

Git for Teenagers

**CONVIENT AUSSI AUX
ADULTES**



Introduction



Git est un système de contrôle de version décentralisé gratuit et open source conçu pour gérer tout projet, du plus petit au plus grand, avec rapidité et efficacité. Il permet à plusieurs développeurs de travailler ensemble sur le même projet, en gérant efficacement les différentes versions de chaque fichier et en assurant que les modifications ne se heurtent pas les unes aux autres.

Git a été créé par Linus Torvalds, le même informaticien finlandais qui a initié le développement du noyau Linux. En 2005, confronté aux besoins de gestion de version pour le développement du noyau Linux, Torvalds a conçu Git pour améliorer la productivité et la collaboration entre les développeurs.



Le nom "Git" est un jeu de mots de Linus Torvalds. Selon lui, c'est un terme d'affection, de mépris, ou les deux, selon le contexte, pour lui-même. "Git" est un terme argotique britannique signifiant "personne désagréable". Torvalds a choisi ce nom parce qu'il aimait penser que "je suis une personne désagréable et que je nomme mes projets d'après moi-même". D'un autre côté, "git" peut être considéré comme l'acronyme de "global information tracker" (suiveur global d'informations), reflétant sa fonction principale en tant que système de contrôle de version.

Fonctionnement



La force de Git est de pouvoir sauvegarder des fichiers dans un dossier (le repository) et de pouvoir permettre de gérer les fichiers avec des sauvegardes régulières nommées “commits”.

Le repository local va alors gérer un historique des modifications et pouvoir envoyer les changements (“push”) sur un repository distant pour partager les modifications locales avec d’autres personnes ou simplement pour les backup.



Git embarque également un système de branches. Une “branche” devient alors une version alternative du dossier et des fichiers sur laquelle il est possible de travailler sans impacter le travail principal.

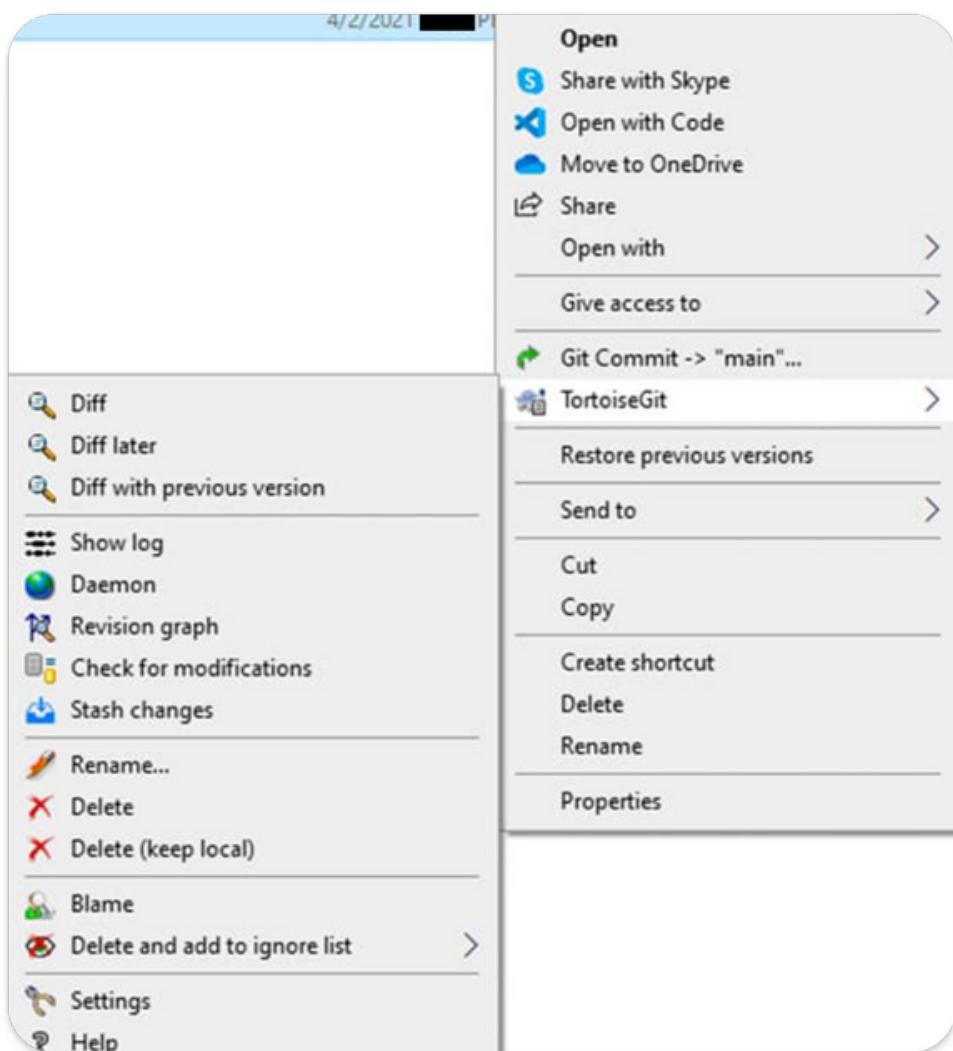
A la fin d'un travail et après satisfaction, il est possible de “merger” une branche de travail sur la branche principale pour associer tous les changements.

Les autres utilisateurs vont alors pouvoir récupérer les fichiers à jour.

Applications Tierces (TortoiseGit)



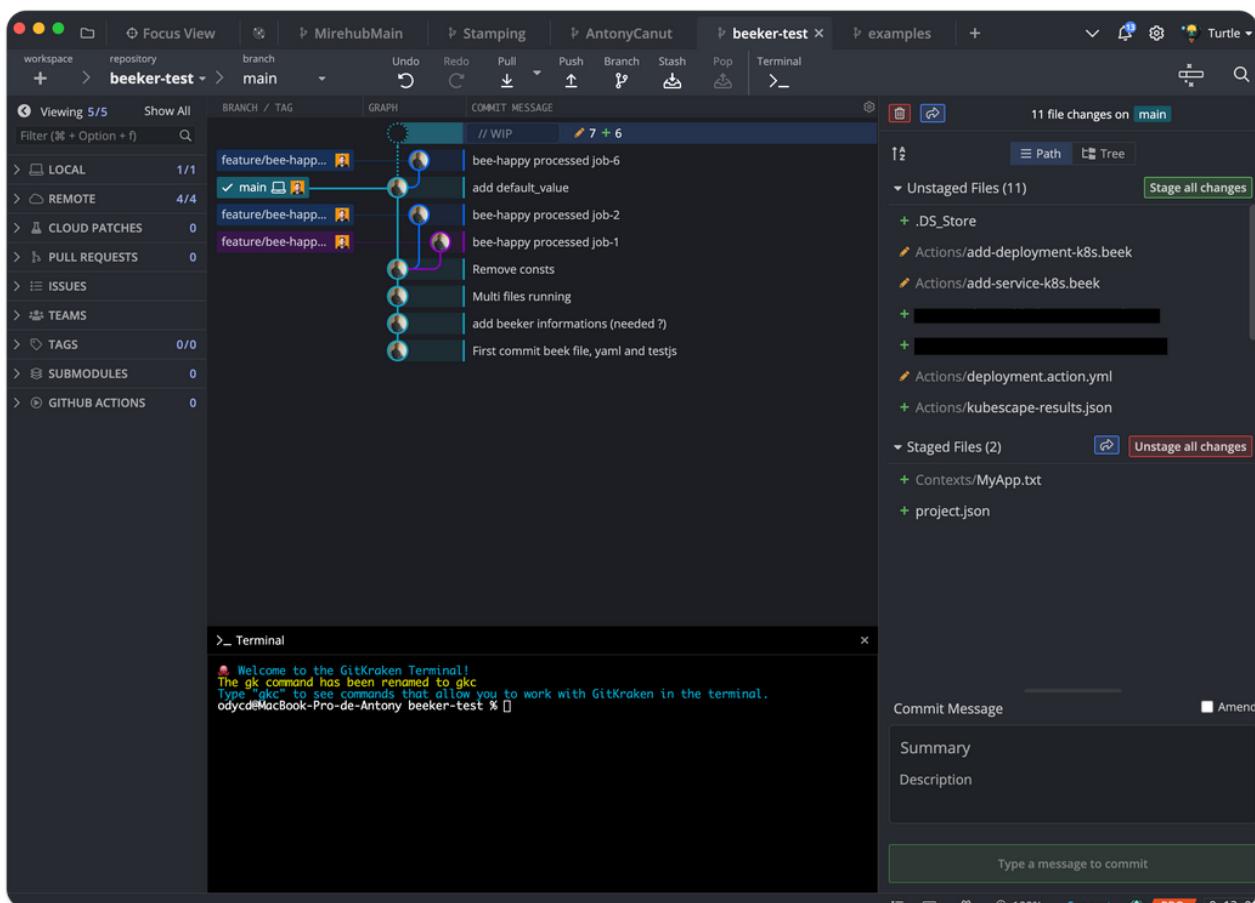
TortoiseGit est une interface Windows pour Git qui n'est pas liée à un IDE spécifique, vous permettant de l'utiliser avec les outils de développement de votre choix. Il offre une intégration facile à explorer avec des commandes Git disponibles directement depuis le menu contextuel. TortoiseGit se distingue par sa facilité d'utilisation, avec des dialogues descriptifs et une indication visuelle de l'état des fichiers directement dans l'explorateur Windows. Il propose également une intégration avec les systèmes de suivi des problèmes et divers outils utiles comme TortoiseGitMerge pour la fusion et la résolution de conflits.



