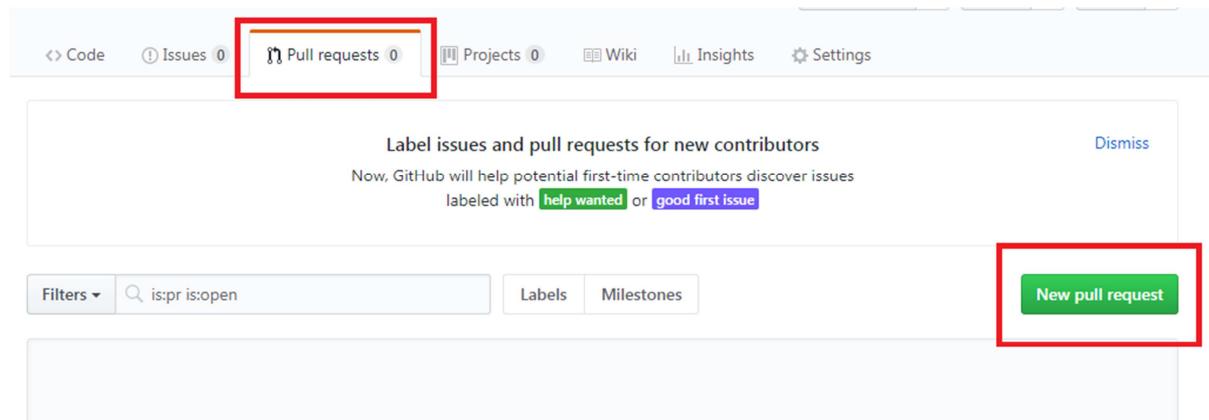
➤ **git push**

Attention, pensez bien à commiter et faire vos rebases avant de pousser votre code.

Faire une Pull Request sur Github (PR)

Une Pull Request est une demande de fusion de sa branche avec une autre branche (souvent la master). Une fois qu'une branche est poussée sur le serveur il est possible de réaliser une telle demande sur github directement :

Lorsque nous sommes dans le projet sur github, il suffit de cliquer sur « Pull requests » puis sur « New pull request ».



Vous devez arriver sur l'écran suivant :

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

A screenshot of the 'Comparing changes' screen. It shows two dropdown menus: 'base: myBranch2' and 'compare: myBranch' (both highlighted with red boxes). To the right of these, a green checkmark icon indicates that the branches are 'Able to merge'. Below this, there is a large green 'Create pull request' button (also highlighted with a blue box). A note below the button says 'Discuss and review the changes in this comparison with others.' and a help icon is in the top right corner.

Vous pouvez choisir sur quelles branches vous souhaitez faire la pull request (à gauche souvent la master, à droite la branche qu'on veut fusionner, celle qui comporte les modifications).

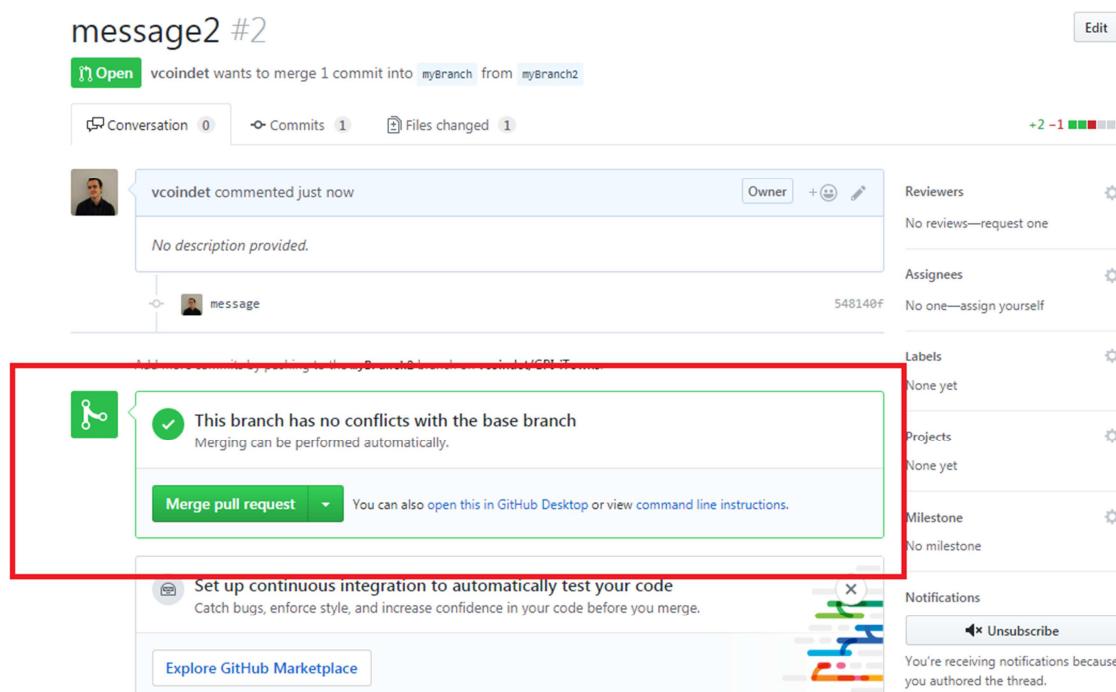
Il vous suffit de cliquer sur « Create pull request » pour créer la demande. Sur l'écran suivant, il vous est possible de donner un titre à la PR et de laisser un message qui explique vos motivations et les modifications apportées. Vous pouvez aussi choisir d'ajouter des reviewers, ou d'assigner la review de code à une personne.

