Découvrir Git et partager son code avec Github

<u>Objectif :</u> Comprendre comment fonctionne le suivi de versions pour gérer l'historique de son code.

Découvrir Git et partager son code avec Github

1. Introduction à Git
2. Exemples de workflow
3. Premier commit
4. Passer du local au collaboratif (Github)
5. Introduction à Github
6. Créer son compte et son premier repository Github
7. Comment nommer ses commits ?
8. Les commandes de base sur Git
9. Les branches + exercice
10. Les conflits Git
11. Installer les extensions GitLens & Git Graph sur VS Code
12. Le Fork & Pull Request
13. Le pull
14. Git Pages
15. Exercice
16. Ressources
17. QCM

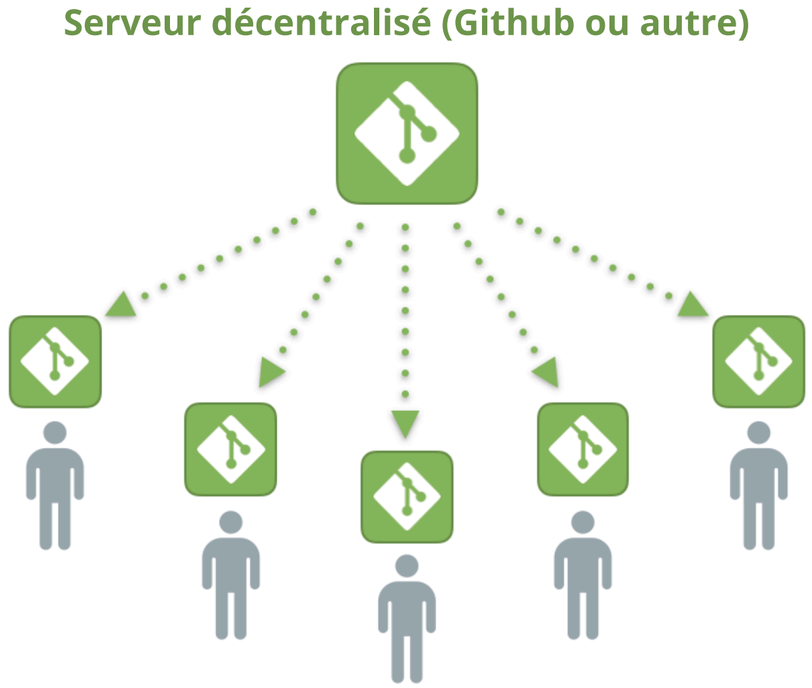
Introduction à Git

👉 **Git, c’est quoi ?**

Git est un outil qui permet de faire du versioning : il enregistre toutes les modifications apportées à un projet sous forme de snapshots (captures d’état). Chaque modification est enregistrée dans un historiquee. Git garde un historique complet de toutes les versions et peut résoudre les conflits.

Le versioning :

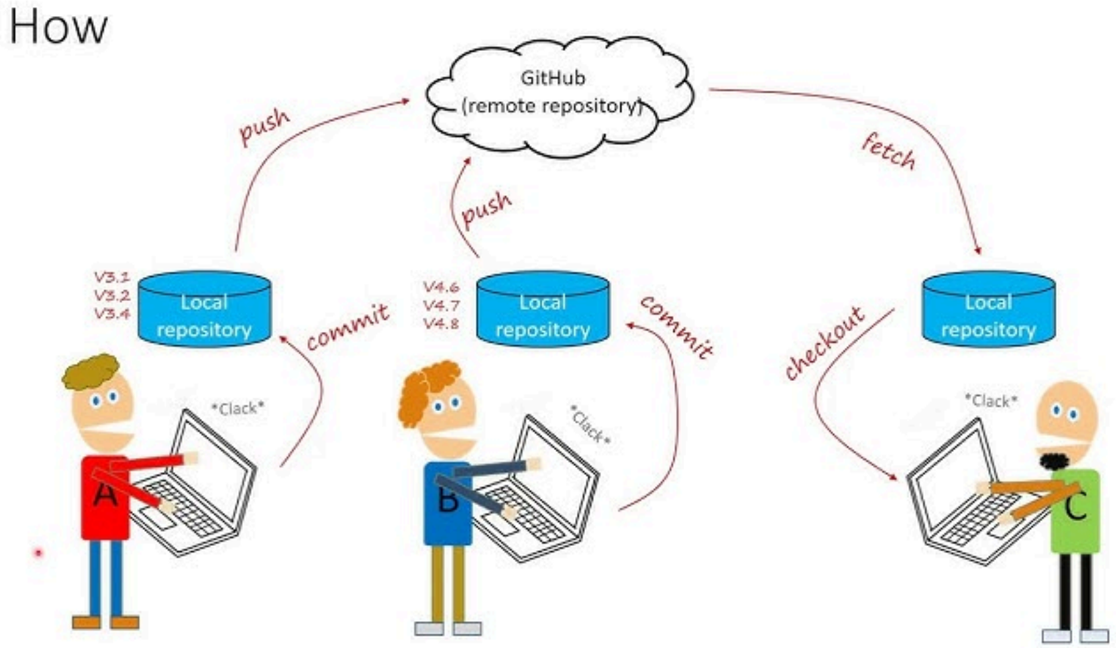
- Qui a modifié quoi ?
- Pourquoi il a été modifié ?
- Historique des modifications
- Résolution des conflits par fusion
- Possibilité d’avoir des branches



