Techniques et Outils d'Ingénierie Logicielle

...

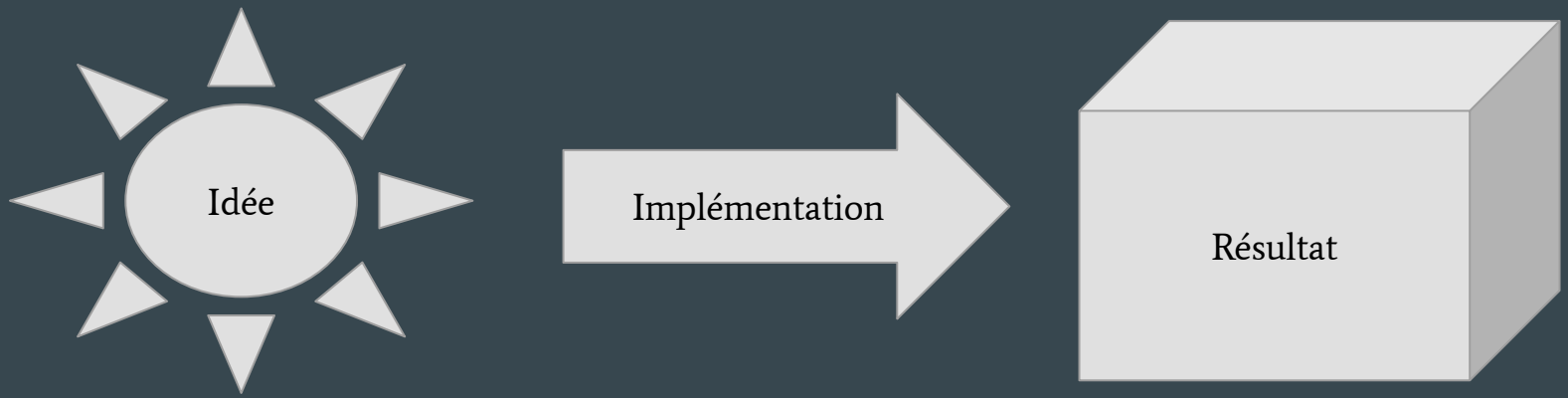
John Gliksberg

Pour récupérer les slides et exemples

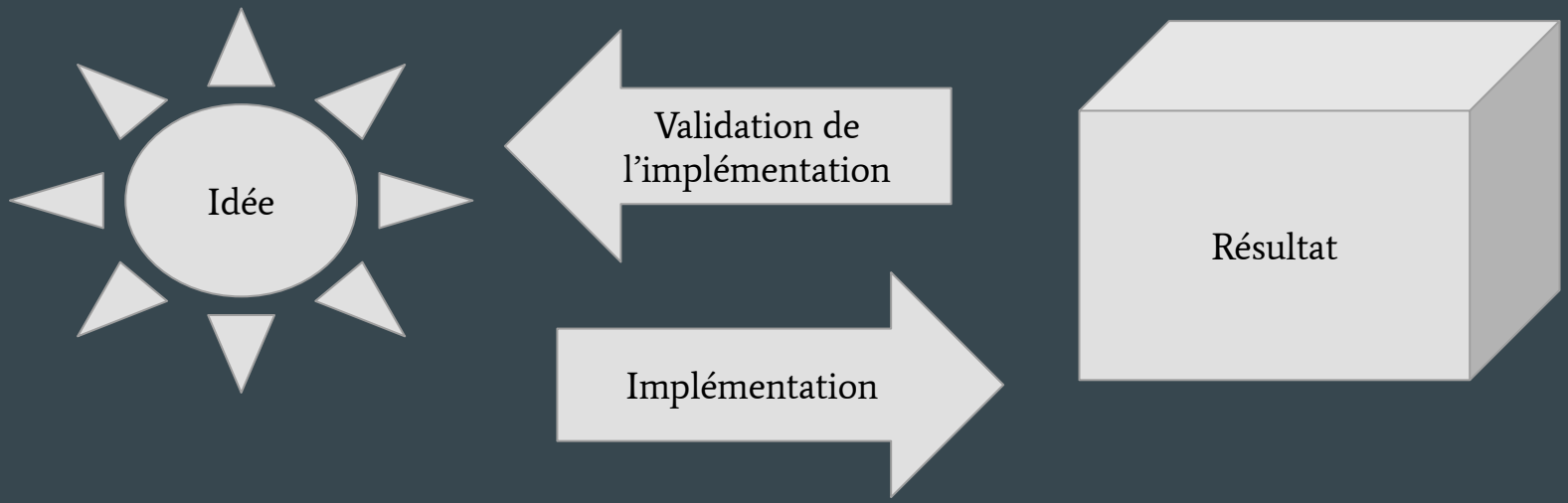
git clone https://github.com/trosh/toi2025