Applications Tierces (GitKraken)



Pour ceux qui préfèrent une interface utilisateur graphique (GUI) à la ligne de commande, GitKraken est une application tierce populaire qui simplifie la gestion des dépôts Git. GitKraken offre une visualisation claire de l'historique des branches, simplifie les processus de fusion (merge) et de rebase, et intègre de nombreuses fonctionnalités pour améliorer la productivité et la collaboration.



Il se connecte facilement à des plateformes de dépôt en ligne telles que GitHub, GitLab, et Bitbucket, facilitant la gestion des projets hébergés sur ces services. Avec des fonctionnalités telles que les pull requests et l'aperçu des diffs, GitKraken aide les équipes à collaborer plus efficacement sur leurs projets.

GitKraken est gratuit pour les développeurs seuls sur des repos opensource.

Configuration de Git



La première chose à faire est de configurer votre nom d'utilisateur et votre adresse e-mail. Cette information sera attachée à vos commits et tags. Pour ce faire, ouvrez un terminal et entrez les commandes suivantes (différent pour l'utilisation d'une GUI) :

```
● ● ●

# Définir votre nom d'utilisateur
git config --global user.name "Votre Nom"
# Cette commande définit le nom d'utilisateur qui sera
utilisé pour tous vos commits.

# Définir votre adresse e-mail
git config --global user.email "votre.email@example.com"
# Cette commande définit l'adresse e-mail qui sera associée à
vos commits.

# Configurer l'éditeur de texte par défaut
git config --global core.editor "nom_de_l'éditeur"
# Remplacez "nom_de_l'éditeur" par le nom de commande de
votre éditeur.
# Exemples : nano, vim, code (pour VS Code).
```

Init & Clone



Pour commencer à utiliser Git sur un nouveau projet, vous devez d'abord initialiser un dépôt. Cela crée un nouveau sous-dossier nommé .git qui contient tous les fichiers nécessaires au système de suivi de version. Pour initialiser un dépôt, naviguez dans le dossier de votre projet de votre terminal et tapez la commande d'init.

Si vous souhaitez travailler sur un projet existant qui est déjà sous suivi versionnel avec Git, vous pouvez cloner ce dépôt sur votre machine locale. Le clonage d'un dépôt copie non seulement les fichiers du projet mais aussi l'intégralité de l'historique de version.

```
● ● ●

# Initialiser un nouveau dépôt Git
git init

# Cloner un dépôt existant
git clone URL_du_dépôt

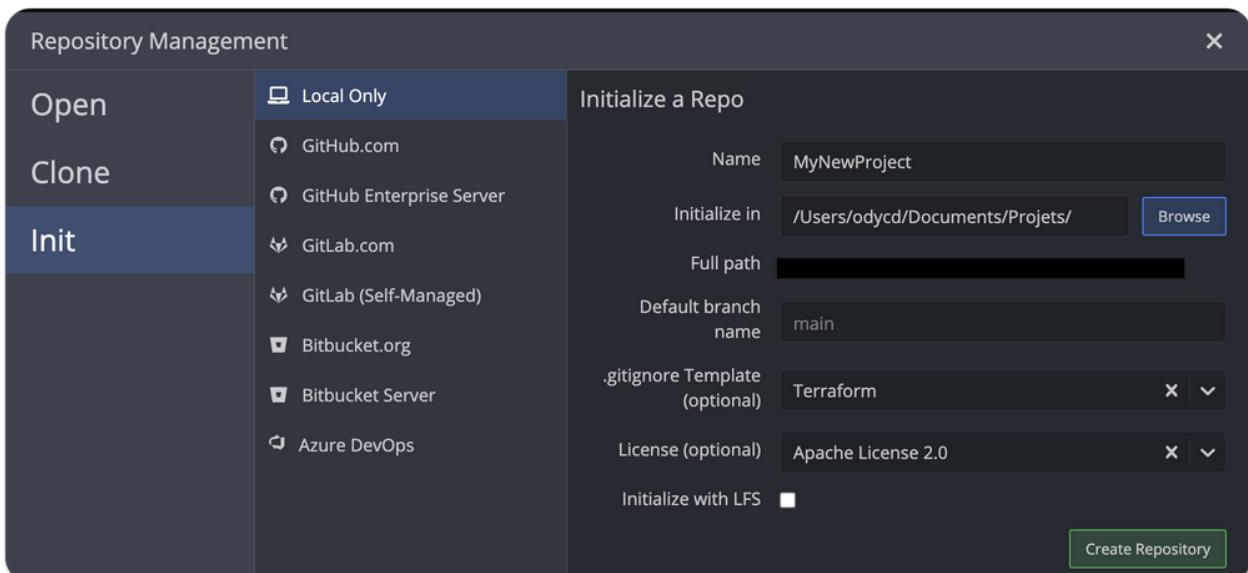
# Cloner un dépôt dans un dossier spécifique
git clone URL_du_dépôt nom_du_dossier
```

Init (GitKraken)



En utilisant GitKraken, les opérations d'initialisation et de clonage d'un dépôt Git peuvent être réalisées via une interface graphique intuitive, rendant ces processus plus accessibles aux débutants ou plus confortables pour ceux préférant une interface visuelle.

GitKraken facilite l'initialisation d'un nouveau dépôt Git sans avoir à ouvrir un terminal.



Cloner un dépôt Git existant est tout aussi simple avec GitKraken, permettant de télécharger des projets et leur historique de version sur votre machine locale.

Clone (GitKraken)



GITHUB

AntonyCanut (Public)

main · 1 Branch · 0 Tags

Go to file · Add file · <> Code

Local · GitHub CLI

HTTPS · SSH · GitHub CLI

git@github.com:AntonyCanut/AntonyCanut.git

Use a password-protected SSH key.

