

Projet Techno (4TIN403U)



Git

Moodle : <https://moodle1.u-bordeaux.fr/course/view.php?id=9142>

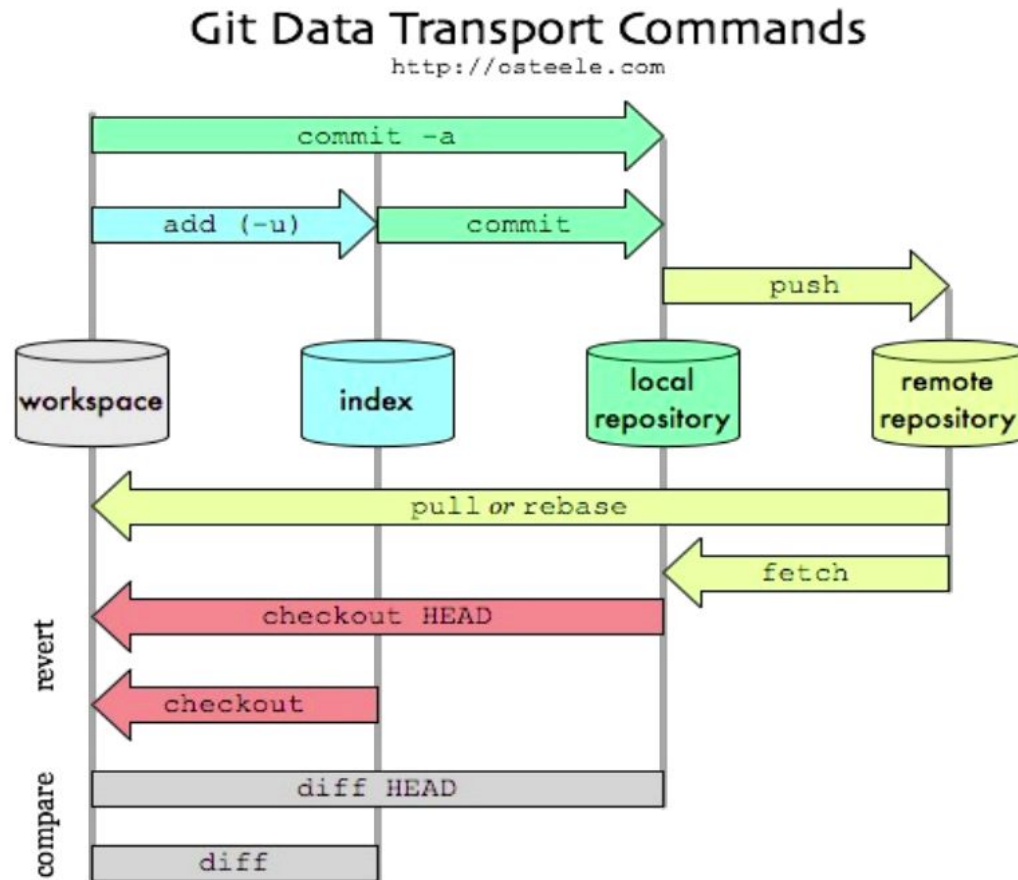
Responsable : Aurélien Esnard

Introduction à Git

Un logiciel de gestion des versions d'un projet (code, doc, ...)

- Développer un logiciel à plusieurs, c'est difficile !
 - Sauvegarder l'historique des versions du code, release, ...
 - Fusionner les différentes contributions, sans rien perdre...
- Développé en 2005 par Linus Torvalds (le créateur de Linux), sous licence GPL
- Décentralisé
 - Chaque machine possède une copie locale du dépôt distant (.git/)
 - Possibilité de travailler sur le dépôt local, même sans le réseau
- Généralement il y a 1 dépôt central et des dépôts secondaires...
- Le gestionnaire de version actuellement le plus populaire !
- Un outil très puissant, mais complexe !

Git : Schéma Global



Le Gitlab du CREMI

Comment créer son propre projet Git ?

⇒ <https://gitlab.emi.u-bordeaux.fr/>

New project › Create blank project

Project name

My awesome project

Project URL **Project slug**


https://gitlab.emi.u-bordeaux.fr/auensnard/ my-awesome-project


Want to house several dependent projects under the same namespace? [Create a group.](#)


Project description (optional)

Description format

Visibility Level ?

☒  **Private**
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.

☐  **Internal**
The project can be accessed by any logged in user except external users.

☐  **Public**
The project can be accessed without any authentication.

☒ **Initialize repository with a README**
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project Cancel

Donnez un nom à votre projet et choisir le niveau de visibilité : *private*, *internal*, *public*.