Introduction à Git

👉 **Pourquoi utiliser Git ?**

1. On peut savoir qui a fait quoi et quelles modifications ont été apportées
2. Chaque modification est enregistrée, ce qui permet de revenir en arrière si un bug apparaît.
3. Plusieurs développeurs peuvent travailler sur le même projet sans écraser le travail des autres.



Introduction à Git

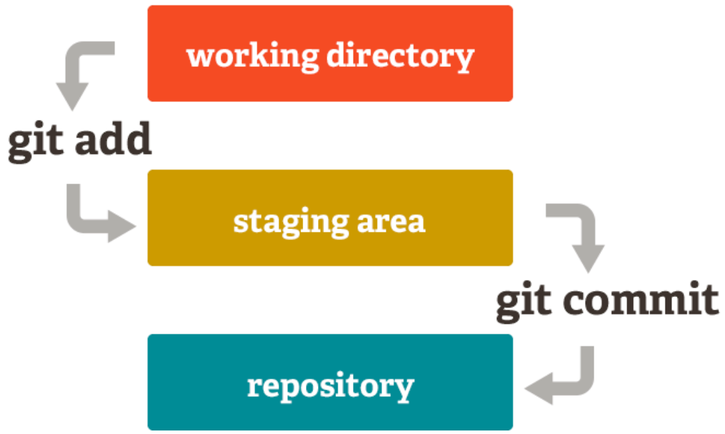
Comment fonctionne Git ?

Un fichier suivi par Git peut être dans trois états :

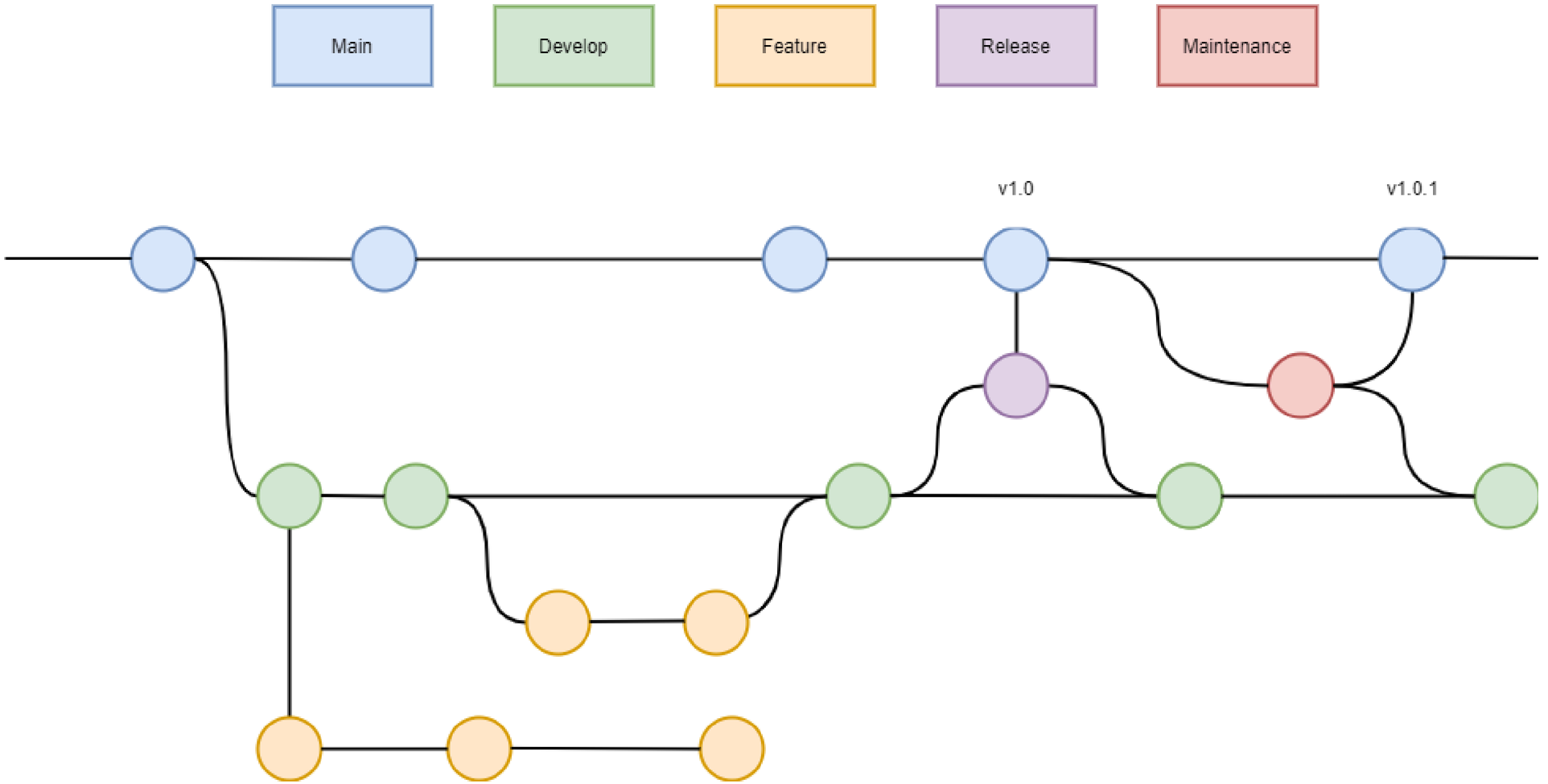
1. **Untracked** (Non suivi) → Fichier ajouté au projet mais pas encore suivi par Git.
2. **Staged** (Indexé) → Fichier ajouté à la zone de staging via **git add**.
3. **Committed** (Enregistré) → Fichier définitivement enregistré dans l'historique via **git commit**.