Lorsqu'une pull request est soumise, des tests d'intégration continue peuvent être réalisés automatiquement à ce moment à l'aide d'outils comme Travis. Dans ce cours, nous avons choisi de faire l'intégration continue avec l'outil Jenkins, donc nous ne verrons pas Travis.

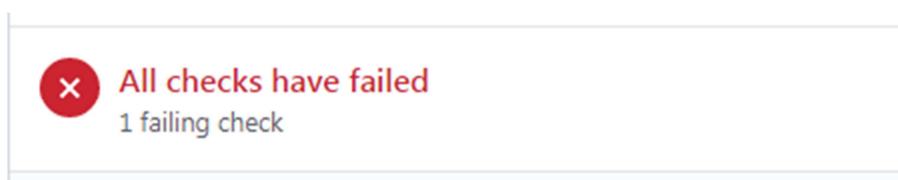
Accepter une Pull Request (scrum master)

Une fois la Pull request créée, les personnes habilitées peuvent l'accepter ou non...

Si tous les tests sont bons, que Github signale que la PR peut être mergée sans créer de conflit, le scrum master peut alors accepter la PR en cliquant sur « Merge Pull Request ».



Si jamais les tests ne sont pas passés, Github le signale en disant « All checks have failed » ou « Some checks have failed »

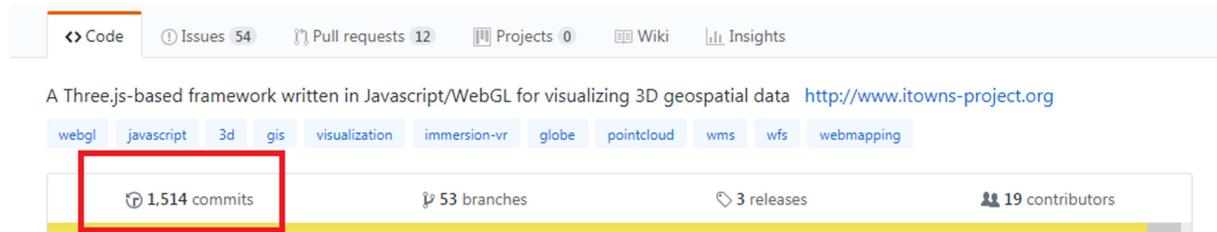


Une fois la PR acceptée, github se charge de fusionner (*merger*) les deux branches, c'est-à-dire insérer les modifications de notre branche de développement dans la branche master.

Lorsqu'un merge a été fait sur la branche master, il est de bonne rigueur d'effectuer un rebase sur sa propre branche de développement pour être sûr d'être bien à jour avec le code sur le serveur.

