

# Maîtriser la CLI



# GitLab

**Nous connaissons tous l'importance des outils qui optimisent notre travail. Parmi ces outils, la CLI de GitLab CI/CD, anciennement `glab`, est pour moi une des plus importantes.**

Ce qui rend `glab` si important, c'est sa capacité à offrir presque toutes les fonctionnalités de GitLab directement depuis la ligne de commande. Cette approche est idéale pour les utilisateurs qui préfèrent une interface en ligne de commande, plus rapide et plus scriptable, par rapport à l'interface graphique web de GitLab. Avec `glab`, vous pouvez gérer les projets, les issues, les merge requests et bien plus encore.

## Installation et Configuration de `glab`

Pour profiter pleinement de `glab`, la première étape consiste à l'installer et à le configurer correctement sur votre système.

### Installation de `glab`

#### Sur Linux

Si vous êtes un utilisateur de Linux, l'installation de `glab` peut se faire facilement avec `asdf-vm` :

```
asdf plugin add glab
asdf install glab latest
asdf global glab latest
```

#### Sur Alpine Linux

Idéal pour intégrer la cli de gitlab dans vos Dockerfiles pour vos runs de Pipeline. Cela va vous ouvrir des portes que vous n'aviez pas imaginées :

```
apk add --no-cache glab
```

## Sur macOS

Pour les utilisateurs de macOS, `glab` peut être installé via Homebrew. Ouvrez simplement votre terminal et tapez :

```
brew install glab
```

## Sur Windows

Sous Windows, `glab` peut être installé à l'aide de Scoop ou de Chocolatey. Avec Chocolatey, la commande est la suivante :

```
choco install glab
```

# Les Principales Fonctionnalités de la CLI Gitlab

`glab` offre une suite complète d'options qui permettent une gestion efficace des projets GitLab. Voici une liste des principales commandes disponibles dans `glab`, chacune accompagnée d'une brève description de sa fonction :

1. `alias` : Permet de créer des alias pour les commandes `glab`, simplifiant ainsi l'utilisation des commandes fréquentes.
2. `api` : Offre un accès direct à l'API de GitLab, permettant d'effectuer des requêtes API personnalisées.
3. `auth` : Utilisée pour l'authentification et la configuration de vos identifiants GitLab dans `glab`.
4. `check-update` : Vérifie si une nouvelle version de `glab` est disponible.
5. `ci` : Permet de gérer les pipelines de CI (Continuous Integration), y compris leur création, visualisation et suppression.
6. `completion` : Génère des scripts d'auto-complétion pour différents shells, facilitant la saisie des commandes `glab`.

7. `config` : Utilisée pour configurer les paramètres globaux de `glab`.
8. `incident` : Permet de gérer les incidents, notamment leur création, mise à jour et fermeture.
9. `issue` : Fournit des outils pour gérer les issues, incluant la création, la liste, la mise à jour et la fermeture.
10. `label` : Permet de gérer les labels pour les issues et les merge requests.
11. `mr` : Fournit des commandes pour gérer les merge requests, y compris leur création, liste, mise à jour et fusion.
12. `release` : Permet de gérer les releases, incluant leur création, listage et suppression.
13. `repo` : Offre des outils pour gérer les dépôts GitLab, y compris la création, le clonage et la gestion des branches.
14. `schedule` : Permet de gérer les pipelines planifiés (scheduled pipelines).
15. `snippet` : Fournit des commandes pour créer et gérer des extraits de code (snippets).
16. `ssh-key` : Permet de gérer les clés SSH associées à votre compte GitLab.
17. `user` : Offre des outils pour visualiser et gérer les informations utilisateur sur GitLab.
18. `variable` : Permet de gérer les variables de CI/CD au niveau du projet ou du groupe.

Chacune de ces commandes est conçue pour intégrer GitLab de manière plus profonde dans votre flux de travail DevOps, en rendant la gestion de vos projets, issues, merge requests et CI/CD plus efficace et automatisée.

## Configuration de `glab`

Une fois `glab` installé, la prochaine étape est de le configurer pour se connecter à votre compte GitLab. Cette connexion est essentielle pour permettre à `glab` d'interagir avec vos projets GitLab.

## Les commandes de configuration

### Authentification avec `glab auth`

La commande `glab auth` est utilisée pour l'authentification avec GitLab. Elle permet de configurer vos identifiants pour une intégration transparente avec vos projets GitLab.