Cycle de vie du code

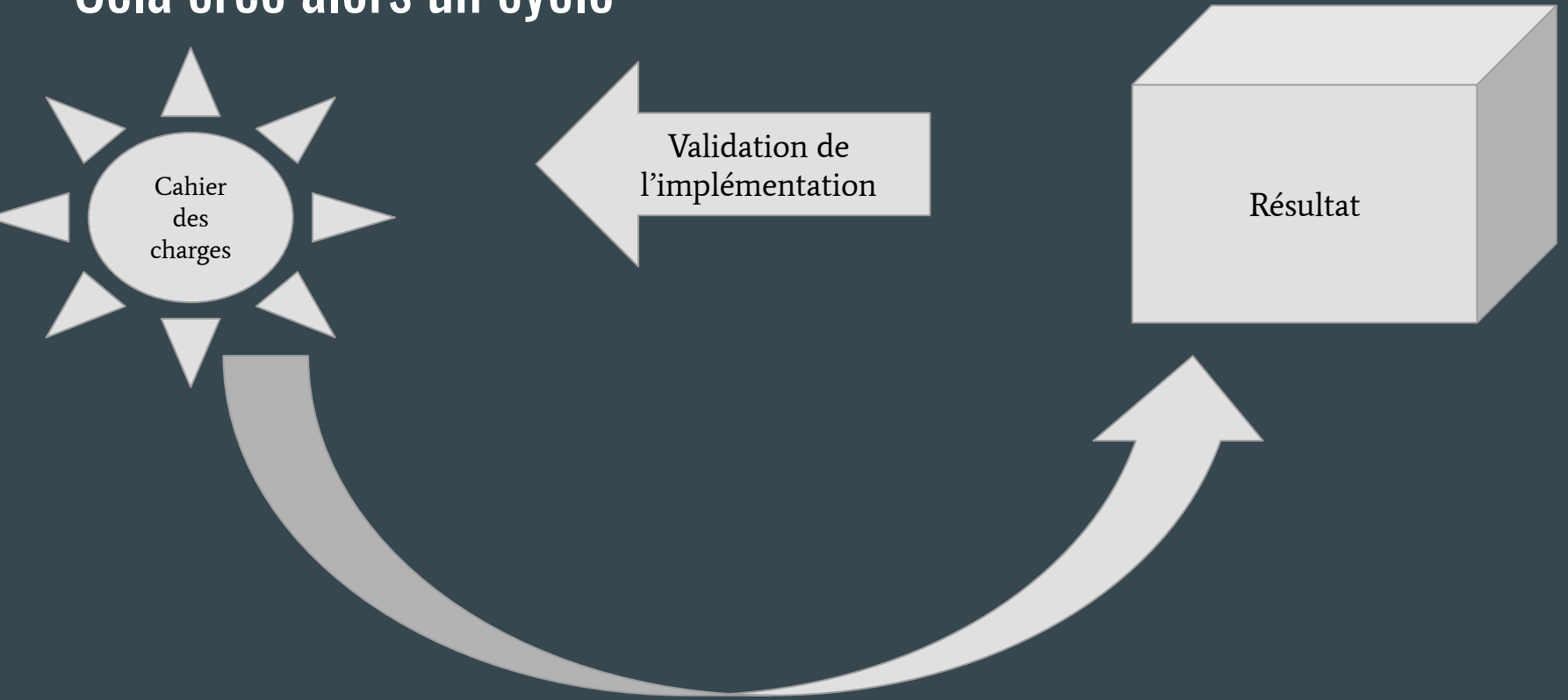
Le code répond à un besoin



On peut comparer le résultat au besoin

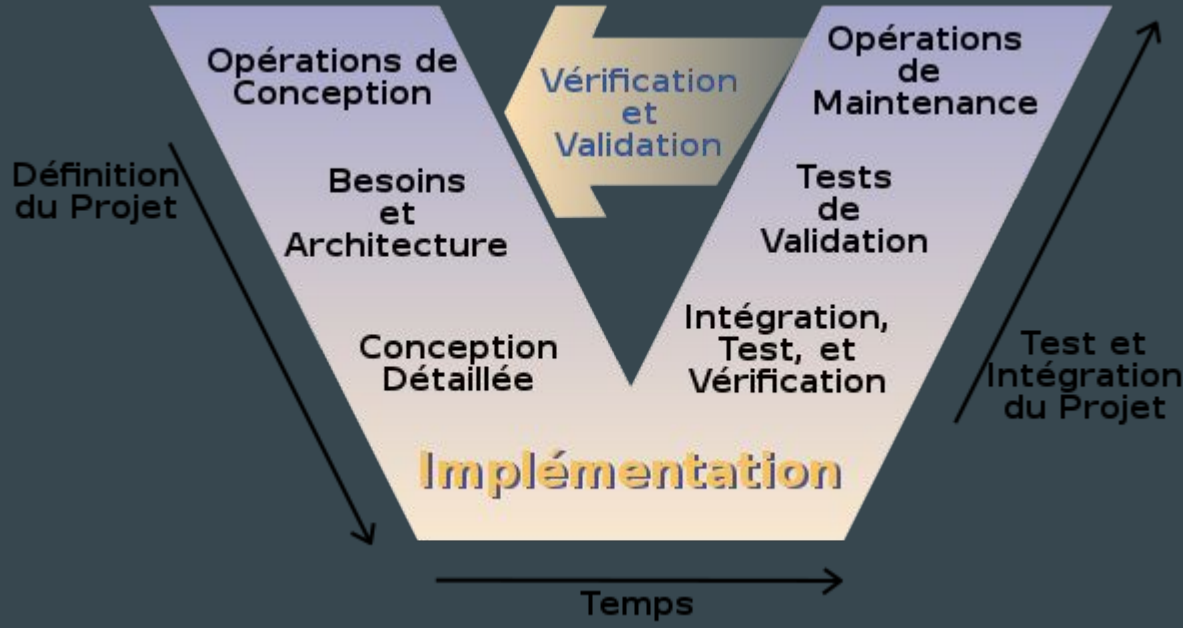


Cela crée alors un cycle



Cela crée alors un cycle

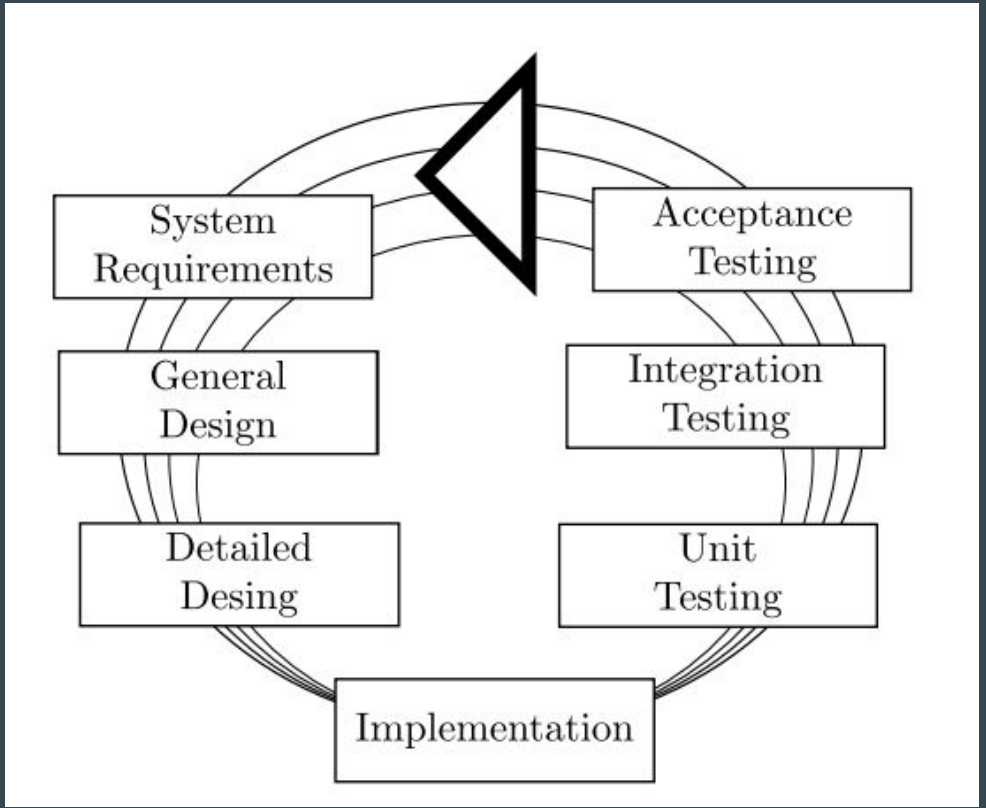
De la spécification à la validation. Le projet et avant tout affaire de communication.



Variation en mode “agile”

Répéter le cycle de manière itérative permet de:

- Raffiner le cahier des charges
- Adapter les tests
- Avancer progressivement
- **Communiquer**



La Forge

Introduction à la forge de codage

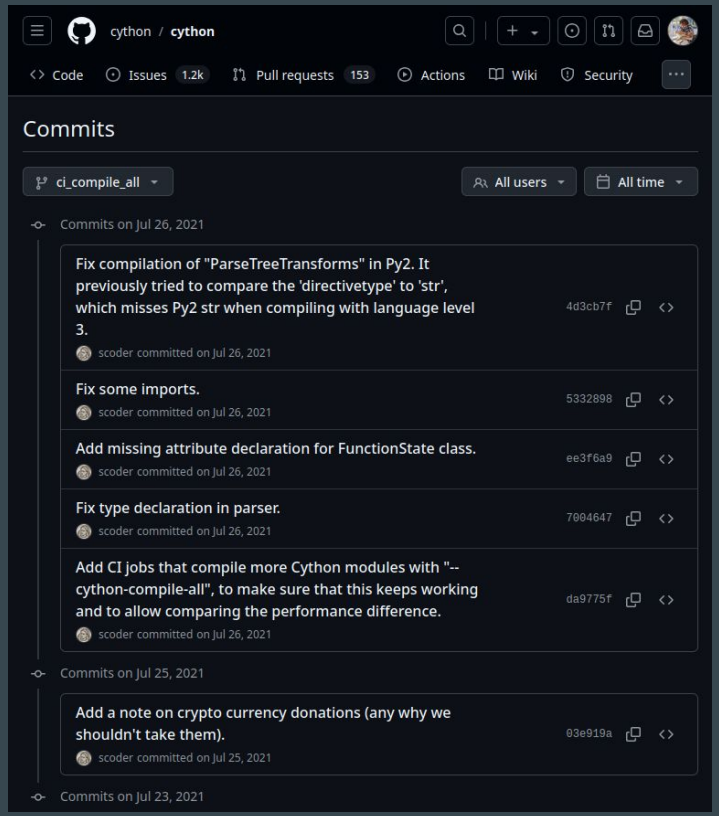
La forge de codage est une plateforme en ligne qui offre un ensemble d'outils et de fonctionnalités destinés à faciliter le processus de développement logiciel.

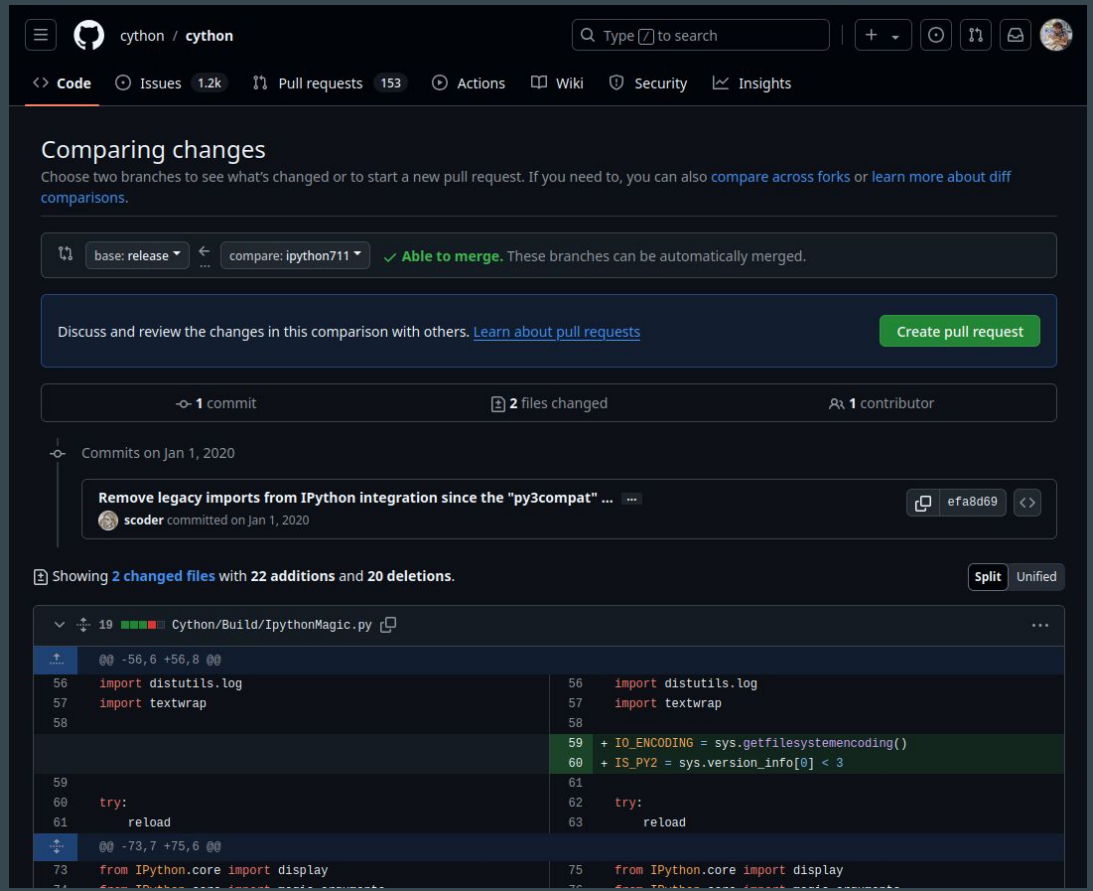
Elle agit comme un environnement collaboratif où les développeurs peuvent travailler ensemble sur des projets, partager des codes sources, des bibliothèques, des modules, et collaborer à la résolution des problèmes.