Un projet Git est composé de plusieurs éléments :

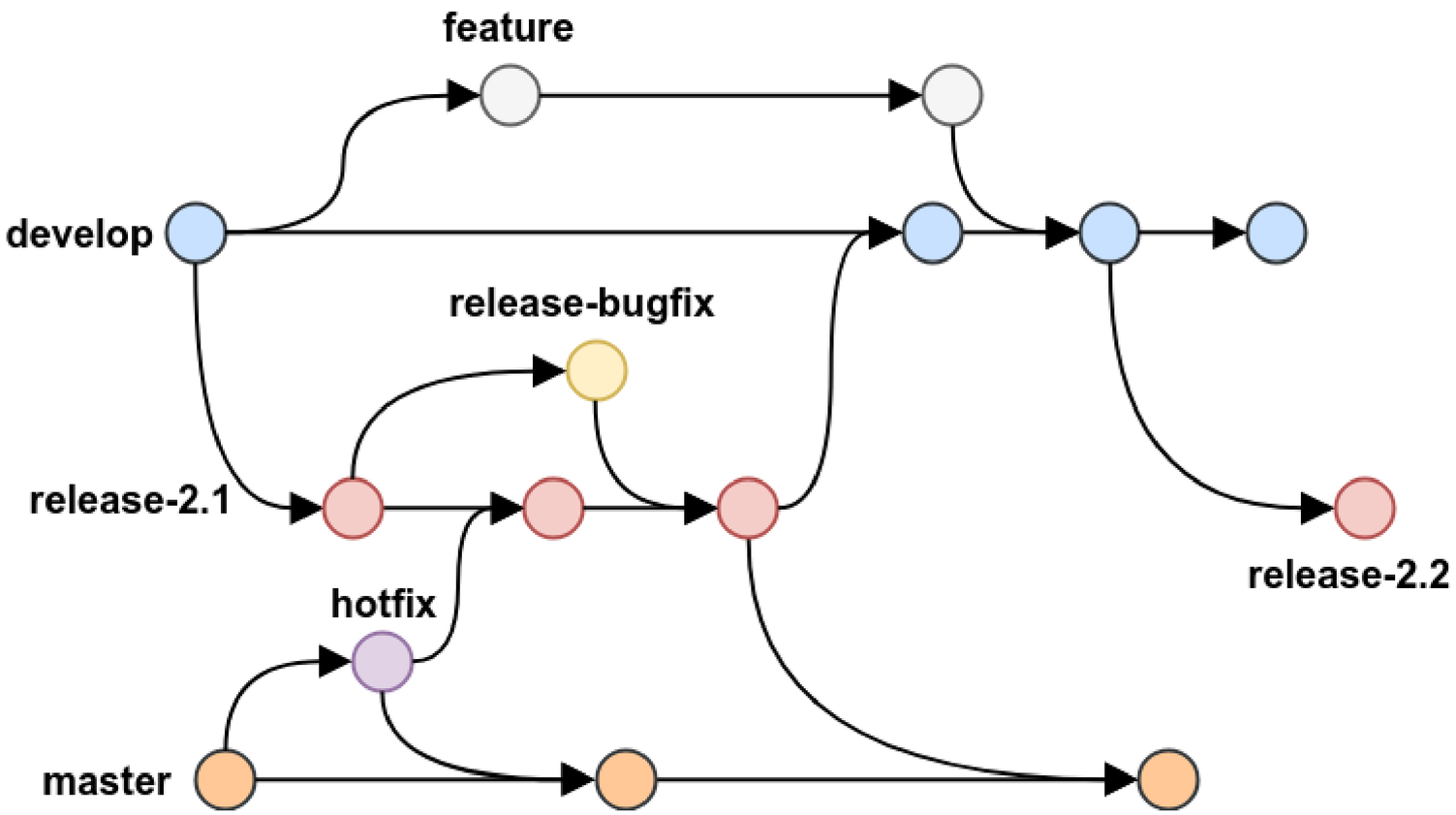
- **Working Directory** (dossier de travail)
- **Staging Area** (Zone de transit)
- **Repository** (Historique des commits)