#### **Authentification Interactive :**

Tapez la commande suivante :

```
* Logging into gitlab.com
? How would you like to login? Token
```

```
Tip: you can generate a Personal Access Token here https://gitlab.com/-
/profile/personal_access_tokens?scopes=api,write_repository
The minimum required scopes are 'api' and 'write_repository'.
? Paste your authentication token: *****
? Choose default git protocol SSH
* glab config set -h gitlab.com git_protocol ssh
✓ Configured git protocol
* glab config set -h gitlab.com api_protocol https
✓ Configured API protocol
✓ Logged in as Bob74
```

Suivez les instructions pour entrer le token d'accès personnel. Choisissez l'instance GitLab (GitLab.com ou instance auto-gérée).

### Authentication Non-Interactive :

Utilisez la variable `GITLAB_TOKEN` pour une authentification sans interaction, idéale pour les scripts ou les environnements CI/CD.

Pour vérifier que tout est correctement configuré, vous pouvez utiliser la commande suivante :

```
glab auth status 10:54:29

gitlab.com
x gitlab.com: api call failed: GET https://gitlab.com/api/v4/user: 401 {message:
401 Unauthorized}
✓ Git operations for gitlab.com configured to use ssh protocol.
✓ API calls for gitlab.com are made over https protocol
✓ REST API Endpoint: https://gitlab.com/api/v4/
✓ GraphQL Endpoint: https://gitlab.com/api/graphql/
x No token provided
```

Ici un problème d'authentification.

## Gestion des Paramètres Globaux avec `glab config`

`glab config` permet de personnaliser, `set` l'utilisation de `glab` en configurant des paramètres globaux.

- Définissez votre éditeur de texte préféré : `glab config set editor vim`.

- Configurez le navigateur pour ouvrir les liens : `glab config set browser firefox`.

Pour afficher les paramètres, il faut utiliser l'option `get` :

```
glab config get browser
/home/bob/.vscode-
server/bin/0ee08df0cf4527e40edc9aa28f4b5bd38bbff2b2/bin/helpers/browser.sh
```

## Gestion des Clés SSH avec `glab ssh-key`

La commande `glab ssh-key` aide à gérer les clés SSH pour sécuriser vos opérations Git.

- Ajoutez une clé SSH à GitLab : `glab ssh-key add /chemin/de/la/cle.pub -t "Titre de la clé"`.
- Listez les clés SSH associées à votre compte : `glab ssh-key list`.
- Supprimez une clé SSH spécifique : `glab ssh-key delete <id_de_la_clé>`.

## Configuration de l'Auto-Complétion `glab completion`

La commande `glab completion` de la CLI GitLab permet de générer des fichiers de configuration pour activer la complétion automatique (auto-complétion) dans différents shells, facilitant ainsi l'utilisation de la CLI GitLab.

### Bash

Intégrez-le dans votre configuration Bash en ajoutant :

```
source <(glab completion bash)
```

à votre fichier `~/.bashrc`.

### Zsh

Ajoutez ce script à votre fichier de configuration Zsh (`~/.zshrc`) :

```
source <(glab completion -s zsh); compdef _glab glab
```

### Fish

Intégrez-le dans votre configuration Fish en ajoutant :

```
glab completion -s fish > ~/.config/fish/completions/glab.fish
```

# Les variables d'environnement

L'utilisation de `glab` peut être personnalisée à travers diverses variables d'environnement et commandes de configuration.

- `GITLAB_TOKEN` : Évite la nécessité de s'authentifier à chaque requête API.
- `GITLAB_URI` ou `GITLAB_HOST` : Spécifie l'URL du serveur GitLab pour les instances auto-gérées.
  - Exemple : `https://gitlab.example.com` (par défaut `https://gitlab.com`).
- `GITLAB_API_HOST` : Spécifie l'hôte où se trouve l'API GitLab, utile lorsque Git et l'API sont sur des (sous)domaines distincts.
- `GITLAB_REPO` : Définit le dépôt GitLab par défaut pour les commandes acceptant l'option `--repo`.
- `GITLAB_GROUP` : Définit le groupe GitLab par défaut pour lister les merge requests, issues et variables.
- `REMOTE_ALIAS` ou `GIT_REMOTE_URL_VAR` : Variable ou alias de remote git contenant l'URL GitLab.
- `GLAB_CONFIG_DIR` : Répertoire de configuration globale de `glab`.
  - Par défaut : `~/.config/glab-cli/`.
- `VISUAL`, `EDITOR` : Définit l'éditeur de texte pour la rédaction.
- `BROWSER` : Définit le navigateur web pour ouvrir les liens.
- `GLAMOUR_STYLE` : Style de rendu Markdown personnalisé (options : `dark`, `light`, `notty`).
  - `NO_COLOR` : Désactive la coloration ANSI dans la sortie.
- `FORCE_HYPERLINKS` : Force l'affichage de liens hypertextes, même en dehors d'un TTY.