GITKRAKEN

Open a repo · Clone a repo

Create a repository

Open · Clone with URL · GitHub.com · GitHub Enterprise Server · GitLab.com · GitLab (Self-Managed) · Bitbucket.org · Bitbucket Server

Repository Management

Open · Clone · Init

Clone a Repo

Where to clone to: /Users/odycd/Documents/Projets/ · Browse

URL: [REDACTED]

Full path: /Users/odycd/Documents/Projets/ · AntonyCanut

Show SSH settings · Clone the repo!

Fetch & Pull



Dans la gestion quotidienne de vos projets avec Git, il est essentiel de savoir comment récupérer les dernières modifications depuis un dépôt distant et les intégrer dans votre dépôt local. Les commandes fetch et pull sont au cœur de ce processus, permettant de synchroniser votre travail avec celui d'autres contributeurs.

La commande fetch télécharge les changements depuis le dépôt distant mais ne les fusionne pas dans votre branche de travail actuelle. C'est une manière de se tenir informé des modifications sans les appliquer immédiatement.

La commande pull, quant à elle, est essentiellement une combinaison de fetch suivi d'un merge. Elle télécharge les modifications du dépôt distant et les fusionne directement dans votre branche de travail actuelle.

Les outils comme GitKraken permettent de faire des fetch de manière automatique et faciliter le processus de pull.



```
# Télécharger les changements du dépôt distant  
git fetch origin  
  
# Télécharger et fusionner les changements du dépôt distant  
git pull origin main
```

Commit



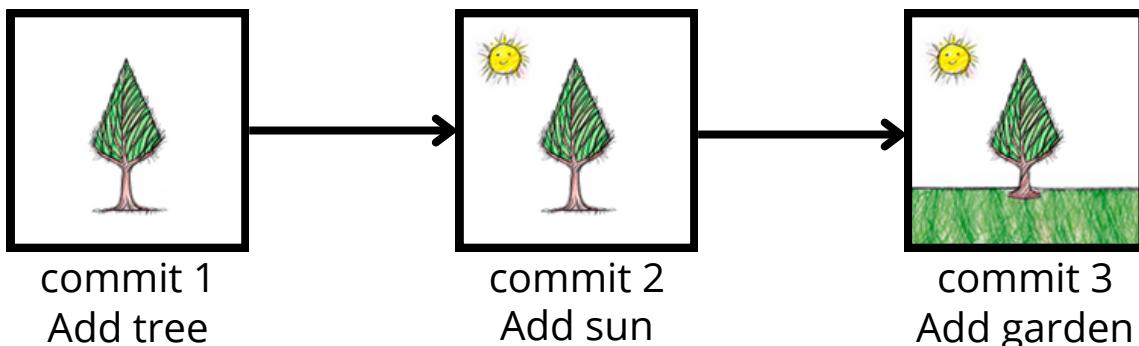
Les commits sont au cœur du système de versionnage de Git, permettant de sauvegarder des instantanés de votre projet à un moment donné. Ces instantanés incluent non seulement vos fichiers et leurs modifications, mais aussi un message de commit décrivant les changements.

Pour faire un commit avec Git, vous devez d'abord ajouter les fichiers modifiés à la zone de staging (index) avec `git add`, puis utiliser `git commit` pour créer l'instantané.

```
● ● ●

# Ajouter des fichiers à la zone de staging
git add chemin/vers/votre/fichier

# Créer un commit
git commit -m "Votre message de commit"
```

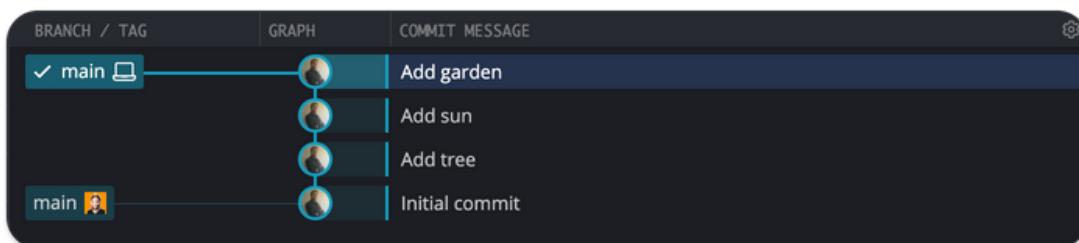
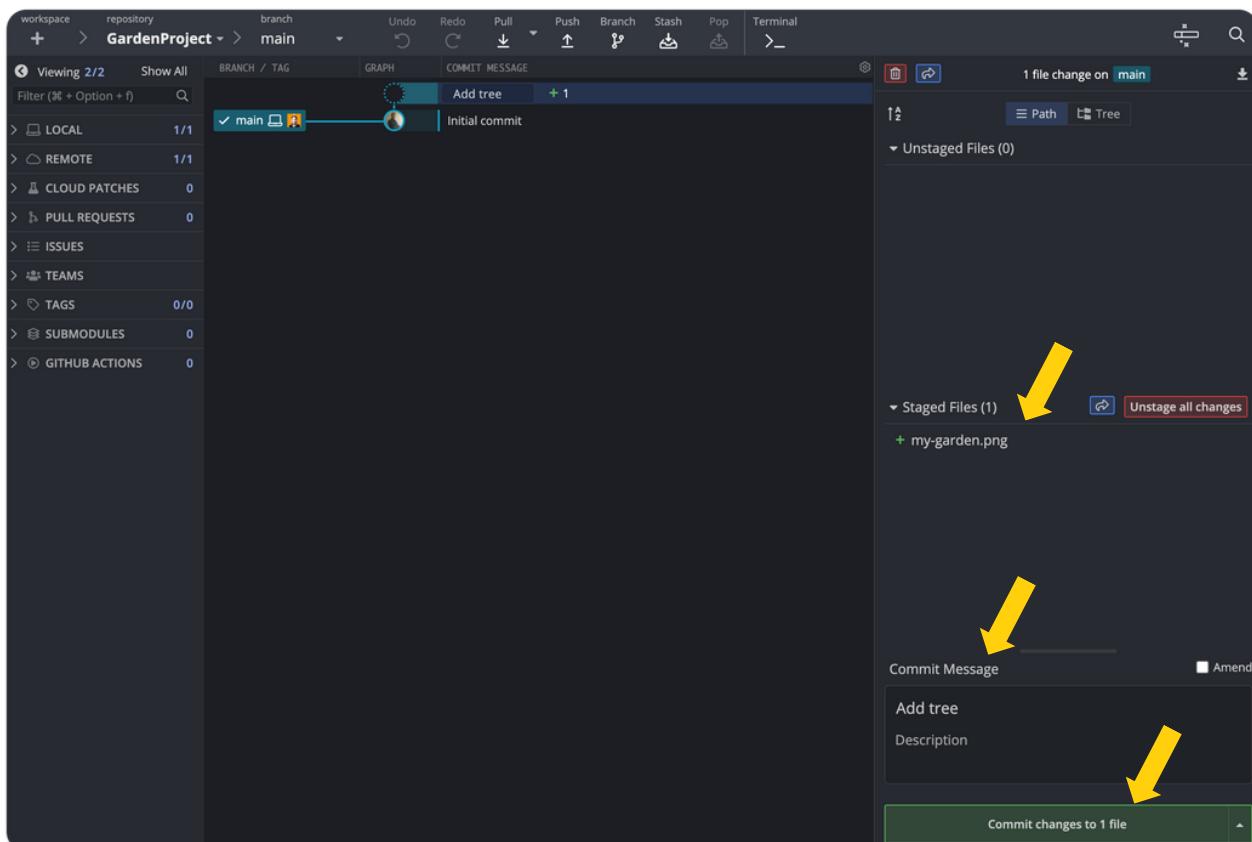


Commit (GitKraken)



GitKraken affiche clairement les fichiers modifiés, non suivis, et en attente de commit, permettant de facilement préparer votre prochain commit sans avoir à se souvenir des chemins de fichier ou des commandes spécifiques.