Exemple de workflow sur Git



Exemple de workflow sur Git



Installer Git

👉 **Installation de Git**

1. Télécharger Git sur www.git-scm.com
2. Lancer l'installateur et choisir les options par défaut
3. Vérifier l'installation en ouvrant un terminal (Git Bash ou cmd) et en tapant : **git --version**
4. Si une version s'affiche (ex. git version 2.34.1), l'installation a réussi.

Si git n'a pas été ajouté, il faut modifier le path :

https://www.answerlookup.com/how-add-git-windows-path-environment-variable

ou https://grafikart.fr/tutoriels/install-git-windows-582

Initialiser Git dans son dossier de travail

👉 Initialiser Git

1. Ouvrir Gitbash
2. Se rendre dans son dossier de travail
3. Entrer la commande “**git init**” (profitez-en pour afficher les dossiers cachés de Windows)
4. Le dossier git a été initialisé pour le projet

👉 Configurer Git

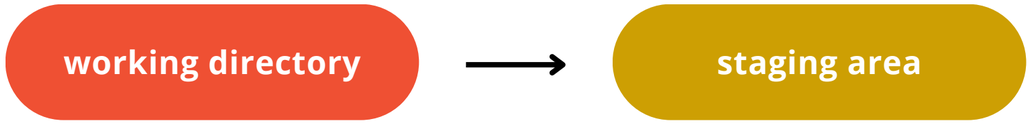
Avant de commencer, il faut configurer Git. La configuration par défaut est correcte, mais on doit renseigner notre nom et email. Ces informations apparaîtront dans l’historique et permettront d’identifier qui a fait quoi.

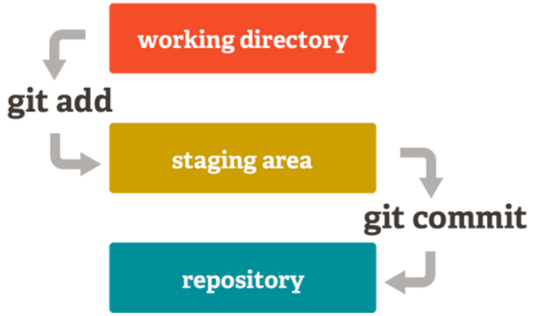
1. Entrer la commande “**git config --global user.name "$votrenom**"
2. Entrer la commande “**git config --global user.email "$votreemail**”
3. Entrer la commande “**git config --global init.defaultBranch main**”
4. La commande “**git config --list**” permet de voir tous vos paramètres de configuration

Premier commit git

👉 Réaliser son premier commit

1. Entrer la commande “**git status**” et vérifier qu’il y a bien index.html non tracké.
2. Entrer la commande “**git add index.html**” pour ajouter le fichier à “stagger”





👉 Réaliser son premier commit

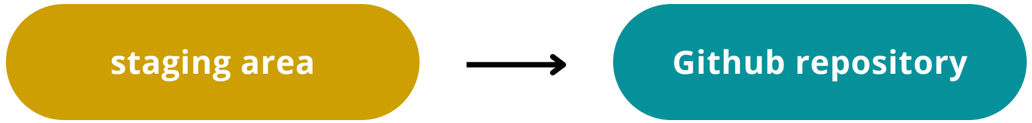
1. Entrer à nouveau la commande “**git status**”
2. Entrer la commande “**git commit -m “Ajout du fichier index.html”**” : Git enregistre définitivement cette version
3. Toujours utiliser l’impératif (Ajoute, Corrige, Modifie).

👉 Le fichier .gitignore

À ce stade, il n’est pas nécessaire, mais créer ce fichier permet d’ignorer certaines extensions / fichiers que vous ne souhaitez jamais commit dans votre projet

Passer du local au collaboratif

Nous avons installé et configuré Git en local, mais pour vraiment tirer parti du versioning, il est temps de connecter notre projet à un repository distant. Cela nous permettra de sauvegarder notre travail, de le partager avec d'autres et de collaborer efficacement



👉 Pour cela, nous utiliserons **Github**, mais il existe d'autres solutions concurrentes :

Introduction à GitHub

👉 **Github, c’est quoi ?**

GitHub est une plateforme en ligne où l’on peut stocker et partager son code avec d’autres développeurs.

👉 **Pourquoi utiliser Github ?**

1. Héberger et partager ton code.
2. Travailler en équipe sur des projets.
3. Contribuer à des projets open-source.
4. Construire un portfolio de développeur.