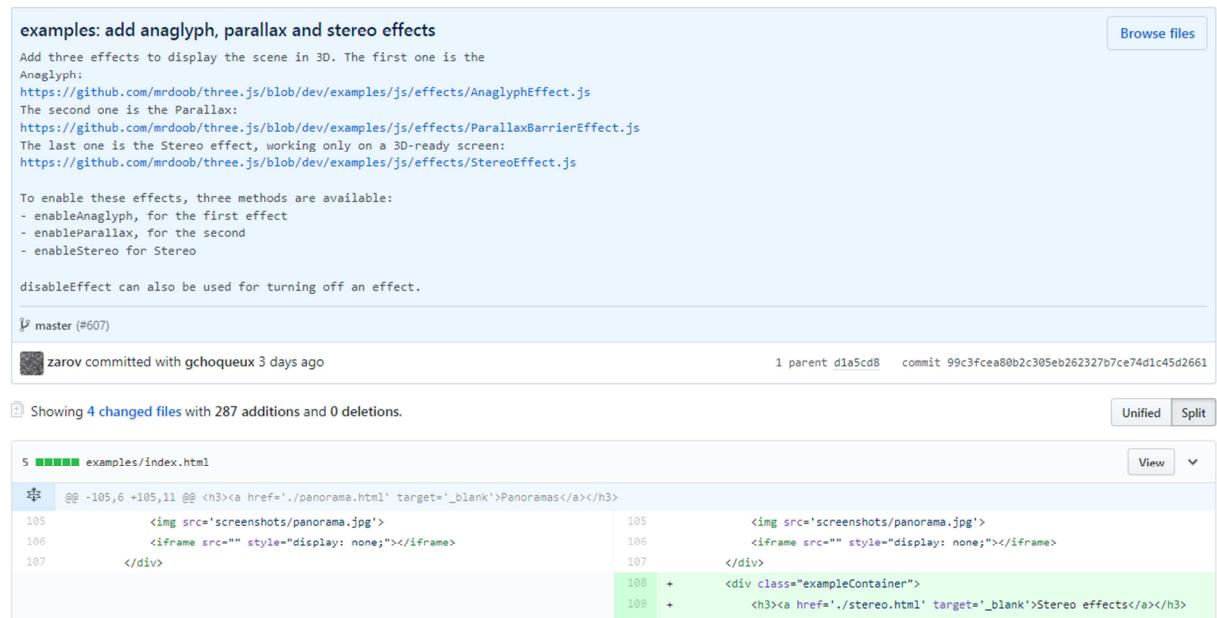
Consulter les commits sur github

Github vous propose de consulter tous les commits qui ont été poussés sur la plateforme : sur la page d'accueil du projet, vous pouvez cliquer sur l'onglet « commits » :



A screenshot of a GitHub project's main page. At the top, there are tabs for 'Code', 'Issues 54', 'Pull requests 12', 'Projects 0', 'Wiki', and 'Insights'. The 'Code' tab is currently selected. Below the tabs, a brief description of the project is shown: 'A Three.js-based framework written in Javascript/WebGL for visualizing 3D geospatial data' with a link to <http://www.itowns-project.org>. Underneath this, there is a horizontal bar with several tags: 'webgl', 'javascript', '3d', 'gis', 'visualization', 'immersion-vr', 'globe', 'pointcloud', 'wms', 'wfs', and 'webmapping'. Below the tags, a summary box contains the text '1,514 commits' (which is highlighted with a red box), '53 branches', '3 releases', and '19 contributors'.

Vous arrivez alors sur une page répertoriant tous les commits, et en cliquant sur un, vous pouvez voir apparaître les messages associés à ce commit, et toutes les différences que ce commit a apporté.



A screenshot of a GitHub commit page. The commit is titled 'examples: add anaglyph, parallax and stereo effects'. It was made by 'zarov' and pushed 3 days ago. The commit message is: 'master (#607) zarov committed with gchoqueux 3 days ago'. The commit hash is 'commit 99c3fcea80b2c305eb262327b7ce74d1c45d2661'. Below the commit message, there is a 'Browse files' button. The main part of the page shows a diff view of the file 'examples/index.html'. The diff shows changes from line 5 to 109. Lines 108 and 109 are highlighted in green, indicating they are new additions. The changes are as follows:

```
5 examples/index.html
diff --git a/examples/index.html b/examples/index.html
@@ -105,6 +105,11 @@ <h3><a href='./panorama.html' target='_blank'>Panoramas</a></h3>
105      <img src='screenshots/panorama.jpg'>
106      <iframe src="" style="display: none;"></iframe>
107  </div>
108 + <div class="exampleContainer">
109 +   <h3><a href='./stereo.html' target='_blank'>Stereo effects</a></h3>
```

C'est très pratique pour consulter rapidement les modifications qui ont été faite sur le code.

Consulter les branches sur Github

De la même façon, vous pouvez aussi visualiser toutes les branches poussées sur le serveur en cliquant sur « Branches ».

A screenshot of a GitHub repository page for a project named "itowns". The top navigation bar includes links for Code, Issues (54), Pull requests (12), Projects (0), Wiki, and Insights. Below the navigation, a brief description states: "A Three.js-based framework written in Javascript/WebGL for visualizing 3D geospatial data" followed by a link to <http://www.itowns-project.org>. A horizontal bar below the description lists various tags: webgl, javascript, 3d, gis, visualization, immersion-vr, globe, pointcloud, wms, wfs, and webmapping. A red box highlights the "53 branches" link, which is part of a larger section showing 1,514 commits, 3 releases, and 19 contributors.

Vous pouvez alors choisir une branche existante et visualiser le code de cette branche directement sur Github en cliquant dessus.

Les tickets Github

Très souvent les projets hébergés sur Github sont open source, ce qui permet de développer une communauté d'utilisateurs et développeurs autour du projet. Github a donc inséré un système de ticket sur sa plateforme afin que les utilisateurs puissent signaler des bugs, demander de l'aide, demander des conseils, etc.

Le but est de faire vivre le projet au maximum, et lorsqu'un bug est signalé les développeurs peuvent le corriger. Lorsque le bug est résolu, le développeur peut prévenir et dire dans quelle branche le bug est corrigé. Si la solution convient, le ticket peut alors être fermé soit par la personne qui a ouvert ce ticket, soit par les développeurs.

Ces tickets peuvent souvent être une réécriture des tickets Redmine, mais ils sont vraiment orientés, voire même écrits par les développeurs, et sont souvent orientés technique, contrairement à un ticket Redmine qui est souvent rédigé par un client ou le Scrum master qui ne connaît pas forcément les aspects techniques.