Vous pouvez sélectionner les fichiers à inclure dans le commit directement depuis l'interface de [GitKraken](#), en cochant les cases à côté de chaque fichier. Cela évite d'avoir à taper chaque nom de fichier individuellement dans le terminal.



Historique



Naviguer dans l'historique Git et explorer les commits antérieurs sont des compétences essentielles pour tout utilisateur de Git. Que ce soit pour auditer des changements, comprendre l'évolution d'un projet, ou simplement revenir à une version antérieure stable.

```
● ● ●

# Voir l'historique des commits
git log

# Afficher l'historique avec un format sur une ligne
git log --pretty=oneline

# Voir les modifications introduites par chaque commit
git log -p

# Comparer le contenu actuel avec un commit spécifique
git diff <commit_hash>

# Examiner le contenu d'un commit spécifique
git show <commit_hash>

# Se déplacer à l'état d'un commit spécifique
git checkout <commit_hash>

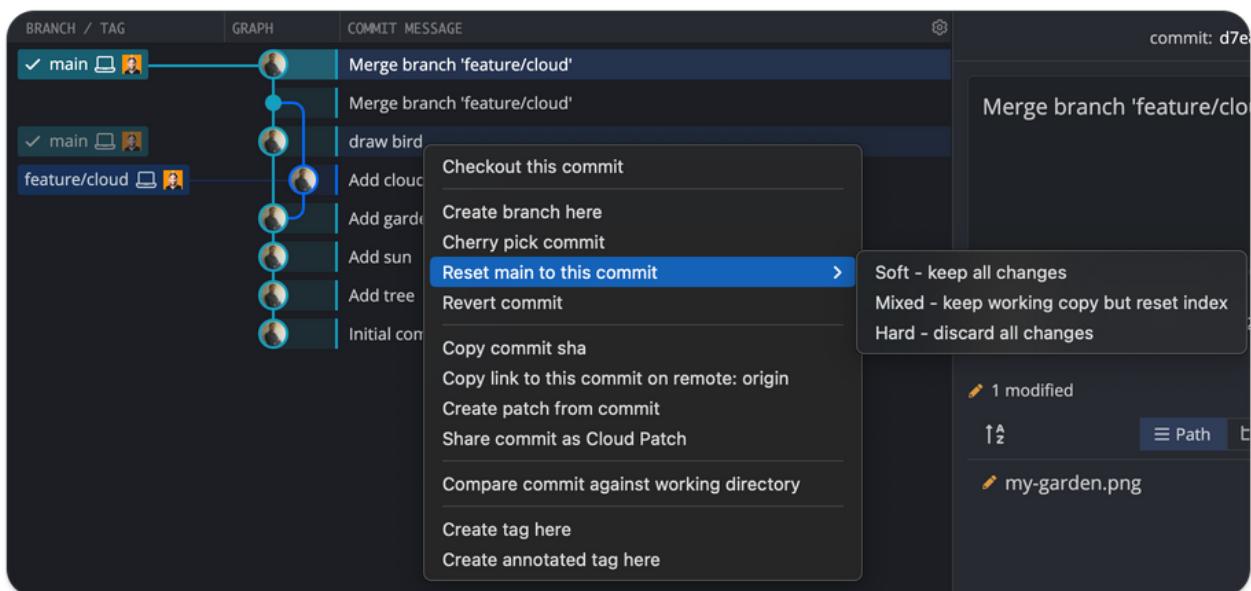
# Afficher l'historique des modifications d'un fichier
# spécifique
git log -- <chemin_du_fichier>

# Revenir à un commit antérieur en conservant les
# modifications récentes
git reset --soft <commit_hash>
```

Historique (GitKraken)



Si vous souhaitez examiner l'état du projet à un commit donné ou tester une ancienne version, GitKraken permet de faire un checkout d'un commit simplement en cliquant droit sur le commit souhaité et en sélectionnant "Checkout this commit". Cela mettra votre espace de travail dans l'état où il était au moment de ce commit.



Push



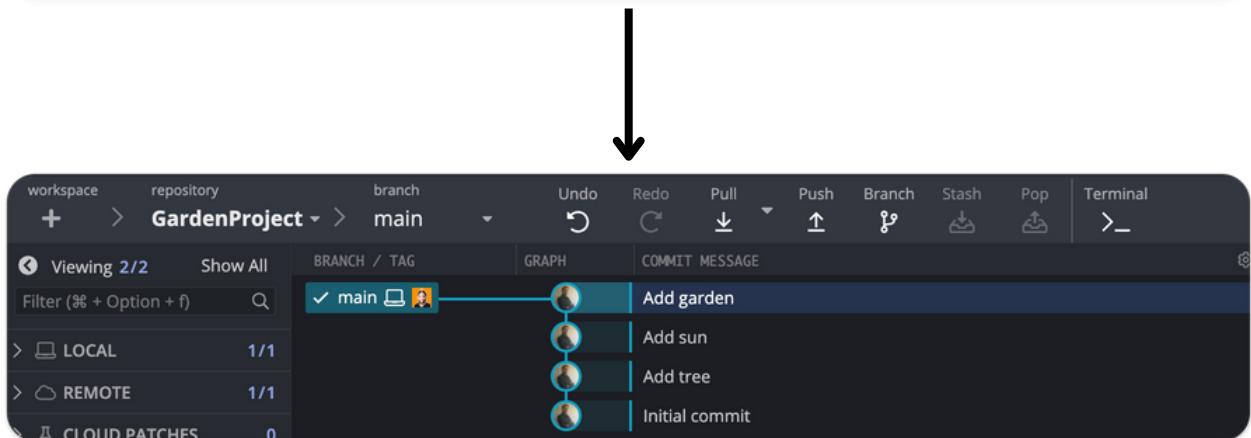
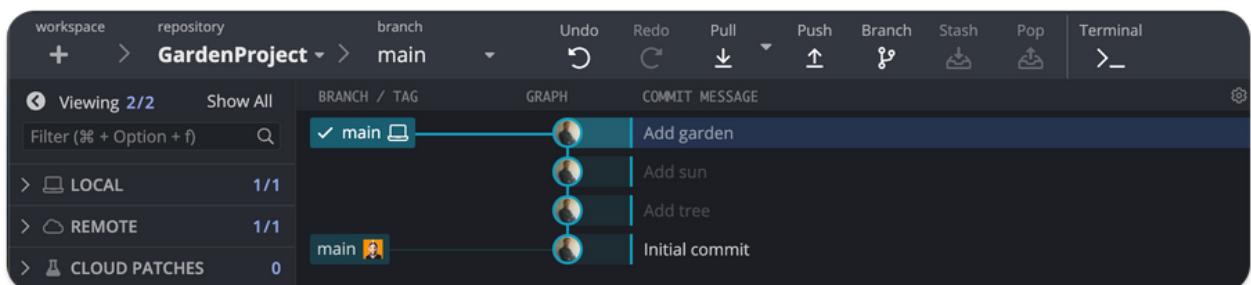
Après avoir créé des commits localement, l'étape suivante dans le flux de travail Git est généralement de "pusher" ces commits vers un dépôt distant pour partager vos modifications avec l'équipe ou pour sauvegarder votre travail sur un serveur externe.