La forge de codage fournit également des outils de gestion de projet, de suivi des problèmes (bug tracking), de gestion de versions (version control), et de documentation, ce qui permet une gestion efficace et transparente des projets de développement.

Grâce à la forge de codage, les développeurs peuvent bénéficier d'une infrastructure solide pour organiser, suivre et gérer l'évolution de leurs projets logiciels, tout en favorisant la collaboration et la contribution de la communauté.

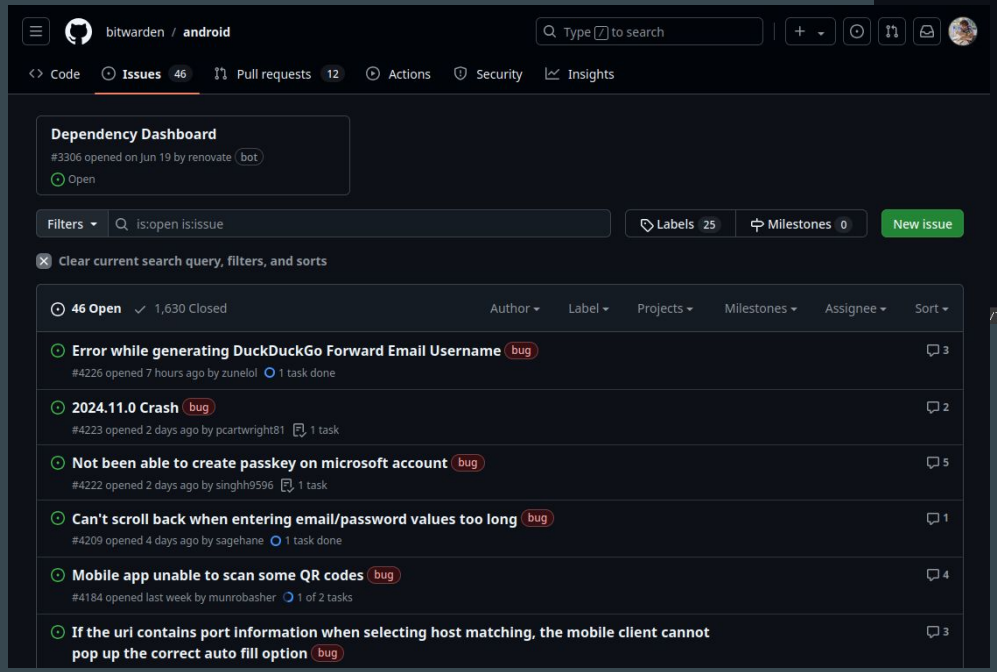
Ce que permet une forge