Récapitulatif Git & Github

GitHub est une plateforme en ligne où l'on peut stocker et partager son code avec d'autres développeurs. C'est comme un dossier partagé en ligne où tout le monde peut voir et contribuer.

- Plateforme en ligne pour héberger des projets Git.
- Accessible via un navigateur, permet de partager du code.
- Partager son code, collaborer en équipe.
- Utilise les commandes Git classiques

Git est un outil que l'on installe pour gérer son code en local. C'est comme le carnet de brouillon où tu écris tes idées

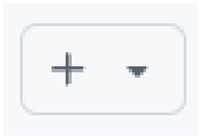
- Outil en ligne de commande pour gérer les versions en local.
- Installé sur l'ordinateur, fonctionne sans connexion Internet.
- Suivre les modifications, travailler en local.
- Utilise les commandes Git classiques

Créer son compte et son repository Github

👉 Créer son compte Github

1. Rendez-vous https://github.com/ et créez votre compte
2. Choisissez le plan gratuit
3. Validez votre adresse e-mail via le lien envoyé
4. Votre compte GitHub est prêt 🎉

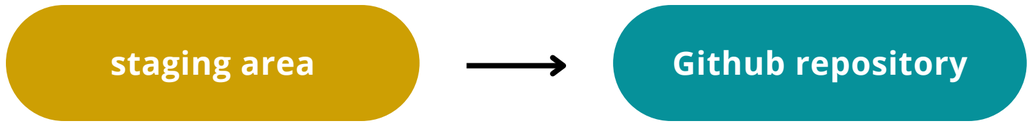
👉 Créer son repository Github

1. Connectez-vous à Github.com
2. Cliquez sur l'icône (**+**) de la page github.com 
3. Choisissez "**New repository**"
4. Dans **Repository name**, indiquez le nom de votre projet puis validez
5. Choisissez la visibilité "**Public**" pour le moment
6. Copiez l'adresse de votre dépôt (exemple **https://github.com/nextader/afec.git**)

Créer son compte et son repository Github

👉 **Ajouter le repository Github à notre projet**

Maintenant, nous allons revenir sur Gitbash pour ajouter Github comme repository distant



👉 **Ouvrir Gitbash et se rendre dans le dossier du projet et entrer les lignes de commandes :**

1. **git branch -M main** (création de la branche main)
2. **git remote add origin $votre_url_de_depot_github** (ajout du projet en remote)
3. **git push -u origin main** (push du projet)

👉 **Github va ensuite vous demander de vous authentifier**

=> Votre code sera ensuite push et accessible depuis Github.com

Comment nommer ses commits ?

👉 **Pourquoi bien nommer ses commits ?**

- ✅ Comprendre rapidement les modifications apportées.
- ✅ Faciliter le suivi de l'historique du projet.
- ✅ Éviter de perdre du temps à chercher "qui a fait quoi".

👉 **Règles générales**

- ✅ **Clarté** : Un commit doit expliquer brièvement ce qui a été fait. Pas de messages vagues comme "update".
- ✅ **Concision** : Pas plus de 50-70 caractères pour la première ligne.
- ✅ **Utiliser l'impératif** : "Ajoute" plutôt que "Ajouté".

👉 **Un bon message de commit suit généralement ce format :**

```
git commit -m "fix: correction du bug d'affichage sur mobile"
```

Comment nommer ses commits ? (récap)

👉 **Convention standard pour les commits :**

Type	Exemple	Explication
feat	feat: ajout du formulaire de contact	Ajout d’une nouvelle fonctionnalité
fix	fix: correction du bug sur le bouton	Correction d’un bug
docs	docs: mise à jour du README	Changement dans la documentation
style	style: correction de l’indentation	Modification CSS ou mise en forme du code
refactor	refactor: simplification du code de la navbar	Amélioration du code sans changer son comportement
perf	perf: optimisation du chargement des images	Amélioration des performances
test	test: ajout des tests unitaires sur la fonction X	Ajout ou modification de tests
chore	chore: mise à jour des dépendances	Tâches annexes (mise à jour, maintenance)

Les commandes de base sur Git

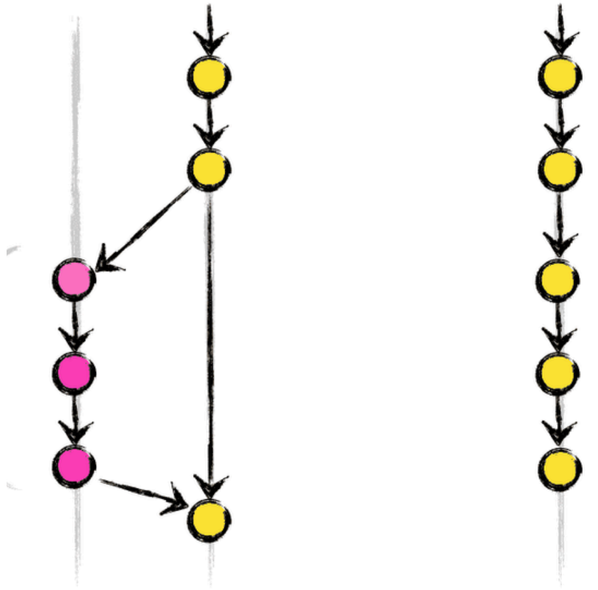
Action	Commande
Initialiser un projet Git	git init
Cloner un repo existant	git clone URL_DU_REPO
Faire un commit	git commit -m "Message"
Pousser un commit sur GitHub	git push origin main
Récupérer les changements distants	git pull origin main
Créer une branche	git branch nom_branche
Changer de branche	git checkout nom_branche / git switch nom_branche
Fusionner une branche	git merge nom_branche