Ce dépôt distant peut être hébergé sur Github, Gitlab, Azure DevOps et bien d'autres.

```

# Pousser les commits locaux vers le dépôt distant
git push

```



Branch



Les branches sont un concept clé dans Git, permettant aux développeurs de travailler sur différentes fonctionnalités, corrections ou expérimentations en parallèle sans interférer avec le travail des autres ou avec la ligne principale de développement.

```
● ● ●

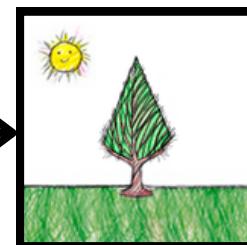
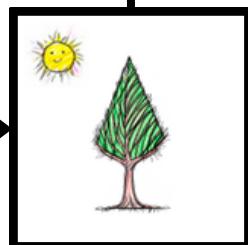
# Créer une nouvelle branche
git branch nom_de_la_nouvelle_branche

# Basculer sur une branche existante
git checkout nom_de_la_branche

# Supprimer une branche locale
git branch -d nom_de_la_branche
# Utilisez '-d' pour supprimer une branche fusionnée ou '-D'
pour forcer la suppression.
```

Branch : feature/cloud

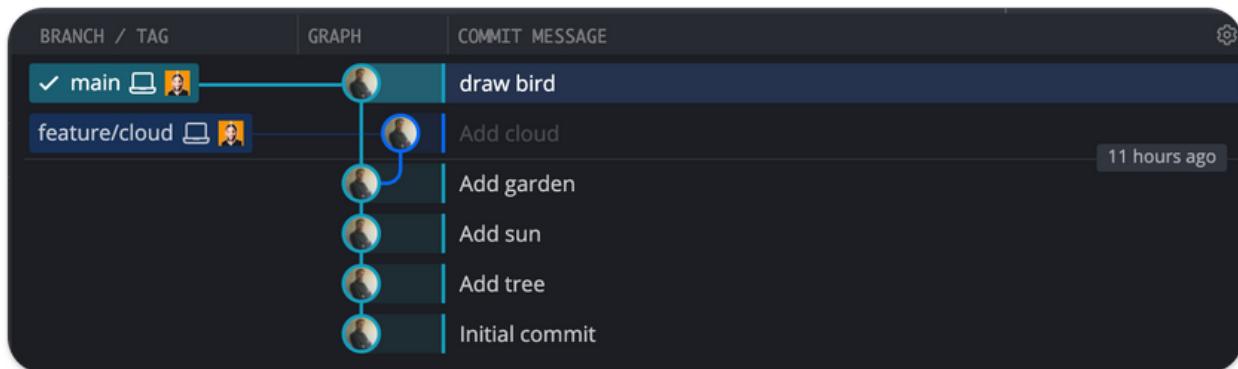
Branch : Main



Branch (GitKraken)

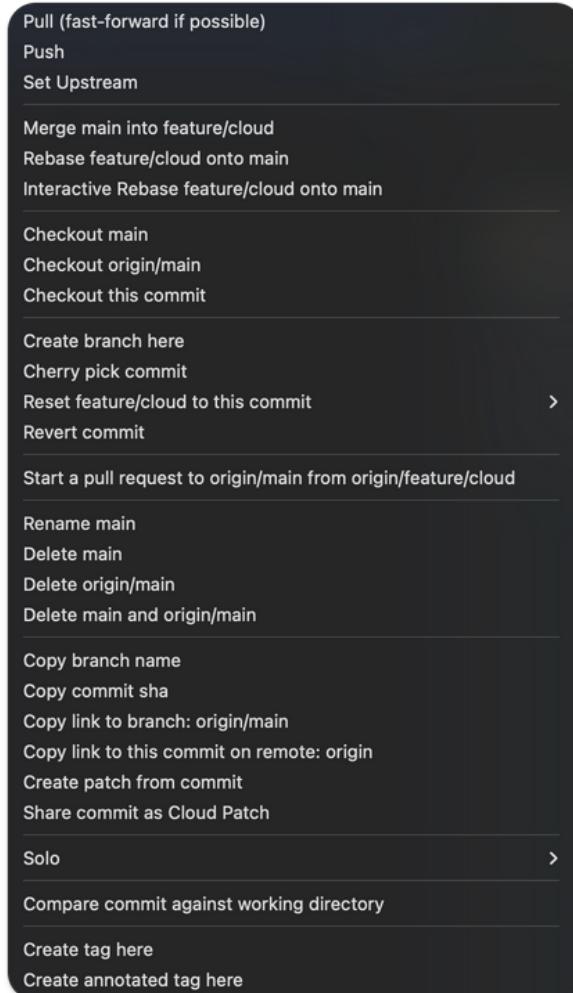


GitKraken simplifie et enrichit l'expérience de gestion des branches grâce à son interface graphique intuitive. La création des branches se fait en un click et leur gestion est très visuelle.



Créer ou supprimer des branches dans GitKraken se fait via une interface simple, souvent avec un simple clic droit et la sélection d'une option dans un menu. Cela évite d'avoir à taper des commandes spécifiques et réduit le risque d'erreur.

Basculer entre les branches est également simplifié, avec une interface qui permet de sélectionner la branche souhaitée dans une liste déroulante ou le graphique des branches.





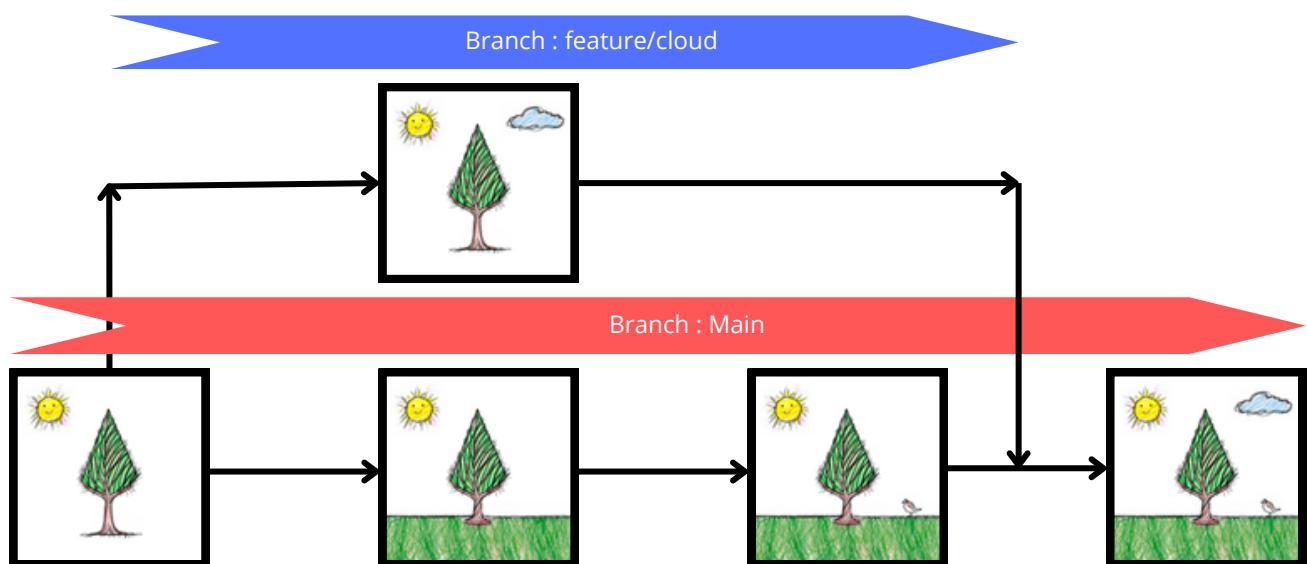
Merge

Le merge est un processus clé dans Git qui permet d'intégrer les changements d'une branche dans une autre, souvent utilisé pour combiner les travaux de fonctionnalités développées en parallèle ou pour incorporer des corrections dans la branche principale.

```
● ● ●

# Basculer sur la branche qui recevra les modifications
git checkout branche_receptrice
# Assurez-vous d'être sur la branche qui doit recevoir les
# changements.

# Fusionner la branche
git merge branche_source
# Cette commande intègre les modifications de "branche_source"
# dans la "branche_receptrice".
```