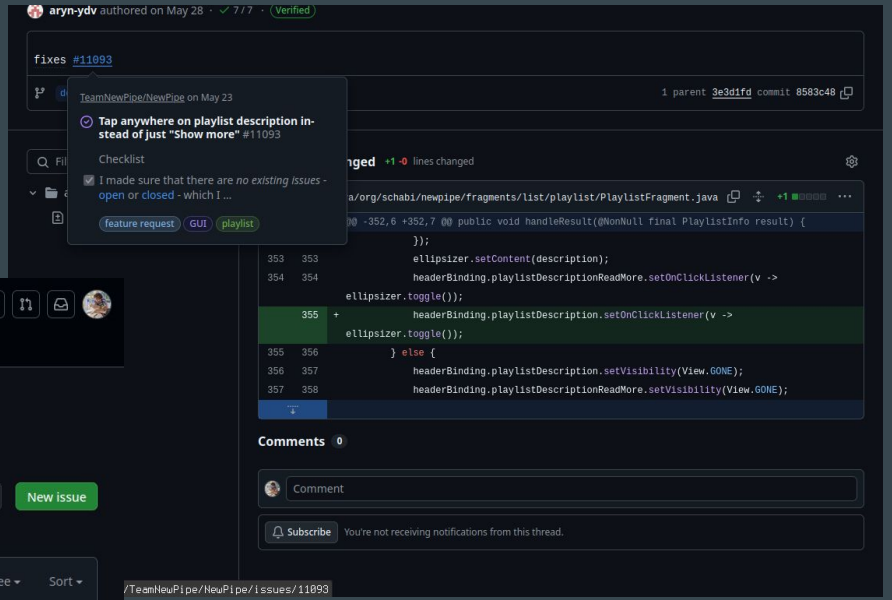




contrôle de version

Ce que permet une forge

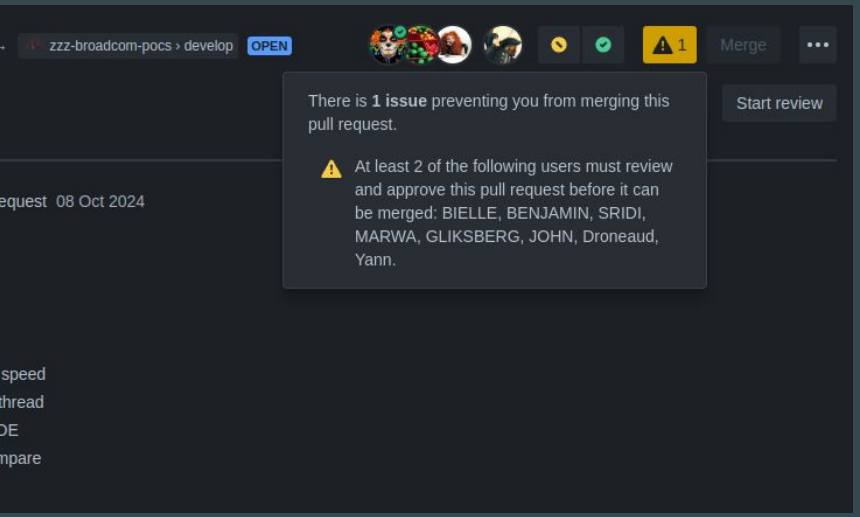


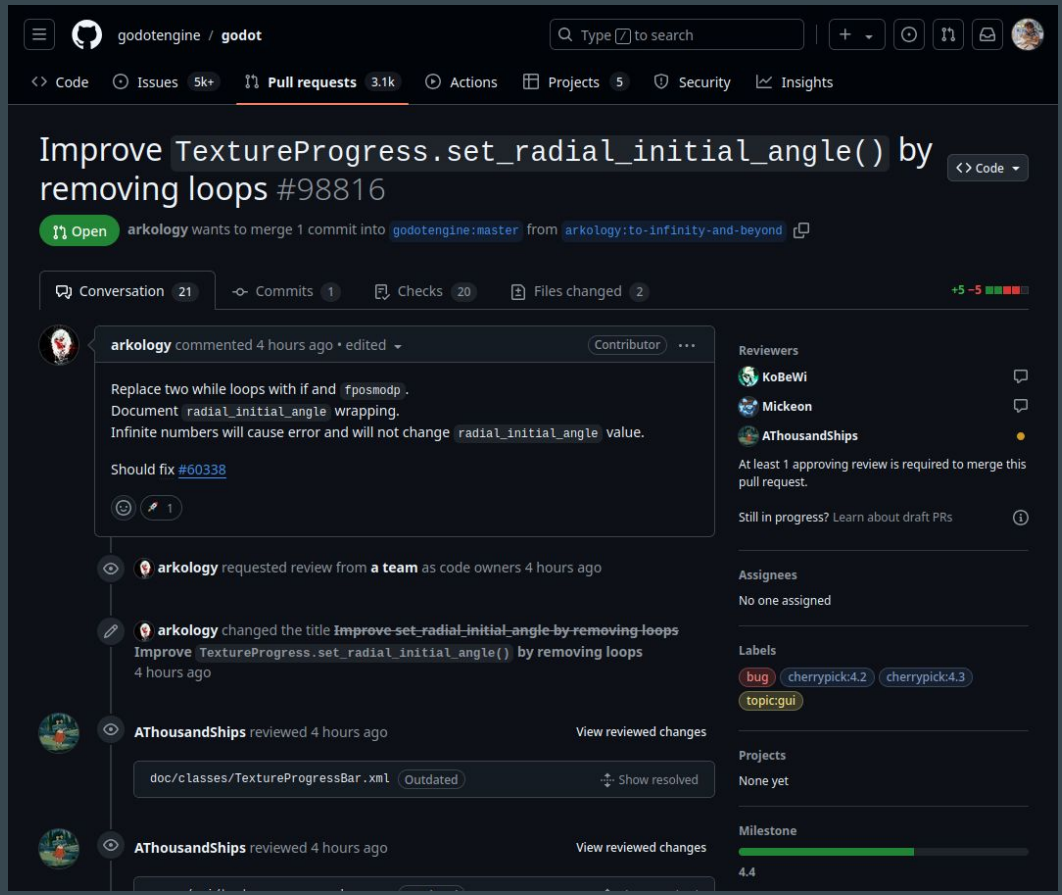


suivi de bugs/tickets

Ce que permet une forge

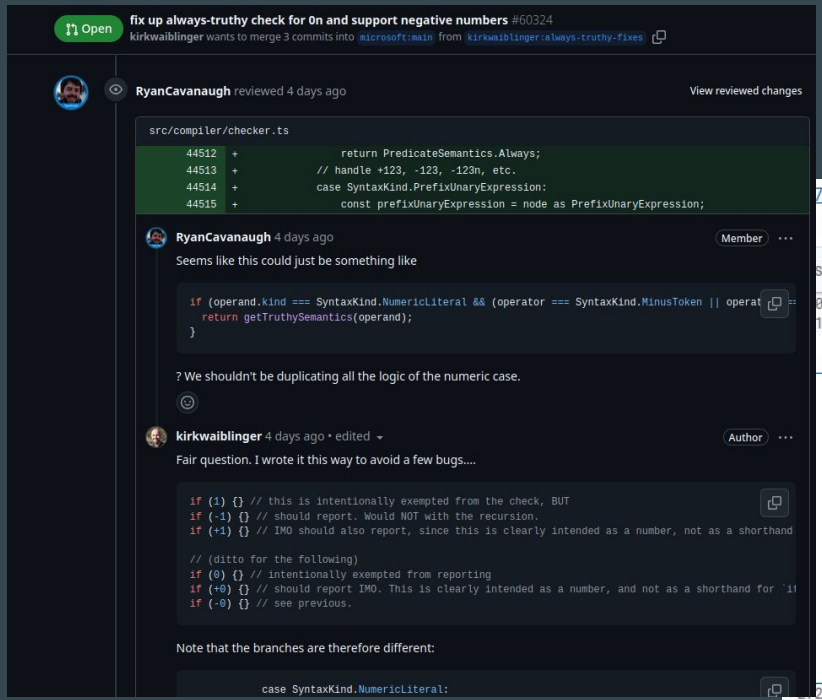
pull/merge requests

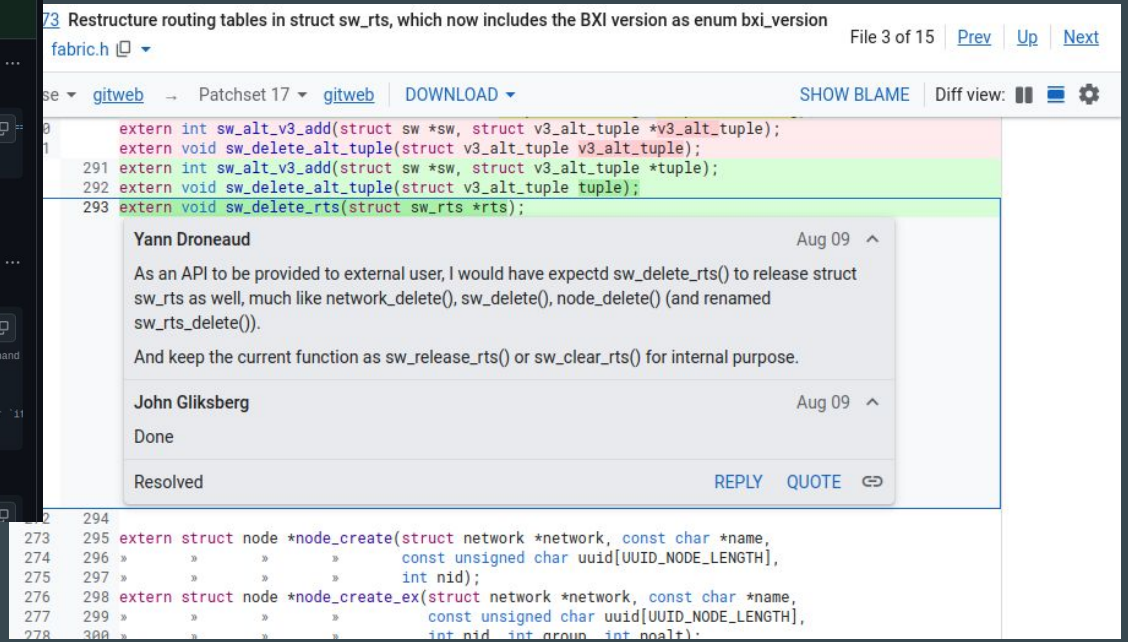




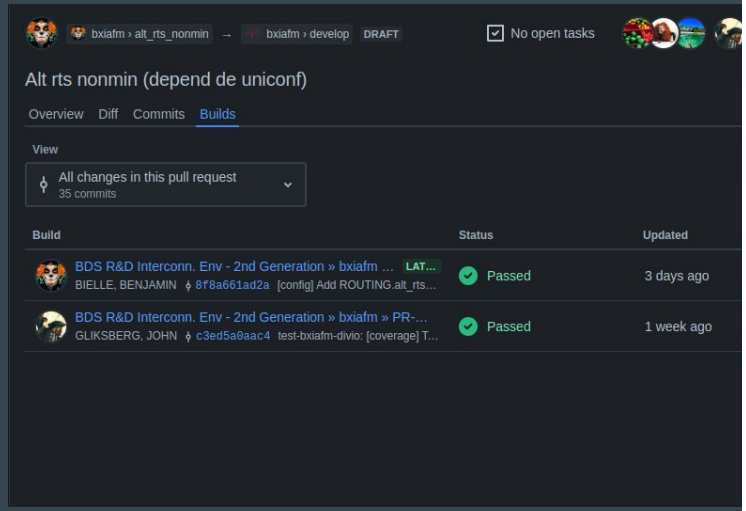
Ce que permet une forge

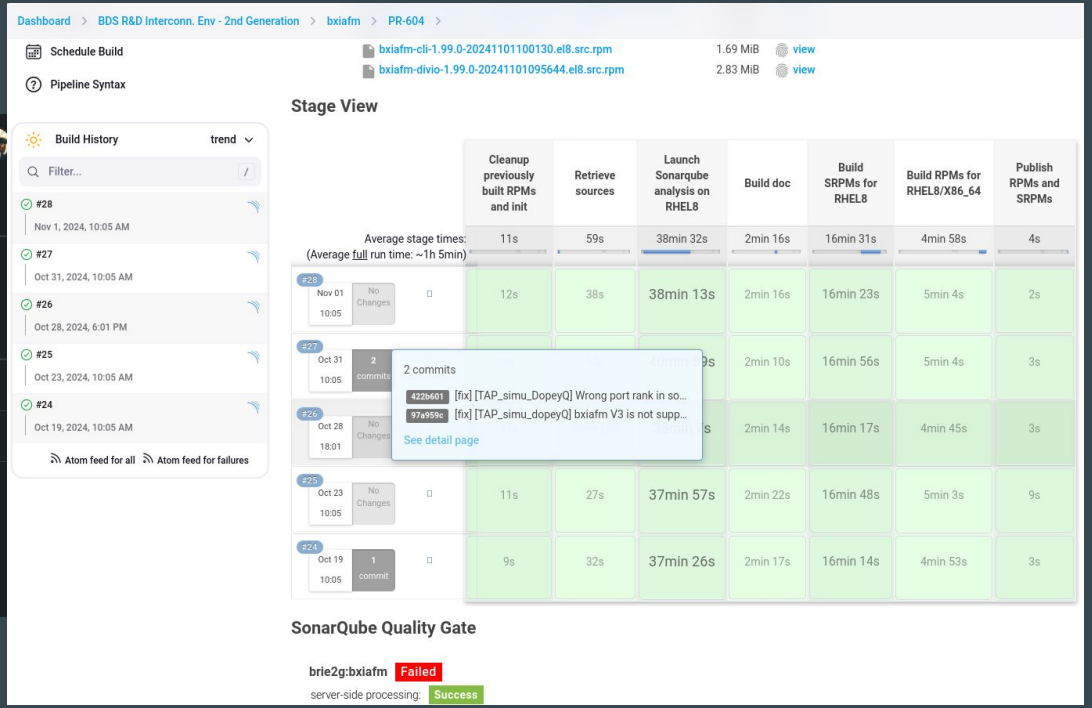
collaboration/discussions





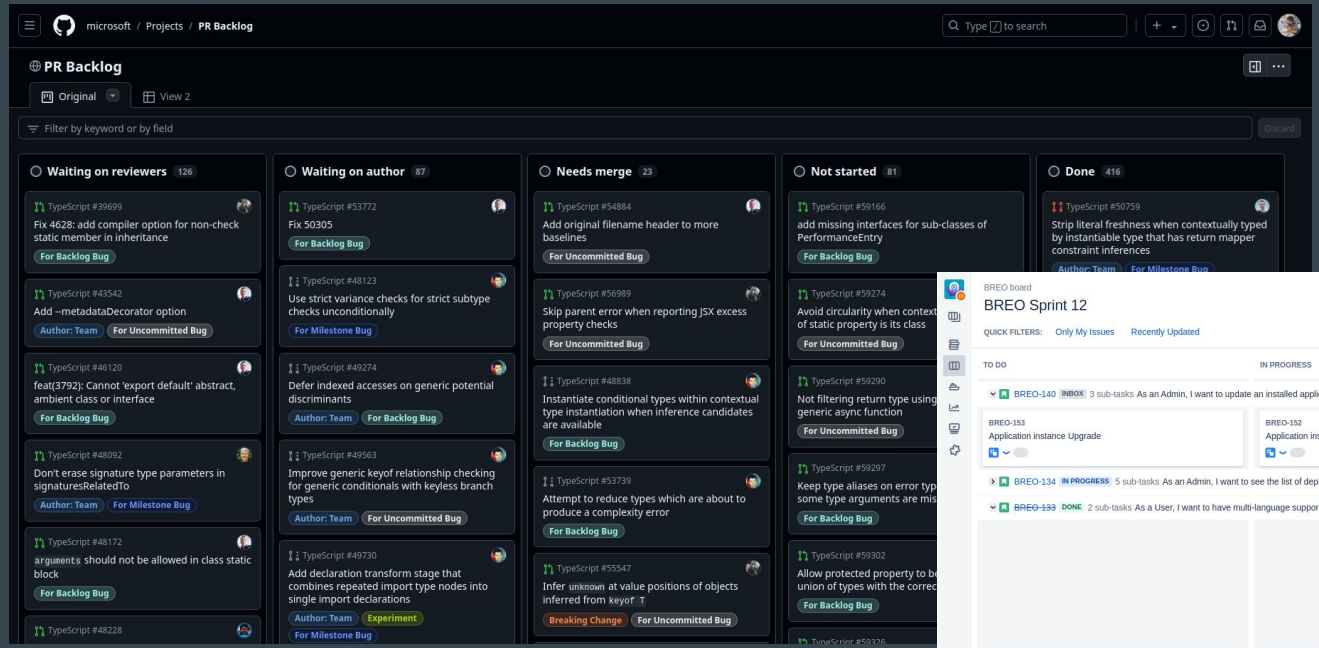
Ce que permet une forge