Les branches sur Git

Le problème de travailler sans branche

Imaginez que vous travaillez sur une nouvelle fonctionnalité, mais en parallèle, vous devez aussi corriger un bug urgent.

- **Si tout est sur la même branche (main), il est difficile de gérer plusieurs évolutions en même temps.**
- **Une erreur peut impacter tout le projet.**



La solution : les branches !

Une branche Git permet de travailler sur une version indépendante du projet sans affecter le code principal.

Exemple :

- **Branche main → Contient le code stable.**
- **Branche feature-contact → Ajoute un formulaire de contact.**
- **Branche fix-bug-navbar → Corrige un problème sur le menu.**

Les branches sur Git

Action	Commande
git branch	Voir les branches existantes
git branch nom_branche	Créer une nouvelle branche
git switch nom_branche	Passer sur une branche
git switch -c nom_branche	Créer et basculer sur une branche
git merge nom_branche	Fusionner une branche avec main
git branch -d nom_branche	Supprimer une branche

Les conflits sur Git

Un **conflit Git** se produit lorsque deux personnes ou plus modifient la même partie d'un fichier et essaient de fusionner leurs modifications. Git ne sait pas quelle version garder et demande à l'utilisateur de choisir.

👉 **Alice modifie index.html et ajoute :**

```
<h1>Bienvenue sur mon site</h1>
```

👉 **Bob moidifie index.html mais avec :**

```
<h1>Bonjour à tous !</h1>
```

1. Alice pousse son code en premier (git push).
2. Quand Bob essaie de pousser (git push), Git lui dit qu'il doit d'abord récupérer les changements (git pull).
3. Après un git pull, Git détecte que les deux versions sont différentes → **Conflit** ! 😱

Les conflits sur Git

👉 **Comment voir qu'il y a un conflit ?**

Après un git pull ou un git merge, Git indique :

```
CONFLICT (content): Merge
conflict in index.html
```

👉 **Puis si on tape :**

```
git status
```

👉 **On voit :**

```
Unmerged paths:
 (use "git add <file>..." to mark resolution)
    both modified:  index.html
```

Les conflits sur Git

👉 **Comment résoudre un conflit Git ?**

Git marque automatiquement le fichier en conflit et montre les deux versions :

```
<<<<<<< HEAD
<h1>Bienvenue sur mon site</h1>
=======
<h1>Bonjour à tous !</h1>
>>>>>>> feature-accueil
```

- **La partie entre <<<<<<< HEAD et =======** est la version locale.
- **La partie entre ======= et >>>>>>> feature-accueil** est la version de la branche distante ou de la branche fusionnée.

👉 Il est possible d'utiliser un IDE pour que cette partie soit plus facile à corriger.

Les conflits sur Git

👉 **Les étapes pour résoudre un conflit Git**

- Ouvrir le fichier concerné (index.html).
- Choisir la bonne version (ou les combiner) :

```
<h1>Bienvenue sur mon site ! Bonjour à tous !</h1>
```

- Supprimer les marqueurs (<<<<<<<, =======, >>>>>>>).
- Enregistrer le fichier.
- Valider la résolution avec Git :

```
git add index.html
git commit –m "fix: résolution du conflit sur index.html"
```

Exercice : créer une branche, la merger et la pusher

👉 Créer une branche

- Positionnez-vous sur votre projet
- Entrez la commande “**git branch settings**”
- Tappez “**git branch**” et vérifiez que vous êtes bien sur la nouvelle branche.
- Créez deux fichiers readme.md et .gitignore
- Ajoutez-les et comittez-les sur votre branche settings.
- Pushez vos modifications sur la branche settings

👉 Merger une branche

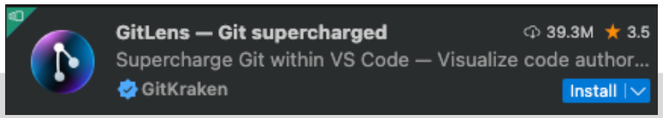
- Retournez sur la branche main : **git switch main**
- Entrez la commande **git merge settings**
- Puis pushez vos modifications : **git push**

Installer Git Graph & GitLens

👉 **Pour voir l’historique des commits sur Git, il existe une commande simple :**

git log --graph --oneline

Mais certaines extensions de VS Code peuvent vous faciliter la vie pour gérer vos commits, vos branches, voir l’historique, etc...