Le Gitlab du CREMI

Administrez votre projet...

⇒ <https://gitlab.emi.u-bordeaux.fr/auesnard/my-awesome-project>

Ajoutez de nouveaux membres...

The screenshot shows the GitLab web interface for a project named 'My awesome project'. The left sidebar contains a navigation menu with options like Project information, Activity, Labels, Members (selected), Repository, Issues, Merge requests, CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Settings. The main content area is titled 'You can invite a new member to My awesome project or invite another group.' and features an 'Invite member' form. The form includes a search bar for 'GitLab member or Email address', a 'Select a role' dropdown menu (currently showing 'Guest'), an 'Access expiration date' field, and 'Invite' and 'Import' buttons. Below the form, there is a 'Members' section with a filter dropdown and a table listing existing members.

Account	Source	Access granted
 Aurelien Esnard It's you @auesnard	Direct member	20 minutes ago by Aurelien Esnard

Donnez un rôle aux nouveaux membres :
guest/reporter < developper < maintenir

- *guest/reporter* : droit limité sur le projet (pas de push)
- *developper* : droit limité en administration (ne peut pas ajouter de nouveau membres par exemple)
- *maintenir* : le rôle le plus élevé que vous pouvez attribuer

Le Gitlab du CREMI

Comment récupérer son projet ?

Faire un clone en HTTPS ✗ (non supporté au CREMI)

```
$ git clone https://gitlab.emi.u-bordeaux.fr/auesnard/my-awesome-project.git
Cloning into 'bis'...
Username for 'https://gitlab.emi.u-bordeaux.fr': auesnard
Password for 'https://auesnard@gitlab.emi.u-bordeaux.fr': xxxxxxxxxxxx
remote: HTTP Basic: Access denied. Fatal: Authentication failed.
```

Faire un clone en SSH ✓

```
$ git clone git@gitlab.emi.u-bordeaux.fr:auesnard/my-awesome-project.git
Clonage dans 'my-awesome-project'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Réception d'objets: 100% (6/6), fait.
```

Prérequis : Il faut ajouter sa clé publique SSH ~/.ssh/id_rsa.pub dans les Préférences de son compte Gitlab.

Le Gitlab du CREMI

Générer sa clé SSH, si ce n'est pas déjà fait...

```
$ ssh-keygen
```

```
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (~/.ssh/id_rsa):
```

```
Enter passphrase (empty for no passphrase):
```

```
Enter same passphrase again:
```

```
Your identification has been saved in ~/.ssh/id_rsa
```

```
Your public key has been saved in ~/.ssh/id_rsa.pub # <-- la clé publique
```

```
$ cat ~/.ssh/id_rsa.pub
```

```
ssh-rsa
```

```
AAAAB3NzaC1yc2EAAAADAQABAAQGC4uJaLtIf+d4KmnLjFrol00WQV1xFErxvSdy8HotObXsKfpIi7Qz93MFC4um7  
SvFWxeXVQwm2gDZf8sfmz5NSZO4+RELMxcteUV56JF6XD5BCG4lovyLFUWsiPxkaOHr3Eng3T1M5UrINudIoEsOhgp7  
9AxjkqpBuFajc2YUtp80Zc/9vioZvCDLWDkcoyQgAFhwzbh8R0JW5bFD0ovVU/UJYnwB4UIs80G83k9ZgwLWMPiqh+2  
n9l7QWDsKnCK7CeP3ESnTEVlHQQtVEP93FInlsfU34VCso7rP1U7JDEfQAtbL3gWqn0/wdDsWFKC6LjKHT91f3MdiLQR  
mq5N7ySRlG7lAsTYkzgBpmj/LA/tSciU10bsYlud+rKQa7z+WbbT/VMHEjA9o7GsBE0mrXSfYK2kb6p+fdm3nmSSg1H  
i+WXw4HgnL2fHuLh14uZO2spKSfZTZkjJWUIjq2u3114+6RZm+1NT7Y3PuRyd5uhxKjNfzsmFpUES9ONsdK3lIGM=  
toto@prout
```

→ Faire un copier / coller du texte de la clé...



Le Gitlab du CREMI

Ajoutez sa clé publique à son compte sur Gitlab

The image shows a screenshot of the GitLab web interface. On the left, the 'My awesome project' page is visible, showing a commit for 'Update README.md' by Aurelien Esnard. On the right, the 'User Settings' page is open, specifically the 'SSH Keys' section. A large grey arrow points from the project page to the SSH Keys page.