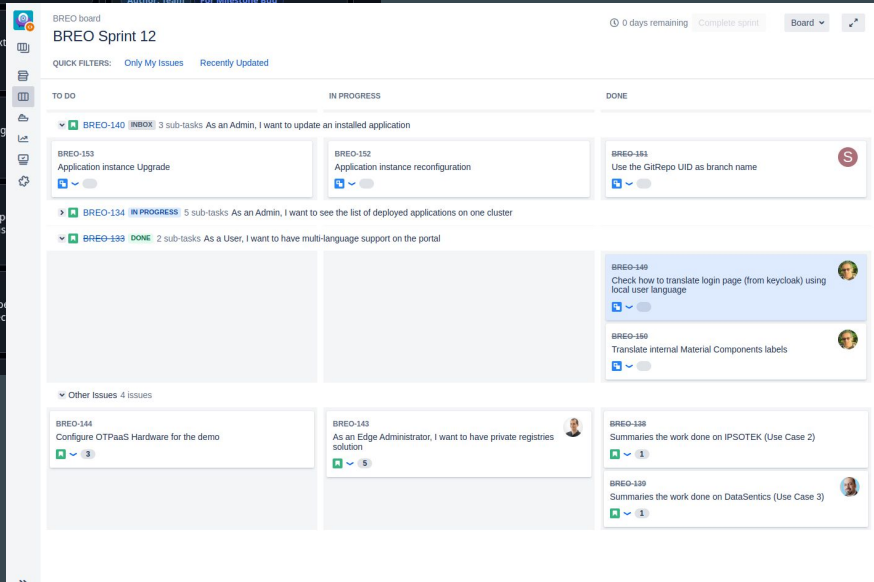




intégration continue

Ce que permet une forge





gestion de projet

Suivi des problèmes (bug tracking)

Le suivi des problèmes (bug tracking) est une fonctionnalité essentielle d'une forge de codage.

Permet aux développeurs de signaler les anomalies, les bogues et les problèmes rencontrés dans le code source ou dans le logiciel.

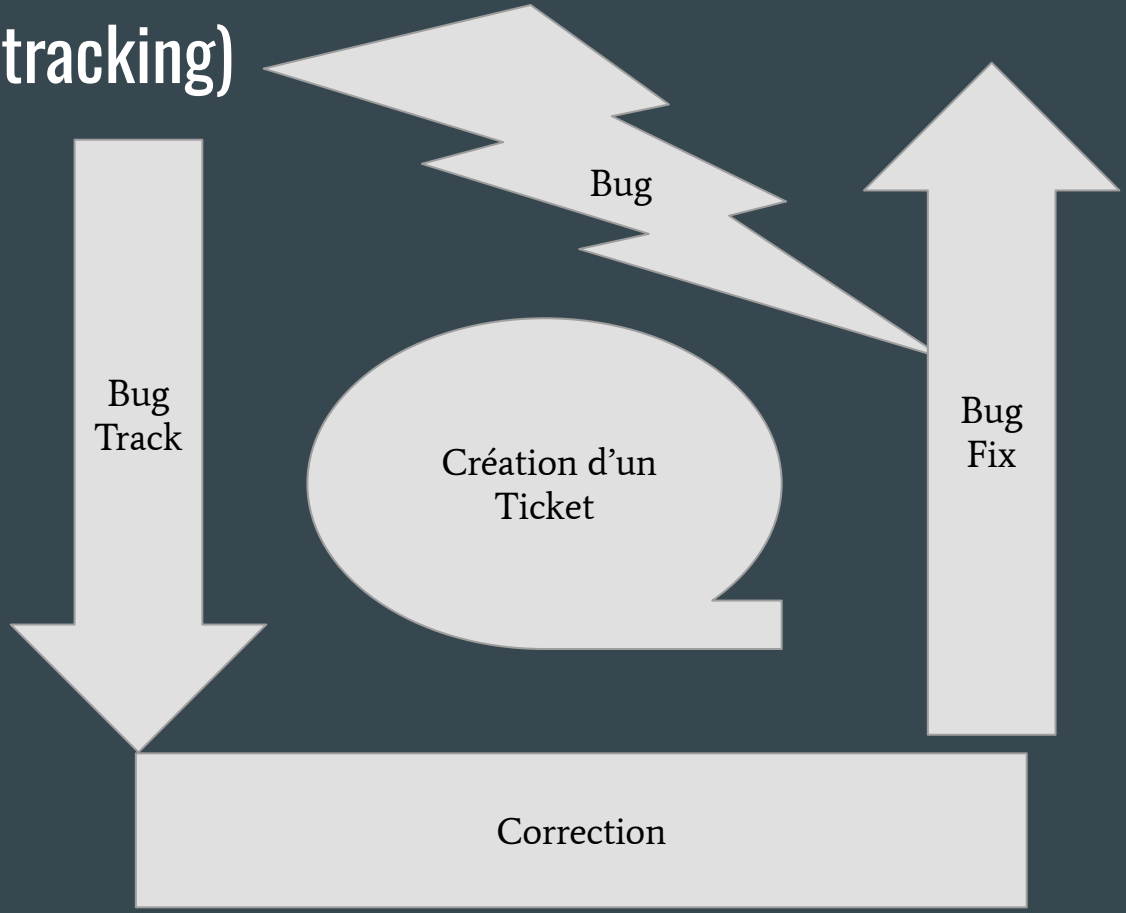
Chaque problème est enregistré dans un système de suivi avec des détails tels que la description du problème, sa gravité, sa priorité et son état.