# Les Commandes Générales `glab`

Les commandes générales sont celles qui ne sont pas spécifiques aux repositories. Elles fonctionnent même si elles ne sont pas exécutées depuis un répertoire contenant un projet gitlab.

## Une Nouvelle Version ? `glab check-update`

La commande update permet de vérifier si une nouvelle version de `glab` est disponible.

```
glab check-update  
You are already using the latest version of glab
```

# Gestion des Repositories avec `glab repo`

La commande `glab repo` est essentielle pour gérer les repositories sur GitLab, offrant de nombreuses options pour une gestion efficace.

- `archive`: Pour archiver un repository.
- `clone`: Pour cloner un repository GitLab.
- `contributors`: Pour afficher les contributeurs d'un repository.
- `create`: Pour créer un nouveau repository.
- `delete`: Pour supprimer un repository.
- `fork`: Pour faire un fork d'un repository.
- `list`: Pour lister les repositories disponibles.
- `mirror`: Pour configurer un miroir pour un repository.
- `search`: Pour rechercher des repositories.
- `transfer`: Pour transférer un repository à un autre utilisateur ou groupe.
- `view`: Pour afficher les détails d'un repository.

## Exemples de commande :

- Lister les repositories disponibles :

```
glab repo list
Showing 30 of 57 projects (Page 1 of 2)

mon-site1/docusaurus git@gitlab.com:mon-site1/docusaurus.git
dockerfiles6/images/container-structure-test
git@gitlab.com:dockerfiles6/images/container-st...
Bob74/aws-blog git@gitlab.com:Bob74/aws-blog.git
dockerfiles6/images/demo-cosign git@gitlab.com:dockerfiles6/images/demo-cosign.git
```

- Rechercher des repositories :

```
glab repo search -s docker
Showing results for "docker"
🔍 nexylan/docker/docker On steroids Docker image. 0 stars 0 forks 0 issues updated
about 19 days ago
https://gitlab.com/nexylan/docker/docker
🔍 jitesoft/dockerfiles/docker Docker image with docker and 0 stars 0 forks 0 issues
updated about 2 months ago
```

```
docker in docker! https://www.docker.com/
https://gitlab.com/jitesoft/dockerfiles/docker
```

En vrac :

```
# Create
glab repo create --group glab-cli
glab repo create my-project
# Un projet privé dans un groupe (existant) avec initialisation du README
glab repo create glab-cli/my-project -d 'ma description' --private --readme -t
ansible,devops

# Review
glab repo view my-project
glab repo view user/repo
glab repo view group/namespace/repo
```

## Les requêtes d'API avec `glab api`

La commande `glab api` permet de lancer des requêtes à l'API GitLab.

### Requetes Standards

La requête `glab api` permet de lancer des call sur l'API de Gitlab. Vous pouvez retrouver les endpoints d'API sur la [documentation](#)

```
glab api /projects/:id/repository/branches/

[
  {
    "name": "main",
    "commit": {
      "id": "011677bac1fb93bac5cea563c0697287c6c926f8",
      "short_id": "011677ba",
      "created_at": "2023-02-05T10:41:25.000+01:00",
```



```

    "parent_ids": ["85c5234352df02cad639787560c4137ec75b5a61"],
    "title": "Initial commit",
    "message": "Initial commit\n",
    "author_name": "Robert Stéphane",
    "author_email": "robert.stephane.28@gmail.com",
    "authored_date": "2023-02-05T10:41:25.000+01:00",
    "committer_name": "Robert Stéphane",
    "committer_email": "robert.stephane.28@gmail.com",
    "committed_date": "2023-02-05T10:41:25.000+01:00",
    "trailers": {},
    "extended_trailers": {},
    "web_url": "https://gitlab.com/Bob74/ansible-gendoc/-
/commit/011677bac1fb93bac5cea563c0697287c6c926f8"
  },
  "merged": false,
  "protected": true,
  "developers_can_push": false,
  ...

```

## Requêtes GraphQL

`glab api` supporte l'utilisation de GraphQL, un langage de requête puissant pour les APIs. Avec l'option GraphQL, vous pouvez :

- Créer des requêtes complexes et précises.
- Obtenir des données spécifiques en une seule requête.
- Manipuler et accéder aux données de manière plus flexible et efficace.