GitLens améliore l’intégration Git dans VS Code en offrant des outils avancés pour analyser et comprendre l’historique du code.

Git Graph est une extension qui permet d’afficher un graphique interactif de l’historique Git directement dans VS Code.

Et la commande git pull dans tout ça ?

👉 **Qu’est-ce qu’un git Pull ?**

Le **git pull** sert à récupérer les dernières modifications faites par d’autres sur le dépôt distant et à les intégrer dans ton dépôt local. En gros, c’est comme dire :

« Hé, y a du nouveau sur le projet ? Si oui, je le télécharge
et je le fusionne avec ce que j’ai chez moi. »

Techniquement, git pull fait deux choses :

1. **git fetch** → il va chercher les changements sur le dépôt distant.
2. **git merge** → il fusionne ces changements avec ta branche actuelle.

Le Fork et le Pull Request

👉 Qu’est-ce qu’un fork ?

Un **fork** consiste à copier un repository GitHub dans son propre compte.

Cela permet de travailler sur un projet sans modifier l’original. On peut :

- Modifier un projet existant sans en être propriétaire.
- Tester des améliorations avant de les proposer au projet principal.
- Contribuer à des projets open-source sur GitHub.

👉 Qu’est-ce qu’un Pull Request ?

Une Pull Request permet de proposer ses modifications au projet original.

Le Fork

1) Forker un projet

1. Aller sur la page GitHub d'un repo à forker
2. Cliquer sur "Fork" (en haut à droite).
3. Le repo est maintenant copié dans votre compte GitHub.

2) Travailler sur son Fork en local

Créer une copie locale du projet :

git clone https://github.com/votre-utilisateur/nom-du-repo.git

3) Ajouter une connexion avec le repo original (optionnel mais utile)

Permet de récupérer plus tard les mises à jour du projet original :

git remote add upstream https://github.com/proprietaire-original/nom-du-repo.git

3) Modifier le code et proposer une Pull Request

Il est recommandé de créer une nouvelle branche pour organiser son travail.

git switch -c ma-modification

Le Fork (suite)

4) Modifier les fichiers et les enregistrer

git add .

git commit -m "feat: amélioration du formulaire de contact"

5) Pousser la branche sur GitHub

git push origin

✅ Les modifications sont maintenant disponibles sur votre fork (le fait de ne pas spécifier la branche indique que vous allez push toutes les branches).

6) Créer une Pull Request (PR)

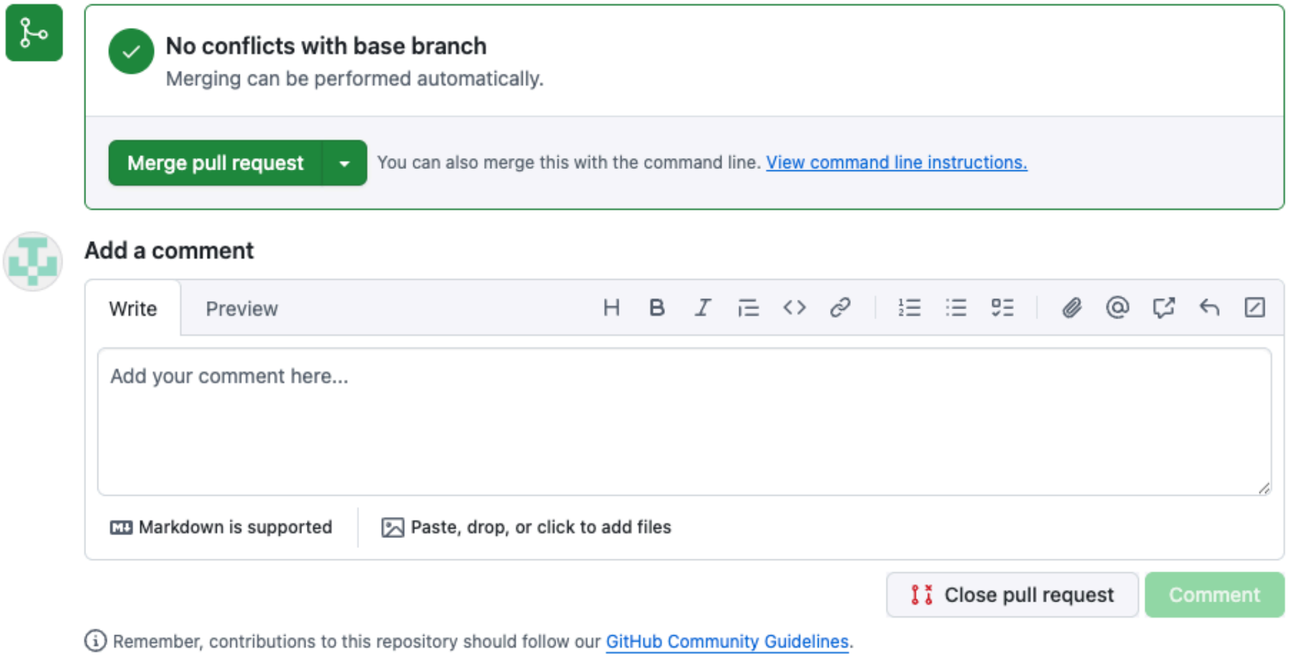
1. Aller sur votre fork sur GitHub.
2. Cliquer sur “Compare & pull request”.
3. Vérifier que la branche principale du projet original est bien sélectionnée comme destination.
4. Ajouter un titre et une description claire des modifications.
5. Cliquer sur “Create Pull Request”.