Les développeurs peuvent attribuer des problèmes à des membres spécifiques de l'équipe pour qu'ils les résolvent.

Permet de suivre l'état de résolution des problèmes, notamment s'ils sont en cours de correction, résolus ou en attente de vérification.

Facilite la communication entre les membres de l'équipe en fournissant un historique des actions prises pour résoudre chaque problème.

Contribue à améliorer la qualité du logiciel en identifiant, documentant et résolvant efficacement les problèmes rencontrés par les utilisateurs.



Pull request (GitHub) ; Merge request (Gitlab)

- Définition : Une pull request (PR) est une demande adressée aux mainteneurs du projet pour qu'ils examinent et intègrent les modifications proposées dans une branche vers une autre (généralement de la branche de fonctionnalité vers la branche principale).
- Fonctionnalités :
  - Permet aux contributeurs de proposer des modifications au code source principal.
  - Facilite la revue de code et la discussion autour des modifications proposées.
  - Offre une plateforme pour commenter, approuver et demander des modifications sur les changements proposés.
- Utilisation :
  - Créer une PR depuis une branche de fonctionnalité vers la branche principale du dépôt.
  - Les autres membres de l'équipe peuvent examiner la PR, ajouter des commentaires, demander des modifications et approuver les changements.
  - Une fois approuvée, la PR peut être fusionnée dans la branche principale, intégrant ainsi les modifications proposées dans le code source principal.

Contenu d'une merge request (MR)

- Titre : Résumé concis des modifications proposées.
- Description : Explication détaillée des changements apportés et des problèmes résolus.
- Fichiers modifiés : Liste des fichiers affectés par la MR.
- Différences : Visualisation des modifications ligne par ligne.
- Discussion : Espace pour les commentaires, questions et discussions autour des modifications proposées.
- Liste des commits : Historique des commits inclus dans la MR.
- État de la MR : Indique si la MR est en attente de révision, approuvée, ou fusionnée.

Merge request (Gitlab)

New merge request

From `master` into `devel` Change branches

Title (required)

Test: Pull Request

Start the title with `Draft:` to prevent a merge request draft from merging before it's ready.

Description

Choose a template

Write Preview

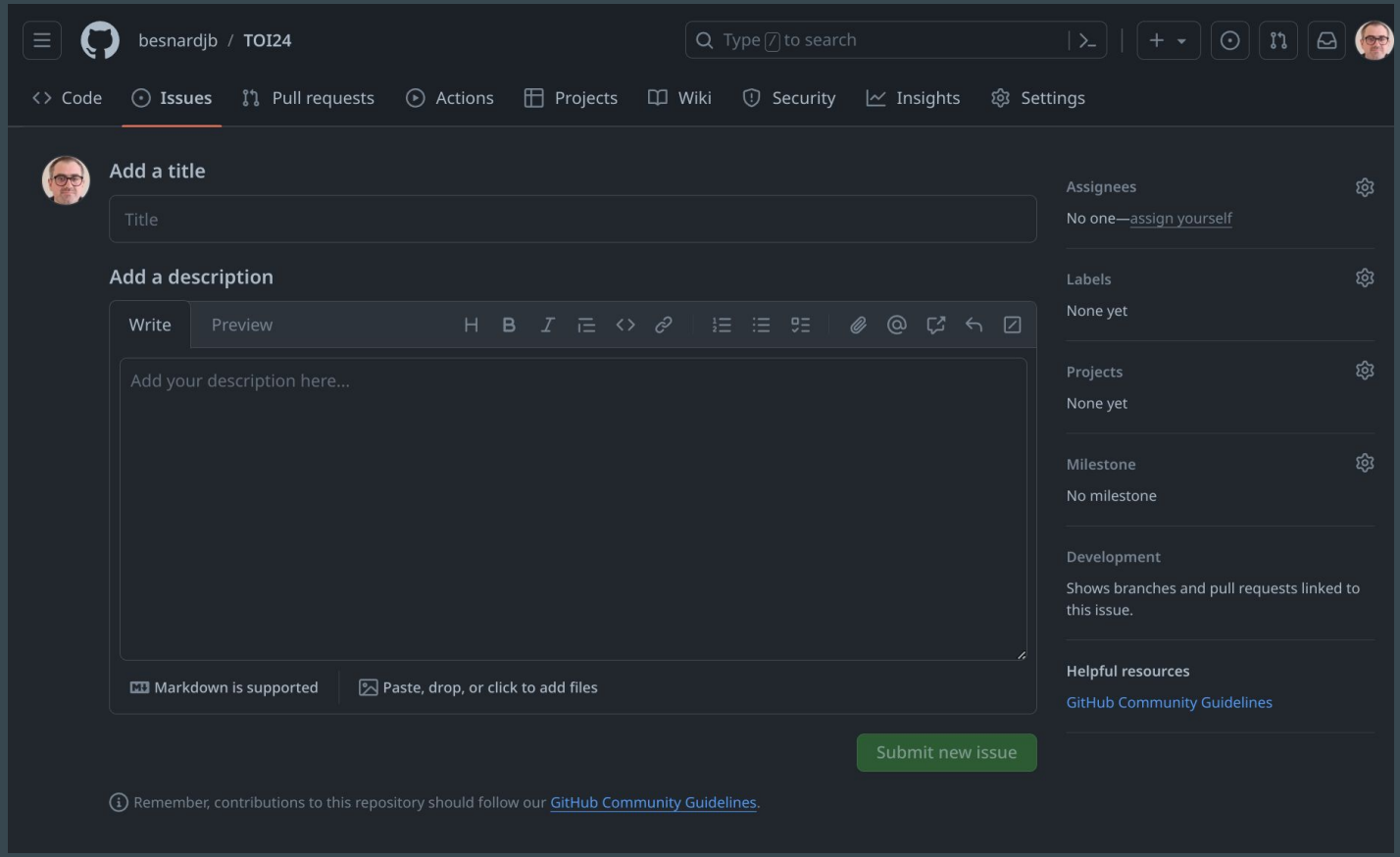
Describe the goal of the changes and what reviewers should be aware of.

Supports Markdown. For quick actions, type `/`.