Voici comment vous pourriez utiliser `glab api` pour effectuer une requête GraphQL :

```

glab api graphql -f query='
query {
  project(fullPath: "gitlab-org/gitlab-docs") {
    name
    forksCount
    statistics {
      wikiSize
    }
    issuesEnabled
    boards {

```

```
nodes {  
  id  
  name  
}  
}  
}  
}
```

Ce type de requête permet une interaction profonde avec les données de GitLab, offrant une flexibilité et une puissance bien supérieures aux requêtes API REST classiques.

# Les commandes spécifiques aux projets

Les commandes décrites ci-dessous ne fonctionnent que si elles sont exécutées depuis un répertoire contenant un repository gitlab.

## Gestion des Labels avec `glab label`

`glab label` est une commande clé pour la gestion des labels dans les projets GitLab, offrant diverses options pour une organisation efficace des issues et des merge requests.

Voici la liste des principales options :

- `create` : Pour créer un nouveau label.
- `list` : Pour lister tous les labels d'un projet.

Ces options facilitent la création, la visualisation, la modification et la suppression de labels, permettant ainsi une meilleure organisation et catégorisation des éléments de travail dans vos projets GitLab.

## Gestion des Issues avec `glab issue`

La commande `glab issue` fournit un ensemble complet d'outils pour gérer les issues dans vos projets GitLab.

Voici la liste des principales options :

- `board` : Pour visualiser et gérer les boards d'issues.

- `close` : Pour fermer une issue.
- `create` : Pour créer une nouvelle issue.
- `delete` : Pour supprimer une issue.
- `list` : Pour lister les issues.
- `note` : Pour ajouter une note à une issue.
- `reopen` : Pour rouvrir une issue fermée.
- `subscribe` : Pour s'abonner aux mises à jour d'une issue.
- `unsubscribe` : Pour se désabonner des mises à jour d'une issue.
- `update` : Pour mettre à jour une issue.
- `view` : Pour afficher les détails d'une issue.

## Exemples :

```
glab issue list
```

```
No open issues match your search in mon-site1/docusaurus
```

# Gestion des Merge Requests avec `glab mr`

`glab mr` est une commande qui permet de gérer les merge requests dans GitLab, offrant une large gamme d'options pour une gestion efficace.

Voici la liste des principales options :

- `approve` : Approuvez une merge request.
- `approvers` : Gérez les approbateurs d'une merge request.
- `checkout` : Récupérez localement une merge request.
- `close` : Fermez une merge request.
- `create` : Créez une nouvelle merge request.
- `delete` : Supprimez une merge request.
- `diff` : Affichez les différences dans une merge request.
- `fork` : Faites un fork d'une merge request.
- `issues` : Listez les issues associées à une merge request.
- `list` : Listez les merge requests.
- `merge` : Fusionnez une merge request.
- `note` : Ajoutez une note à une merge request.
- `rebase` : Rebasez une merge request.
- `reopen` : Rouvrez une merge request fermée.
- `revoke` : Révoquez une approbation de merge request.
- `subscribe` : Abonnez-vous aux mises à jour d'une merge request.
- `todo` : Ajoutez une merge request à votre liste de tâches.
- `unsubscribe` : Désabonnez-vous des mises à jour d'une merge request.
- `update` : Mettez à jour une merge request.
- `view` : Affichez les détails d'une merge request.

Ces options permettent une gestion complète et flexible des merge requests dans vos projets GitLab. Chacune de ces options offre une flexibilité et un contrôle précis sur la gestion des merge requests, essentiels pour une collaboration et une intégration de code efficaces dans vos projets GitLab.

## Gestion des Incidents avec `glab incident`

`glab incident` offre des commandes spécifiques pour gérer efficacement les incidents dans vos projets GitLab.

- `close` : Pour fermer un incident.
- `list` : Pour lister les incidents d'un projet.
- `note` : Pour commenter un incident dans GitLab.
- `reopen` : Pour rouvrir un incident résolu.
- `subscribe` : Pour s'abonner à un incident.
- `unsubscribe` : Pour se désabonner d'un incident.
- `view` : Pour afficher le titre, le corps et d'autres informations sur un incident.

Ces options permettent un suivi et une gestion complets des incidents, essentiels pour la résolution rapide et efficace des problèmes critiques.

## Gestion des Releases avec `glab release`

La commande `glab release` fournit un ensemble d'outils pour la gestion des releases dans vos projets GitLab.

- `create` : Pour créer une nouvelle release.
- `delete` : Pour supprimer une release.
- `download` : Pour télécharger les assets d'une release.
- `list` : Pour lister toutes les releases d'un projet.
- `upload` : Pour téléverser des assets à une release.
- `view` : Pour afficher les détails d'une release.

Ces commandes couvrent l'ensemble du cycle de vie d'une release, de sa création à sa suppression, en facilitant la distribution et la gestion des versions dans vos projets GitLab.