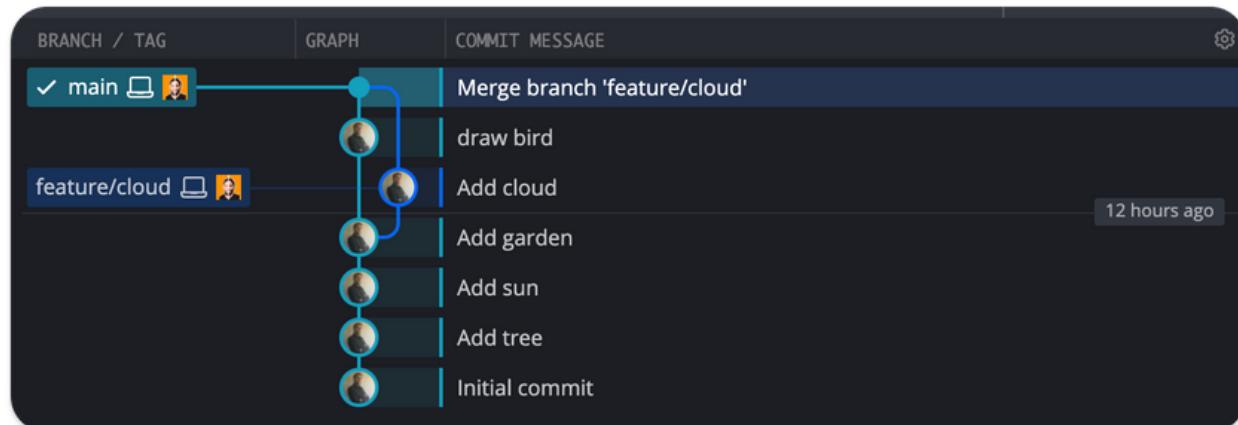
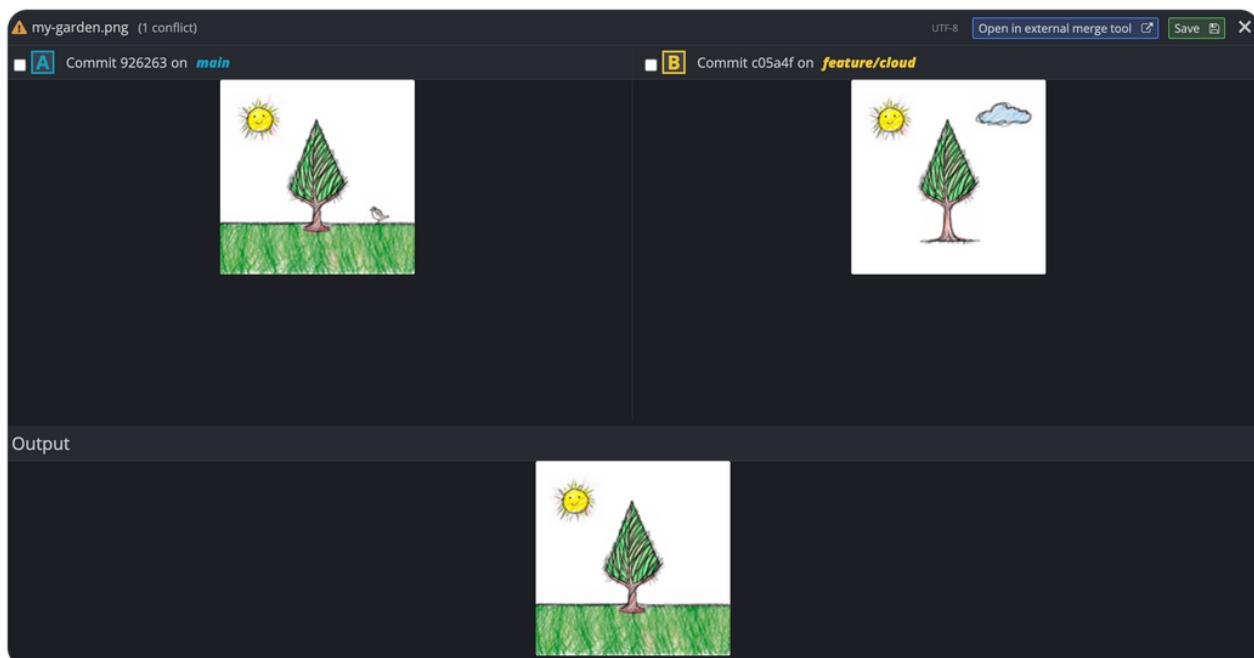


Merge (GitKraken)



Lorsqu'une fusion est initiée, GitKraken propose une interface simplifiée qui résume les changements à intégrer et permet de démarrer le processus en quelques clics.

Si des conflits surviennent, GitKraken présente un outil de résolution de conflits intégré, rendant le processus de résolution plus clair et moins intimidant que dans la ligne de commande. Cet outil permet de choisir manuellement entre les versions des fragments en conflit ou de les fusionner ligne par ligne



Rebase



Le rebase est une fonctionnalité puissante de Git qui permet de modifier l'historique des commits d'une branche. Il est souvent utilisé pour mettre à jour une branche de fonctionnalité avec les derniers changements d'une autre branche, typiquement la branche principale, ou pour nettoyer l'historique avant de fusionner une branche.

Le rebase interactif (`git rebase -i`) est un outil puissant pour modifier l'historique de commits. Il permet de fusionner, modifier, supprimer ou réorganiser les commits.

```
● ● ●

# Mettre à jour la branche de fonctionnalité avec les derniers
# changements de la branche principale
git checkout feature-branch
git rebase master

# Lancer un rebase interactif pour les X derniers commits
git rebase -i HEAD~X

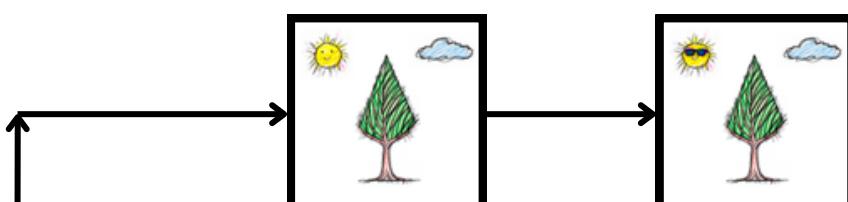
# Si le rebase ne se passe pas comme prévus, on annule
git rebase --abort
```