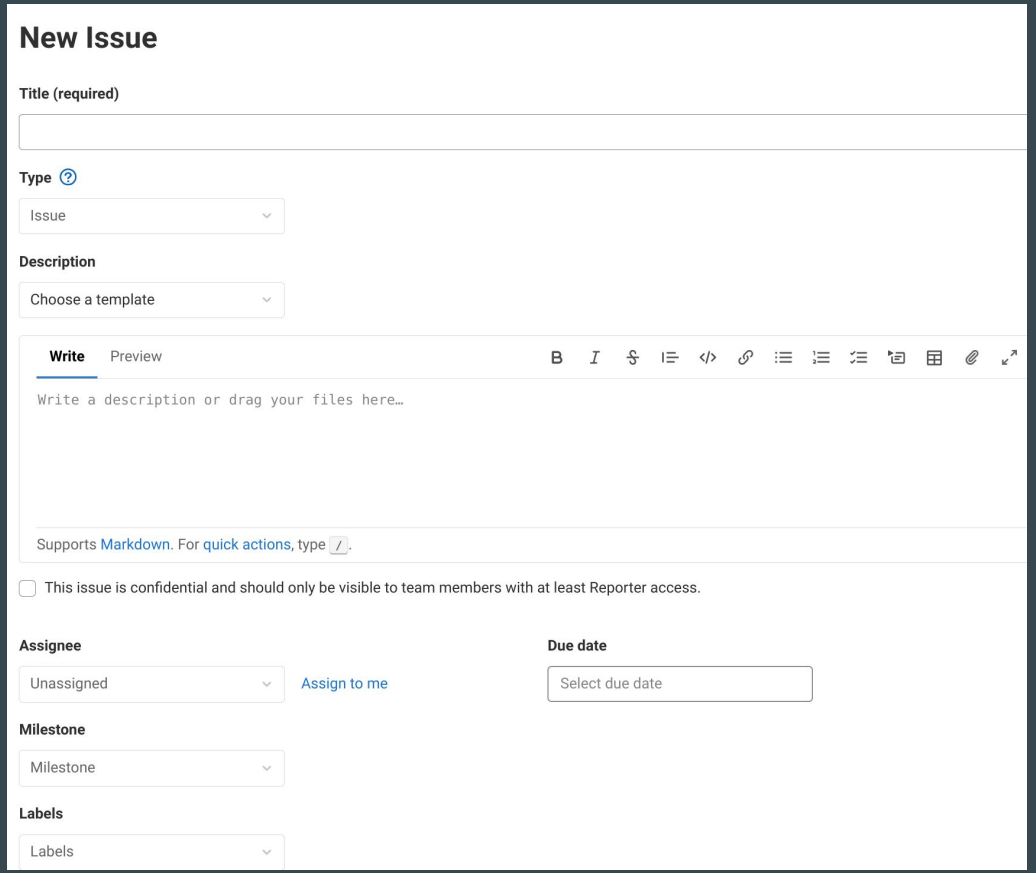
Contenu d’un ticket

- Titre du bug : Un résumé concis mais descriptif du problème rencontré.
- Description : Une explication détaillée du bug, y compris les étapes pour le reproduire, le comportement attendu et observé, ainsi que toute information pertinente pour comprendre le problème.
- Priorité : Le niveau de priorité attribué au bug, qui peut être défini en fonction de la gravité de l'impact sur l'application ou sur les utilisateurs.
- Gravité : La gravité du bug, indiquant l'importance de sa résolution, souvent déterminée en fonction de son impact sur l'expérience utilisateur ou sur le bon fonctionnement de l'application.
- Environnement : Les détails sur l'environnement dans lequel le bug a été observé, tels que le système d'exploitation, le navigateur, la version du logiciel, etc.
- Références : Toute référence à des documents, des captures d'écran, des journaux ou d'autres ressources qui pourraient aider à comprendre ou à résoudre le bug.
- Affecté à : Le membre de l'équipe chargé de résoudre le bug ou d'enquêter dessus.
- État du ticket : Indique si le bug est en attente de traitement, en cours de résolution, résolu ou en attente de vérification.
- Historique des modifications : Une liste des actions prises sur le ticket, y compris les commentaires des membres de l'équipe, les mises à jour de statut et les changements de priorité ou de gravité.
- Date de création et dernière mise à jour : Les dates auxquelles le ticket a été créé et lorsque

Ticket sur Github



Ticket sur Gitlab



Le contrôle des versions

- La gestion des versions, également connue sous le nom de contrôle de version, est un système qui enregistre les modifications apportées à un fichier ou à un ensemble de fichiers au fil du temps.
- Elle permet de garder une trace de l'historique des modifications, y compris qui a effectué les modifications, quand elles ont été effectuées et quelles étaient les modifications spécifiques.
- La gestion des versions est essentielle pour les projets de développement logiciel car elle facilite la collaboration entre les membres de l'équipe, permet le suivi des modifications et offre une sauvegarde en cas de besoin de revenir à une version antérieure du code.

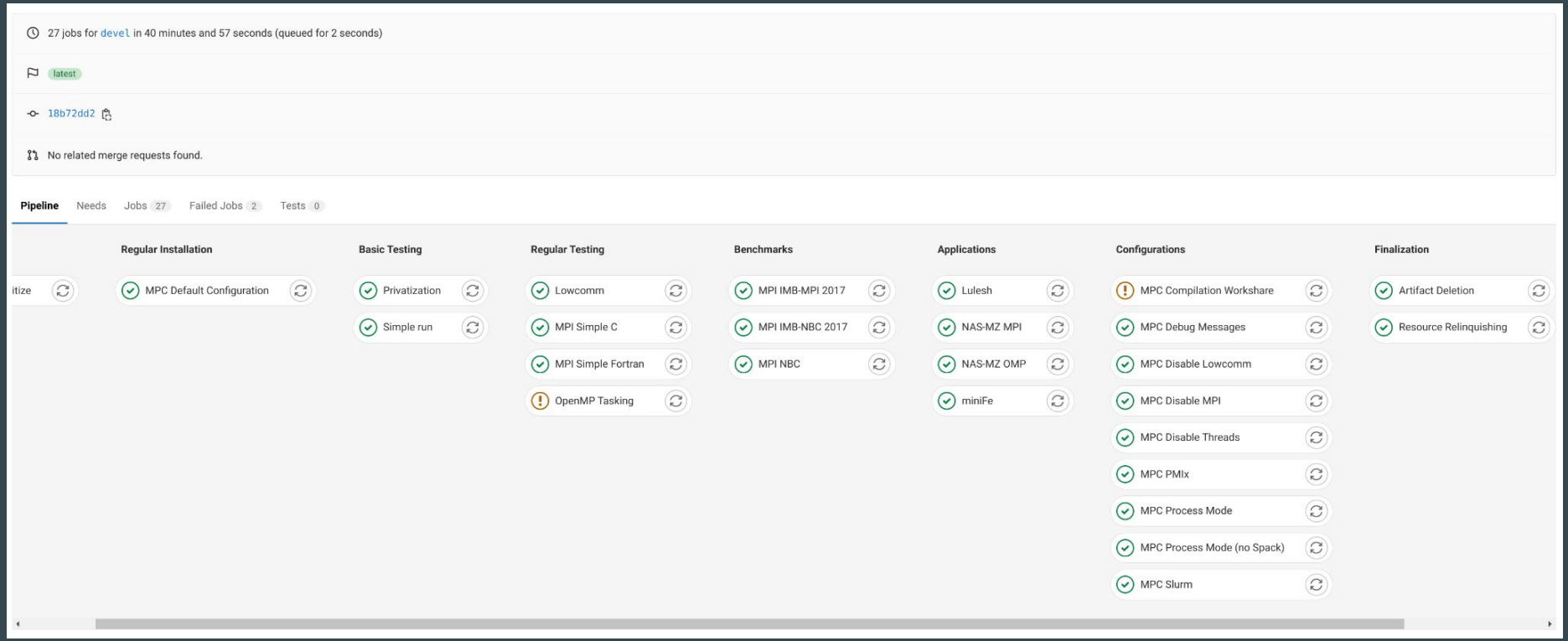
Intégration continue

- L'intégration continue (CI) est une pratique de développement logiciel qui consiste à fusionner fréquemment les modifications de code dans un référentiel partagé.
- Chaque fusion déclenche une série d'automatisations, telles que la compilation du code, l'exécution des tests automatisés et le déploiement des modifications dans un environnement de test ou de production.
- L'objectif principal de l'intégration continue est de détecter et de corriger rapidement les problèmes d'intégration, tels que les conflits de code ou les régressions, avant qu'ils ne deviennent des problèmes plus graves.

Intégration continue

- Les outils d'intégration continue populaires incluent Jenkins, Travis CI, CircleCI et GitLab CI/CD, qui offrent des fonctionnalités pour automatiser le processus d'intégration et de déploiement.
- L'intégration continue favorise une approche agile du développement en encourageant des cycles de développement courts et des livraisons fréquentes de fonctionnalités stables.
- Elle contribue également à améliorer la qualité du code en permettant une validation rapide des modifications et en fournissant un retour d'information immédiat aux développeurs.

Intégration continue



Système de contrôle des versions

Le contrôle des versions

- **Git**: le plus connu actuellement. Il est utilisé par les développeurs pour suivre l'historique des modifications apportées au code source durant son évolution, et offrir la possibilité d'envisager différentes versions du projet.
- **Mercurial** : créé en 2005 par une société canadienne, cette solution open-source est similaire à Git mais plus simple.
- Les anciens...
  - CVS: À l'origine développé par les laboratoires Bell Telephone et utilisant le modèle client/serveur, il a connu un certain succès avant de se voir remplacer par des alternatives plus modernes comme Git ou Mercurial.
  - SVN: étendu à d'autres types de fichiers que du code source (comme la documentation), cette solution permettait une gestion centralisée et collaborative, mais est maintenant moins utilisé par les développeurs qui préfèrent Git.

GIT

Création de GIT

Git a commencé en 2005, créé par Linus Torvalds. Il s'agissait d'un logiciel libre distribué pour faciliter la gestion des versions numériques (VCS). Torvalds lui-même travaillait sur le noyau Linux et BitKeeper, un logiciel de suivi des modifications du code source. La relation entre les deux s'est finalement arrêtée suite à une crise de licence en 2005.

Torvalds alors se lance dans son propre VCS pour répondre à cette contrainte et nomme ce nouveau logiciel Git (tout en gardant un lien avec les contenus numériques). Il est devenu très populaire, notamment grâce au système distribué qu'il propose.

"Le développement de Git a commencé le 3 avril 2005. Torvalds annonça ce projet le 6 avril et se mit en auto-hébergement la journée suivante. Le premier fusionnement de branches s'est produit le 18 avril. Torvalds a atteint ses objectifs ; le 29 avril, le jeune Git fut testé enregistrant des correctifs pour l'arbre du noyau Linux à un rythme de 0.67 par seconde. Le 16 juin, Git a géré la sortie de la version du noyau 2.6.12."