## Création de snippets avec `glab snippet`

La commande `glab snippet` est un outil puissant pour gérer les extraits de code (snippets) dans GitLab, permettant un partage et une collaboration efficaces sur des morceaux de code ou des informations techniques.

**Exemple :**

```
glab snippet create sidebars.js --title "Title of the snippet"
* Creating snippet in mon-site1/docusaurus
$3634399 Title of the snippet (sidebars.js)
https://gitlab.com/mon-site1/docusaurus/-/snippets/3634399
```

## Lister les événements avec `glab user events`

La commande `glab user events` est spécialement conçue pour suivre les activités et les événements associés à un utilisateur GitLab. Elle est particulièrement utile pour surveiller l'engagement et les contributions au sein d'un projet ou à travers plusieurs projets sur GitLab.

Cette commande peut être utilisée pour suivre les contributions récentes d'un utilisateur, comme les commits, les merge requests et les commentaires sur les issues.

```
glab user events

Pushed to branch dev at mon-site / docusaurus
"corrections"
```

# Les commandes pour les pipelines

## Gestion des Pipelines avec `glab ci`

La commande `glab ci` est essentielle pour gérer les aspects de l'intégration continue (CI) dans GitLab, offrant un large éventail de fonctionnalités pour optimiser et surveiller les pipelines CI.

Une option importante qui permet d'indiquer un autre repository :

```
-R, --repo OWNER/REPO
```

Les principales options :

- `artifact` : Pour gérer les artefacts d'un job CI.
- `ci` : Pour afficher les pipelines récents.
- `delete` : Pour supprimer un pipeline.

- `get` : Pour obtenir des détails spécifiques sur un pipeline.
- `lint` : Pour valider le fichier `.gitlab-ci.yml`.
- `list` : Pour lister les pipelines.
- `retry` : Pour relancer un pipeline échoué.
- `run` : Pour exécuter un pipeline manuellement.
- `status` : Pour vérifier le statut d'un pipeline.
- `trace` : Pour suivre les logs d'exécution d'un job CI.
- `trigger` : Pour déclencher un pipeline via une API.
- `view` : Pour visualiser les détails d'un pipeline.

Ces commandes permettent une gestion complète et efficace des pipelines CI, depuis leur création jusqu'à leur surveillance, en passant par l'optimisation et le débogage.

## Planifier des pipelines avec `glab schedule`

La commande `glab schedule` est conçue pour gérer les pipelines planifiés dans GitLab, permettant une automatisation et une planification efficaces des tâches CI/CD.

- `create` : Pour planifier un nouveau pipeline. Cette option permet de configurer des pipelines pour s'exécuter automatiquement à des moments spécifiés.
- `delete` : Pour supprimer un pipeline planifié en utilisant son ID. Cela est utile pour annuler les pipelines qui ne sont plus nécessaires.
- `list` : Pour obtenir la liste des pipelines planifiés. Cette commande donne un aperçu de tous les pipelines programmés dans un projet.
- `run` : Pour exécuter manuellement un pipeline planifié. Cela permet de déclencher un pipeline spécifique sans attendre sa prochaine exécution programmée.

Ces commandes offrent une gestion complète des pipelines planifiés, de leur création à leur suppression, en passant par leur exécution et leur surveillance.

## Cas d'Usage de `glab`

L'utilisation de `glab` dans ces contextes permet d'optimiser les workflows, d'augmenter l'efficacité et de réduire les délais de développement et de déploiement.

## Utilisation sur le Poste de Travail

Sur le poste de travail d'un développeur ou d'un administrateur système, `glab` permet une interaction rapide et efficace avec GitLab.

- **Gestion des Repositories** : Clonage, création et gestion des repositories directement depuis le terminal.

- **Suivi des Incidents et Issues** : Gestion rapide des incidents et issues sans passer par l'interface web.
- **Requêtes API Personnalisées** : Utilisation de `glab api` pour des requêtes spécifiques, facilitant l'intégration avec d'autres outils ou scripts.

## Utilisation dans les Pipelines CI/CD

`glab` est particulièrement utile dans les pipelines CI/CD pour automatiser diverses tâches liées à GitLab.

- **Automatisation des Merge Requests** : Création automatique de merge requests suite à des commits ou des changements dans les branches.
- **Gestion des Issues** : Ouverture et mise à jour des issues basées sur les événements de pipeline.
- **Reporting** : Envoi de rapports de build ou de tests directement depuis les pipelines.

---

Revision #1

Created 3 February 2025 18:04:13 by Nicolas

Updated 3 February 2025 18:09:18 by Nicolas