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

To add an SSH key you need to [generate one](#) or use an [existing key](#).

Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/id_ed25519.pub' or '~/.ssh/id_rsa.pub' and begins with 'ssh-ed25519' or 'ssh-rsa'. Do not paste your private SSH key, as that can compromise your identity.

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQCC4uJaLtlf+d4KmnLjFroLO0WQV1xferxvSdy8HotObXsK
fpli7Qz93MFc4um7SvFWxeXVQwm2gdZf8sfmz5NSZO4+RELmxcTeUV56JF6XD5BCG4lovyLFUW
siPka0Hr3Eng3T1M5UrnNudloEsOhgp79AxjkpBuFajc2YUtp80Zc/9vtoZvCDLWDkcoyQgAFhwzb
h8R0JW5bFD0ovVU/UJYnwb4Uls80G83k9ZgwLWMPigh+2n9l7QWDsKnCK7CeP3ESnTEVIHQIVE
P93FlnIsfU34Vcso7rP1U7JDEfQAtbL3gWqn0/wdDsWFKC6LJKHT91f3MdiLQRmq5N7ySRIG7lAsT
YkzqBpmj/LA/tSciU10bsY1ud+rKQa/z+Wbbt/VMHEIA9o7GsBE0mrXSfYK2kb6p+fdm3nmSSg1H
i+WXw4HgnL2fHuLh14uZO2spKSfZTZKjJWUliq2u3114+6RZm+1NT7Y3PuRyd5uhxKjNfzsmFpUE
S9ONsdK3lIGM= toto@prout
```

Title

toto@prout

Expires at

mm/dd/yyyy

Give your individual key a title. This will be publicly visible.

Key can still be used after expiration.

Add key

Git : Les Bases

Récupération d'un projet Git

```
git clone git@gitlab.emi.u-bordeaux.fr/auesnard/test
```

Un peu de configuration avant de travailler...

```
git config --global user.name "Prénom Nom"  
git config --global user.email " prenom.nom@etu.u-bordeaux.fr "  
git config --global -l
```

Avant toute commande...

```
git status
```

Git : Les Bases

Ajout ou modification d'un ou plusieurs fichiers et commit...

```
git add file1 file2
git add file3
git commit -m "my message"
```

Suppression d'un fichier ou d'un répertoire

```
git rm file1
git rm -r dir1
git commit -m "my message"
```

Synchronisation avec le dépôt distant (*origin*)

```
git pull          # récupération des autres commits, fusion...
git push          # envoi de nos derniers commits
```

Git : Les Bases

En cas de conflit...

```
git pull          # échec à la fusion !
```

On édite le(s) fichier(s) responsable(s) du conflit, en comparant les deux versions conflictuelles...

```
...
<<<<<<< HEAD
ma version HEAD dans le dépôt local
=====
l'autre version qui vient du dépôt remote
>>>>>>> <commit>
...
```

On corrige manuellement, puis on fait un nouveau commit / push :

```
git commit -m "correction du conflit brol"
git push
```

Git : Les Bases

Log

```
git log                # basique
git log --graph --oneline --all  # un peu plus joli...
```

Diff

```
git diff <commit> <commit> [file] # comparaison explicite entre deux commits
git diff <commit>           [file] # comparaison du workspace avec un commit
git diff                   [file] # comparaison du workspace avec HEAD
```

avec <commit>, un identifiant de commit qui peut être :

- un ID explicite comme 83109ce0d335a...4e8bd58f tel qu'affiché par git log ;
- HEAD, qui désigne le dernier commit dans la branche courante du dépôt local ;
- HEAD~1, qui désigne l'avant dernier commit dans la branche courante du dépôt local ;
- un nom de branche sur le dépôt local comme master ;
- un nom de branche sur le dépôt distant (origin), comme origin/master.

Git : Les Bases

Synchronisation de l'index, mais pas des fichiers...

```
git fetch                # on récupère les infos sur origin/master
git diff origin/master    # on compare la version locale avec origin/master
```

On peut ainsi anticiper des conflits !

Retrouver une ancienne version...

```
git checkout <commit>    # detached HEAD !!!
git checkout master       # revenir à la version courante...
```

Retrouver un ancien fichier...

```
git checkout -- <file>    # ou git restore <file>
git log --name-only
git checkout <commit> -- <file>
```

Git : Oups !

Supprimer mon dernier commit local (non publié)

```
git reset --hard HEAD~1      # ... en effaçant les modifications
git reset HEAD~1             # ... sans effacer les modifications
```

Supprimer mon dernier commit, après publication !