Commandes principales de Git

Un système de gestion de versions comme GIT permet:

- git init : Initialise un nouveau dépôt Git dans le répertoire actuel.
- git clone [url] : Clone un dépôt Git distant dans un nouveau répertoire local.
- git add [fichier] : Ajoute un fichier à l'index pour être suivi par Git.
- git commit -m "message" : Crée un nouveau commit avec les fichiers ajoutés à l'index, accompagné d'un message descriptif.
- git status : Affiche l'état actuel de la branche locale par rapport au référentiel distant, ainsi que les fichiers modifiés et non suivis.
- git branch [nom] : Crée une nouvelle branche avec le nom spécifié.
- git checkout [nom_branche] : Bascule vers une autre branche spécifiée.
- git merge [branche] : Fusionne une branche spécifiée dans la branche actuelle.
- git pull : Récupère les modifications depuis le référentiel distant et fusionne automatiquement avec la branche locale.
- git push : Envoie les modifications locales vers le référentiel distant.
- git log : Affiche l'historique des commits avec leurs messages, auteurs et horodatages.
- git diff [fichier] : Affiche les différences entre le fichier actuel et la dernière version indexée.
- git reset [fichier] : Annule les modifications d'un fichier spécifique dans la zone de préparation (index).

Définir son nom et email sur Git

git config --global user.name "Your Name"

git config --global user.email "your.e-mail@domain.com"

→ $HOME/.gitconfig

```
[user]
        name = …
        email = …
```

Par répo :

git config user.email “your-e.mail@entreprise-salaire.super”

→ <repo>/.gitconfig

Commande : git init

Description : Initialise un nouveau dépôt Git dans le répertoire actuel.

Fonctionnalités :

- Crée un nouveau dépôt Git vide dans le répertoire local.
- Initialise le suivi des modifications apportées aux fichiers dans ce répertoire.
- Permet de commencer à versionner les fichiers et de suivre leur historique de modifications.

Utilisation :

Exécuter git init dans le répertoire de travail pour initialiser un nouveau dépôt Git.

Utilisé généralement une fois au début d'un projet pour démarrer le suivi des modifications.

man git-init

Commande : git clone

Commande : git clone [url]

Description : Clone un dépôt Git distant dans un nouveau répertoire local.

Fonctionnalités :

- Copie l'intégralité du dépôt distant dans un nouveau répertoire local.
- Crée une copie exacte du dépôt, y compris toutes les branches, les commits et l'historique des modifications.
- Établit automatiquement un lien avec le dépôt distant d'origine pour faciliter la synchronisation.

Utilisation :

- Exécuter git clone [url] en remplaçant [url] par l'URL du dépôt distant que vous souhaitez cloner.

Utilisé pour récupérer un dépôt distant sur votre machine locale afin de travailler sur le code ou de contribuer au projet.

man git-clone

Commande : git add

Description : Ajoute un fichier à l'index pour être suivi par Git.

Fonctionnalités :

- Prépare un fichier spécifique pour être inclus dans le prochain commit.
- Met à jour l'index (zone de préparation) avec les modifications apportées au fichier.
- Permet à Git de suivre les changements effectués sur le fichier et de les inclure dans le prochain commit.

Utilisation :

- Exécuter git add [fichier] en remplaçant [fichier] par le nom du fichier que vous souhaitez ajouter à l'index.

Utilisé pour sélectionner les fichiers spécifiques à inclure dans le prochain commit, tout en laissant les autres fichiers non suivis.

man git-add

Commande : git branch [nom]

Description : Crée une nouvelle branche avec le nom spécifié.

Fonctionnalités :

- Permet de diviser le travail en différentes branches pour développer des fonctionnalités isolées ou des correctifs.
- Crée une nouvelle référence qui pointe vers le commit actuel, initialement basée sur la branche actuelle.
- Facilite le développement parallèle en permettant à plusieurs fonctionnalités d'être développées indépendamment les unes des autres.

Utilisation :

- Exécuter git branch [nom] en remplaçant [nom] par le nom de la nouvelle branche que vous souhaitez créer.
- Utilisé pour créer une nouvelle branche à partir du commit actuel, à partir de laquelle vous pouvez travailler sur de nouvelles fonctionnalités ou des correctifs.

Commande : git checkout [nom_branche]

Description : Bascule vers une autre branche spécifiée.

Fonctionnalités :

- Permet de naviguer entre différentes branches du dépôt Git.
- Met à jour le répertoire de travail et l'index pour refléter l'état de la branche spécifiée.
- Utilisé pour passer d'une branche à une autre afin de travailler sur des fonctionnalités distinctes ou de récupérer du code à partir d'une branche existante.

Utilisation :

- Exécuter git checkout [nom_branche] en remplaçant [nom_branche] par le nom de la branche vers laquelle vous souhaitez basculer.
- Utilisé pour passer d'une branche à une autre, récupérer du code à partir d'une branche spécifique ou créer une nouvelle branche à partir d'une branche existante.

Commande : git merge [branche]

Description : Fusionne une branche spécifiée dans la branche actuelle.

Fonctionnalités :

- Combine les modifications apportées à une branche spécifiée dans la branche actuelle.
- Intègre les changements des commits de la branche spécifiée dans l'historique de la branche actuelle.
- Permet de fusionner le travail effectué sur différentes fonctionnalités ou branches de développement dans une seule branche.

Utilisation :

- Exécuter git merge [branche] en remplaçant [branche] par le nom de la branche que vous souhaitez fusionner dans la branche actuelle.
- Utilisé pour intégrer les modifications d'une branche spécifique dans la branche actuelle, généralement après avoir terminé le développement d'une fonctionnalité ou d'une tâche sur une autre branche.

Commande : git pull

Description : Récupère les modifications depuis le référentiel distant et fusionne automatiquement avec la branche locale.

Fonctionnalités :

- Télécharge les modifications apportées au dépôt distant depuis la branche actuelle.
- Fusionne automatiquement les modifications téléchargées avec la branche locale, mettant à jour le répertoire de travail et l'historique des commits.
- Permet de synchroniser facilement le code local avec les dernières modifications apportées par d'autres contributeurs au dépôt distant.

Utilisation :

- Exécuter git pull pour télécharger et fusionner les modifications depuis le dépôt distant dans la branche locale.
- Utilisé pour mettre à jour le code local avec les dernières modifications du dépôt distant, avant de commencer à travailler sur de nouvelles fonctionnalités ou de contribuer au projet.

Commande : git push

Description : Envoie les modifications locales vers le référentiel distant.

Fonctionnalités :

- Transfère les commits locaux vers le référentiel distant, mettant à jour la branche distante avec les modifications locales.
- Permet de partager le travail réalisé localement avec d'autres contributeurs en publiant les modifications sur le référentiel distant.
- Facilite la collaboration en permettant aux membres de l'équipe de partager leurs contr**ibutions et de synchroniser le code entre les différentes branches et dépôts.**

Utilisation :

- Exécuter git push pour envoyer les modifications locales vers le référentiel distant associé à la branche actuelle.
- Utilisé après avoir effectué des commits locaux pour publier les modifications sur le référentiel distant, assurant ainsi que d'autres membres de l'équipe puissent accéder aux dernières versions du code.

Commande : git log

Description : Affiche l'historique des commits avec leurs messages, auteurs et horodatages.

Fonctionnalités :

- Affiche une liste chronologique des commits effectués dans le dépôt.
- Fournit des informations détaillées sur chaque commit, y compris le message du commit, l'auteur, la date et l'heure du commit.
- Permet de visualiser l'évolution du code au fil du temps et de suivre les modifications apportées par les différents contributeurs.

Utilisation :

- Exécuter git log pour afficher l'historique des commits dans le répertoire de travail.
- Utilisé pour examiner l'historique des modifications, identifier les commits spécifiques et comprendre l'évolution du code au fil du temps.

Voir aussi tig(1)

Commande : git diff [version] [-- fichier]

Description : Affiche les différences entre le fichier actuel et la dernière version indexée.

Fonctionnalités :

- Affiche les modifications apportées au fichier depuis la dernière fois qu'il a été ajouté à l'index.
- Met en évidence les lignes ajoutées et supprimées pour visualiser les changements avec précision.
- Permet de comparer les modifications locales par rapport à la dernière version validée dans le dépôt.

Utilisation :

- Exécuter git diff [fichier] pour afficher les différences pour un fichier spécifique.
- Utilisé pour examiner les modifications apportées à un fichier depuis la dernière fois qu'il a été ajouté à l'index, permettant ainsi de vérifier les changements avant de les valider.

Commande : git reset [version] [--<mode> | -- fichier]

Description : Annule les modifications d'un fichier spécifique dans la zone de préparation (index).

Fonctionnalités :

- Annule les modifications apportées à un fichier spécifique depuis la dernière fois qu'il a été ajouté à l'index.
- Retire le fichier de la zone de préparation (index) sans modifier les modifications locales.
- Permet de désindexer un fichier avant de le réindexer avec de nouvelles modifications.

Utilisation :

- Exécuter git reset [fichier] pour annuler les modifications d'un fichier spécifique dans la zone de préparation.
- Utilisé pour corriger les erreurs avant de valider les changements en désindexant un fichier pour le réexaminer ou pour annuler les modifications inutiles.