Le propriétaire du projet original peut maintenant examiner et accepter la PR.

Le Pull Request

Que se passe-t-il du côté du repository original ?

Le propriétaire du repository original peut accepter ou refuser vos modifications. Si elles sont acceptées, alors il les merge dans son projet, si elles sont refusées, alors il peut vous notifier pourquoi et vous demander d'apporter d'éventuels améliorations / changements.



Point d’honneur à git remote

git add remote

Cette commande sert à ajouter un nouveau dépôt distant à ton projet Git.
C’est ce que tu fais la première fois que tu veux connecter ton dépôt local à un dépôt en ligne (par exemple sur GitHub).

Exemple : **git remote add origin https://github.com/toncompte/ton-projet.git**

git remote set-url

Cette commande sert à changer l’URL d’un dépôt distant qui existe déjà. Tu l’utilises quand l’URL a changé, ou si tu veux passer du HTTPS au SSH (ou inversement), ou simplement si tu veux pointer vers un autre repo.

Exemple : **git remote set-url origin git@github.com:toncompte/ton-nouveau-projet.git**
Là, tu modifies juste l’URL de origin sans en créer un nouveau.

git remote -v pour vérifier les dépôts distants

Accéder à son projet via Github Pages

GitHub Pages est un service gratuit proposé par GitHub qui permet d'héberger facilement un site web statique (HTML, CSS, JavaScript) directement depuis un repository GitHub.

👉 **Idéal pour :**

- Mettre en ligne un portfolio, une documentation, ou une page de projet.
- Héberger un site statique sans avoir besoin d'un serveur.
- Déployer un site en quelques clics.

Comme notre projet est publié en mode "Public", il est tout à fait possible d'y accéder depuis Git Pages.

👉 **Activer GitHub Pages**

- Aller dans l'onglet "Settings" du repo
- Descendre jusqu'à "Pages"
- Sélectionner la branche contenant le site (main ou gh-pages)
- GitHub génère une URL comme https://utilisateur.github.io/mon-site/

Exercice Par groupe de 2 à 3 personnes

👉 **Étape 1**

1. Une seule personne crée un repository sur github (entraidez-vous :))
2. Initialiser ce projet dans un nouveau dossier nommé “exercicegit”
3. Créer un fichier index.html et écrivez du HTML libre si possible
4. Commiter et Pusher sur Github
5. Activer Git Pages, rendre visible votre travail

👉 **Étape 2**

1. Les autres membres du groupe doivent forker ce projet et le cloner en local
2. Apporter des modifications au projet : ajouter un fichier readme.me, style.css, modifier le code HTML...
3. Créer une branche, committer et pousser la modification.
4. Créer une Pull Request sur GitHub pour proposer la modification au repo original.
5. La personne qui a créé le repository peut accepter ~~ou refuser~~ la modification

Exercice 2 : gestion d’un conflit

1. Créer un fichier test.txt et faire un commit (git add + git commit)
2. Créer une nouvelle branche et modifier test.html (git switch -c $branch + git commit)
3. Revenir sur main et modifier aussi test.txt (git switch main + git commit)
4. Fusionner les deux branches et observer le conflit (git merge nouvelle-branche)
5. Résoudre le conflit en combinant les versions (avec éditeur vs code par exempl)

Ressources

Learngitbranching
(lecture) https://learngitbranching.js.org/?locale=fr_FR

Commandes Git
(lecture) https://support.atlassian.com/bitbucket-cloud/docs/git-and-mercurial-commands/
(lecture) https://www.hostinger.fr/tutoriels/commandes-git

Tutoriaux Git
(lecture) https://comprendre-git.com/fr/
(lecture) https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud
(vidéo) https://grafikart.fr/tutoriels/git-presentation-1090
(vidéo Fork & Pul request) https://grafikart.fr/tutoriels/fork-pull-request-591

Documentation officielle Git
(lecture) https://git-scm.com/book/en/v2

Aide-mémoire Git
(pdf) https://training.github.com/downloads/fr/github-git-cheat-sheet.pdf

Comprendre Fork & Pull Request
https://www.youtube.com/watch?v=D5QGiIM1j20