```
git reset --hard HEAD~1      # supprimer le dernier commit en local
git push --force origin master  # forcer la synchronisation...
```

```
# Attention, ça met le bazarre chez les autres...
git fetch -a -p ; git reset --hard origin/master
```

Solution alternative moins dangereuse...

```
git revert HEAD~1            # ajout d'un commit annulant l'avant-dernier commit
git push
```

Git : les Branches

Branches locales & distantes

```
master    ----> origin/master    # default branch
dev       ----> origin/dev        # branch to develop new features
hotfix                               # local branch for bug fix
```

Création d'une branche locale, à partir du commit courant...

```
git branch hotfix                # à partir master
git branch
    hotfix
* master
```

Changement de branche

```
git switch hotfix
git branch
* hotfix
    master
```

Git : les Branches

Correction du bug dans la branche locale

```
# je corrige facilement mon bug...
git add file1 ; git commit -m "bug fix #1"
# mince, y'a toujours un bug ?!
```

```
# le lendemain, je corrige un autre bug...
git add file2 ; git commit -m "bug fix #2"
# c'est enfin réparé !
```

Application des corrections...

```
git switch master          # je bascule sur la branche master
git diff hotfix            # comparaison entre master et hotfix
git pull                  # je synchronise master avec origin/master
git merge hotfix          # je fusionne la branche hotfix dans master
git log                   # deux nouveaux commits : bug fix 1 & 2
git push                  # je publie mes corrections pour tous
git branch -d hotfix      # je supprime la branche locale
```


Git : les Branches

Utiliser une branche distante comme tag d'une version...

```
git branch v1          # création de la branche locale v1
git push -u origin v1   # publication de la branche distante origin/v1
```

Récupérer une branche distante...

```
git pull               # on synchronise les branches origin/*
git branch -a          # affiche aussi les branches distantes
git switch v1          # branche locale implicitement créée
```

Utiliser une branche distante pour tester une idée...

```
git branch dev         # création de la branche locale dev
git push -u origin dev  # publication de la branche distante origin/dev
git switch dev         # on travaille dedans...
...                   # commits
git push               # publication des commits dans origin/dev
git switch master      # on revient sur master pour faire autre
chose...
```

Git : les Branches

Suppression d'une branche locale

```
git branch -d hotfix
```

Suppression d'une branche distante

```
git push origin --delete dev          # ne supprime pas la branche locale
```

Afficher toutes les branches

```
git fetch -a -p          # mettre à jour son index (supprime les branches  
mortes)  
git branch -a
```

Réécrire l'histoire plus simplement...

```
git switch dev  
git rebase master          # réécrit l'histoire de dev au dessus de master...
```

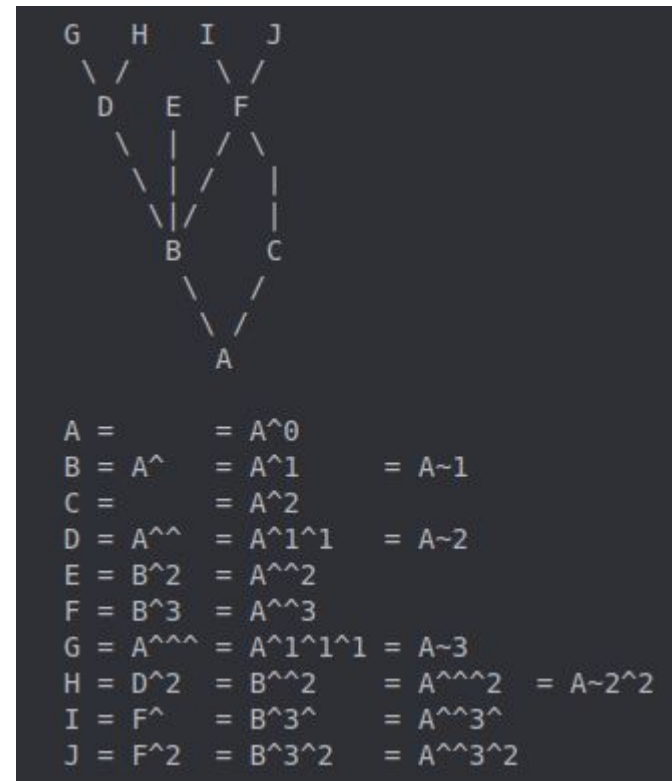
Pour aller plus loin... → <https://learngitbranching.js.org/>

Revision : HEAD~2^3 ou HEAD~3^2 ?

Les opérateurs `<rev>~<n>` et `<rev>^<n>` permettent de naviguer dans l'historique de votre dépôt Git à partir d'une certaine *revision* `<rev>`, comme *HEAD*, *master*, *origin/master*, *61fcec2* (*hash* au format SHA-1), ...

Par exemple :