Rebase

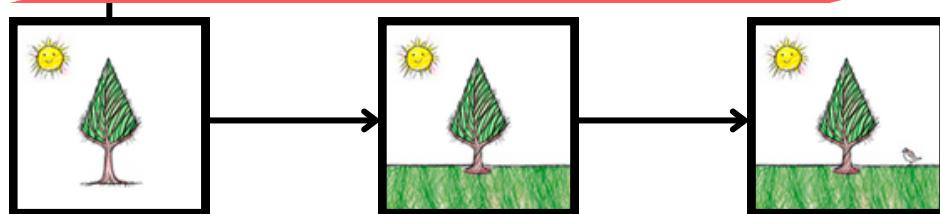


Avant Rebase

Branch : feature/cloud

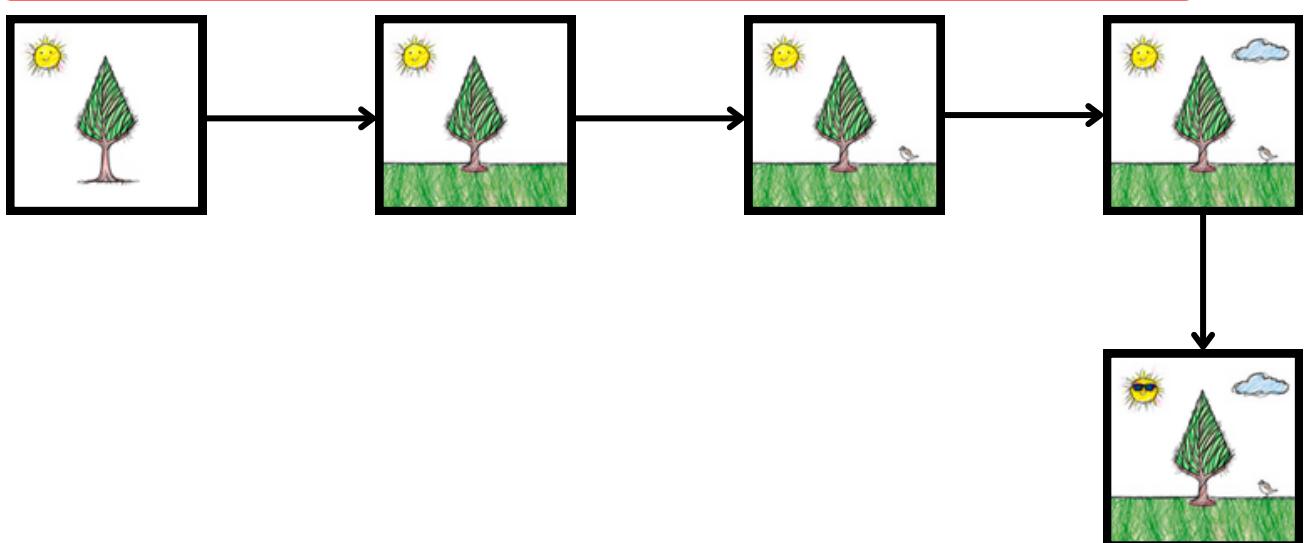


Branch : Main



Après Rebase

Branch : Main



Merge vs Rebase



Bien que Merge et Rebase atteignent le même objectif : l'intégration des modifications, elles le font de manière différente et ont leurs propres avantages et cas d'utilisation.

Le merging consiste à prendre les commits d'une branche (par exemple, une branche de fonctionnalité) et à les intégrer dans une autre (souvent la branche principale ou master). Un commit de merge est créé pour cette intégration, conservant l'historique de tous les commits de la branche de fonctionnalité ainsi que de la branche cible.

Avantages :

- Préserve l'intégralité de l'historique des commits.
- Les commits de merge fournissent un contexte clair sur l'intégration des branches.

Le rebasing réécrit l'historique en déplaçant la branche de fonctionnalité pour qu'elle commence à partir du dernier commit de la branche cible. Cela crée une séquence linéaire de commits, comme si tous les travaux avaient été effectués dans l'ordre chronologique.

Avantages :

- Crée un historique de projet linéaire et propre, facilitant la navigation et l'examen.
- Évite les commits de merge inutiles et rend l'historique plus compréhensible.

Stash



Le "stash" est une fonctionnalité de Git qui permet de mettre de côté temporairement des modifications non commitées afin de pouvoir changer de branche avec un répertoire de travail propre. Cette capacité est particulièrement utile dans les flux de travail multitâches où un contexte de développement propre est nécessaire pour basculer entre différentes tâches.

```
# Mettre en stash les modifications en cours
git stash push -m "Message descriptif du stash"
# Cette commande met de côté les modifications en cours et leur
associe un message descriptif.

# Lister tous les stashes
git stash list
# Affiche une liste de tous les stashes disponibles avec leurs
identifiants et messages associés.

# Appliquer le dernier stash
git stash pop
# Cette commande applique les modifications du dernier stash et
le supprime de la pile de stashes.
```

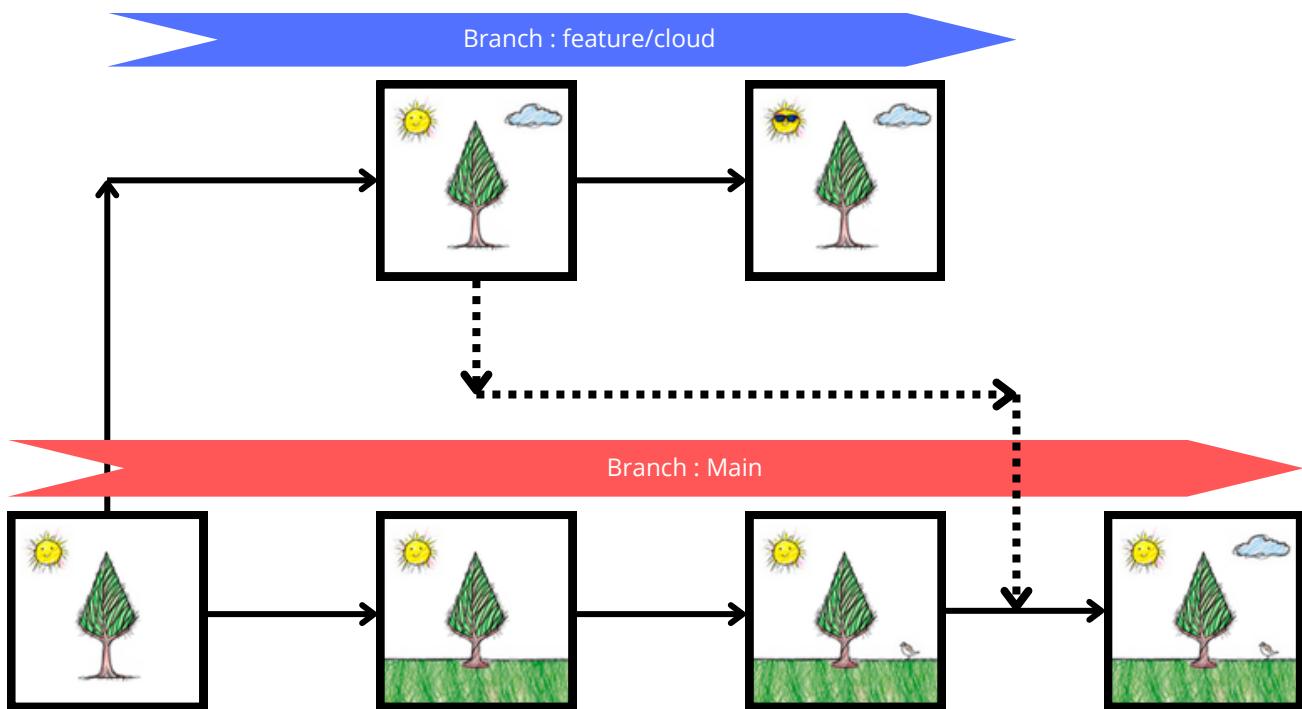
Avec [GitKraken](#), appliquer un stash à votre répertoire de travail est aussi simple que de cliquer sur le stash souhaité et de sélectionner l'action appropriée, sans avoir besoin de se souvenir des commandes spécifiques ou des identifiants de stash.