```
A    <-- HEAD
| \
B | <-- HEAD~1 = HEAD^1 (this first ancestor of HEAD)
| C <-- HEAD^2 (the second ancestor of HEAD)
D | <-- HEAD~2 = HEAD^1^1
| /
E    <-- HEAD~3
```



Git : Divers Trucs

Le fichier .gitignore

Le fichier .gitconfig

Nettoyer tous les fichiers n'appartenant pas à son dépôt

```
git clean -i -x -d
```

Marre de taper son mot de passe...

```
git config credential.helper store
```

Git : Conclusion

Un outil puissant, mais complexe...

Les bonnes pratiques

- Ne commitez que des versions qui compilent correctement (sans erreurs et sans warnings)
 - sinon vous risquez de bloquer les autres...
- Commentez chaque commit !
 - C'est important pour le suivi des modifications...
 - Quel commit a corrigé quelle erreur ?
 - Quel commit a introduit telle ou telle fonctionnalité ?
- « Commit Early, Commit Often »
 - Mieux vaut plusieurs petits commits logiques & cohérents, qu'un seul gros commits...
 - Cela permet de mieux tracer les modifications et de diminuer les conflits !

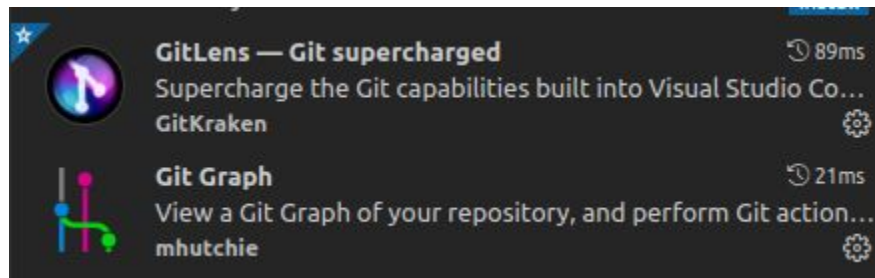
Utilisation de Git dans VS Code

Installer les Extensions : *Git Lens* et *Git Graph*

- La synchronisation pull & push en un clic...
- Changement, création et publication de branche...
- Faire un commit rapidement avec <ctrl>+<enter>
- Consulter les logs et l'historique avec *Git Graph*
- Visualiser les diffs...
- Résoudre les conflits...



Tutoriel : <https://code.visualstudio.com/docs/editor/versioncontrol>



Démo Git

Démo avec deux utilisateurs sur Gitlab

Laptop: orel@prout -> Gitlab: auesnard (Aurelien Esnard, owner)

Laptop: toto@prout -> Gitlab: _mmoodle (Moodle Manager)

Création d'un projet privé sur Gitlab

- <https://gitlab.emi.u-bordeaux.fr/pt2/demo> (projet *demo*, groupe *pt2*)
- Ajout dans le projet (ou groupe) du membre *toto* comme *maintainer*

Récupération d'une copie locale pour chaque utilisateur (SSH)

```
orel@prout:~$ git clone git@gitlab.emi.u-bordeaux.fr:pt2/demo.git
```

```
orel@prout:~$ git config --global user.name "Aurelien Esnard"
```

```
orel@prout:~$ git config --global user.email "aurelien.esnard@u-bordeaux.fr"
```

Vérifiez votre identité...

```
$ git config --global -l
```

Démo Git dans VS Code

Support Git de base + extensions Git Lens et Git Graph...

The screenshot displays the Visual Studio Code interface with the Git Graph extension and Git Lens extension. The top panel shows the file explorer with 'README.md' open. The middle panel shows the Git Graph view, which displays a commit history table. The bottom panel shows the Git Lens view, which displays the commit history for the current file.

Git Graph View:

Graph	Description	Date	Author	Commit		
	Uncommitted Changes (1)	11 Oct 2021 23:36	*	*		
main	origin	origin/HEAD	2nd commit	11 Oct 2021 23:30	Moodle Manager	5e5ca9b6
	1st commit	11 Oct 2021 23:28	Aurelien Esnard	d08f5aa9		
	Initial commit	11 Oct 2021 23:00	Aurelien Esnard	88a66152		

Git Lens View:

1 # demo
2
3 Ceci est un commit de Aurelien Esnard.
4
5+ Ceci est un commit de Moodle Manager.
6+
7+