Cherry Pick



Le "cherry pick" est une fonctionnalité avancée de Git qui permet de sélectionner et d'appliquer des commits spécifiques d'une branche à une autre. Cela peut être particulièrement utile pour intégrer des corrections ou des fonctionnalités spécifiques sans fusionner l'ensemble des branches.

```
# Effectuer un cherry pick d'un commit
git cherry-pick <commit_hash>
# Remplacez "<commit_hash>" par le hash du commit que vous
souhaitez appliquer à la branche actuelle.
```



Bisect



Git bisect est un outil puissant pour diagnostiquer et identifier le commit spécifique qui a introduit un bug dans le code. En utilisant une approche de recherche binaire, Git bisect permet de réduire rapidement l'espace de recherche pour trouver le commit fautif. Malheureusement [GitKraken](#) obligera quand même l'utilisateur à utiliser les lignes de commandes, mais son interface rendra l'exercice plus simple.

```
● ● ●

# Démarrer une session bisect
git bisect start
# Ceci indique à Git de commencer le processus de bisect.

# Marquer le commit actuel ou un commit spécifique comme
# mauvais
git bisect bad <commit_hash>
# Si aucun hash n'est fourni, Git utilise le commit actuel.
# Ceci marque le point de départ pour la recherche.

# Marquer un commit connu sans le bug comme bon
git bisect good <commit_hash>
# Ceci définit le point de comparaison pour la recherche
# binaire.

# Git bisect divise alors l'historique des commits pour trouver
# le premier commit mauvais.
# Une fois identifié, Git indiquera le premier commit mauvais,
# c'est-à-dire le commit qui a introduit le bug.

# Terminer la session bisect
git bisect reset
# Ceci termine le processus de bisect et revient à la branche
# précédemment check-out.
```

.gitignore



Le fichier `.gitignore` joue un rôle crucial dans les projets Git, permettant aux développeurs de spécifier les fichiers et dossiers à ignorer dans le suivi de version. Ce mécanisme assure que les fichiers non essentiels, temporaires ou sensibles ne soient pas ajoutés au dépôt par inadvertance. Il se trouve à la racine de votre projet.

```
# Ignorer tous les fichiers .log
*.log

# Ignorer un dossier spécifique
node_modules/

# Ne pas ignorer un fichier spécifique malgré les règles
# précédentes
!important.log

# Ignorer tous les fichiers dans un dossier spécifique, sauf un
# sous-dossier
build/
!build/important/
```

gitignore.io est un service en ligne qui simplifie la création de fichiers `.gitignore` bien structurés et adaptés à de nombreux environnements de développement, langages de programmation et outils. Cet outil est particulièrement utile pour les développeurs qui travaillent sur des projets complexes ou qui utilisent plusieurs outils et langages, car il aide à générer automatiquement des règles `.gitignore` qui correspondent à leurs besoins spécifiques.

Tag



Les tags dans Git servent à marquer des points spécifiques dans l'historique d'un projet, souvent utilisés pour identifier les versions de release. Contrairement aux branches, qui évoluent avec le temps, les tags sont fixes et représentent un instantané permanent d'un état du projet à un moment donné.

```
# Lister tous les tags existants
git tag

# Créer un tag léger
git tag nom_du_tag

# Créer un tag annoté avec un message
git tag -a nom_du_tag -m "message du tag"

# Afficher les informations d'un tag annoté
git show nom_du_tag

# Pousser un tag spécifique vers un dépôt distant
git push origin nom_du_tag

# Pousser tous les tags vers le dépôt distant
git push origin --tags

# Supprimer un tag localement
git tag -d nom_du_tag

# Supprimer un tag d'un dépôt distant
git push --delete origin nom_du_tag
```

Submodules



Les submodules Git permettent d'intégrer et de gérer des projets Git dans un autre projet Git, agissant comme des références à des dépôts spécifiques à un commit particulier. Cette fonctionnalité est utile pour inclure des bibliothèques externes, des frameworks ou d'autres projets sur lesquels votre projet principal dépend.

```
# Ajouter un submodule  
git submodule add <repository-url> <path-to-submodule>  
  
# Initialiser les submodules  
git submodule init  
  
# Mettre à jour les submodules  
git submodule update
```

Pour apporter des modifications à un submodule, naviguez dans son répertoire et travaillez comme vous le feriez dans un dépôt Git normal. Pour que les changements soient répercutés dans votre projet principal, vous devrez commit les modifications dans le submodule, puis commit le nouveau commit référencé dans le projet principal.

Si vous avez des droits d'écriture sur le dépôt du submodule, vous pouvez pousser les modifications directement depuis le répertoire du submodule. Assurez-vous également de pousser le commit de mise à jour dans le projet principal pour que d'autres utilisateurs puissent obtenir la même version du submodule.

Méthodologies

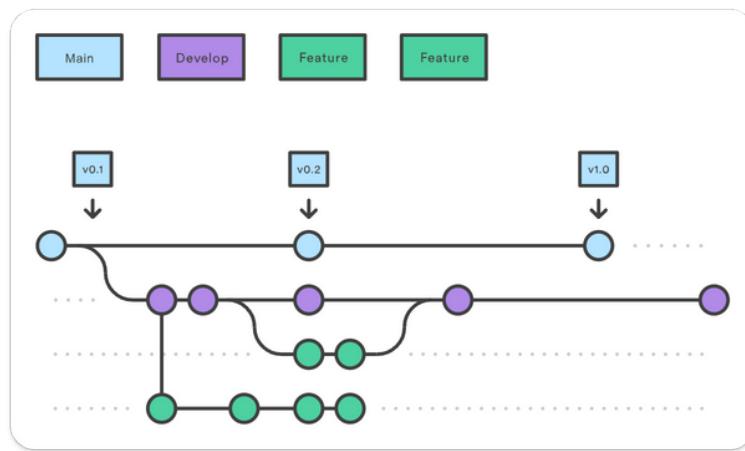
GitFlow



La gestion efficace des branches est cruciale dans tout projet de développement logiciel utilisant Git. Plusieurs méthodologies ont été développées pour structurer le flux de travail, GitFlow étant l'une des plus populaires.

GitFlow est une méthodologie de gestion de branches conçue pour le développement logiciel basé sur des releases. Elle définit une structure fixe de branches pour différentes tâches, telles que le développement, les releases, et les correctifs d'urgence.

- Branches principales:
 - main : contient le code de production.
 - develop : sert de branche de développement principal, où les fonctionnalités sont fusionnées pour les prochaines releases.
- Branches de support:
 - feature : branches utilisées pour développer de nouvelles fonctionnalités.
 - release : préparent une nouvelle version de production.
 - hotfix : permettent de réaliser rapidement des correctifs sur la branche de production.



source : [Atlassian](#)

Git... Hub ?



GitHub est une plateforme de développement collaboratif utilisée pour héberger des projets utilisant le système de contrôle de version Git. Elle facilite la collaboration entre les développeurs en fournissant des outils pour le partage de code, la gestion de projet, et l'intégration de divers services.

- **Dépôts Git** : GitHub permet aux utilisateurs de créer et de gérer des dépôts Git où le code du projet est stocké, versionné et partagé.
- **Fork et Pull Request** : Les utilisateurs peuvent "forker" des projets (créer une copie personnelle) et soumettre des "pull requests" pour proposer des modifications à des projets tiers, facilitant ainsi la collaboration ouverte.
- **Gestion de Projet** : Avec des fonctionnalités comme les issues, les milestones et les projets, GitHub aide les équipes à organiser le développement, suivre les bugs et planifier les tâches.
- **Actions GitHub** : Un outil d'automatisation pour créer des workflows personnalisés, y compris l'intégration et la livraison continues (CI/CD), les tests, et plus encore.
- **GitHub Pages** : Un service d'hébergement gratuit pour publier des sites web directement à partir des dépôts GitHub.
- **Révision de Code** : Les pull requests incluent un système de révision de code, permettant aux équipes de commenter et d'approuver les modifications avant qu'elles ne soient intégrées.
- **Sécurité** : GitHub propose des outils pour identifier et corriger les vulnérabilités de sécurité dans les dépendances des projets.

Dans la même collection

Orchestration et Gestion de Conteneurs



Infrastructure as Code



Sécurité & Gestion des secrets



Développement & CI/CD



ANTONYCANUT



ANTONY KERVAZO-CANUT

