

INTRODUCTION Å GIT

QU'EST-CE QUE GIT

Git est un **système de contrôle de version décentralisé** et **collaboratif** permettant de gérer les modifications d'un **projet informatique**.

SYSTÈME DE CONTRÔLE DE VERSION

Permet de **suivre les modifications** apportées à un projet et de **revenir à une version antérieure** si nécessaire.

DÉCENTRALISÉ

Chaque utilisateur possède une copie complète du dépôt, ce qui permet de **travailler hors ligne** et de faciliter la **collaboration**.

COLLABORATIF

Permet à **plusieurs personnes** de travailler **simultanément** sur un même projet sans écraser les modifications des autres.

AVANTAGES DE GIT

- **Gestion des versions:** suivre l'évolution du code et revenir à une version précédente si nécessaire.
- **Collaboration:** facilite le travail en équipe en gérant les modifications de chaque membre de l'équipe.
- **Sauvegarde et récupération:** en cas de perte de fichiers, il est possible de revenir à une version antérieure du code.

INSTALLATION DE GIT

WINDOWS

Téléchargez l'installateur de Git pour Windows à partir du site officiel : <https://git-scm.com/>

LINUX

Installez **Git** à partir du gestionnaire de paquets de votre distribution. Par exemple, pour Debian/Ubuntu :

```
sudo apt-get install git
```

MACOS

Installez **Git** à l'aide de **Homebrew** :

```
brew install git
```

Sinon, téléchargez l'installateur pour **MacOS** à partir du site officiel : <https://git-scm.com/>

CONFIGURATION DE GIT

INITIALISATION D'UN DÉPÔT

COMMANDÉ `git init`

Pour initialiser un nouveau dépôt Git, utilisez la commande `git init` dans le dossier de votre projet :

```
cd your_project_folder  
git init
```

.GITIGNORE

Un fichier `.gitignore` permet de spécifier les fichiers et dossiers que **Git** doit ignorer lors de l'**ajout** et la **création des commits**. Exemple de contenu pour un fichier `.gitignore`:

```
*.log  
*.tmp  
/node_modules/
```

CONFIGURATION GLOBALE

NOM ET EMAIL UTILISATEUR

En raison du caractère **décentralisé** de Git, chacun doit configurer son nom et son adresse email. Utilisez les commandes suivantes :

```
git config --global user.name "Votre Nom"  
git config --global user.email "votre@email.com"
```

CONFIGURER UN ÉDITEUR

Pour définir votre **éditeur préféré** (par exemple, **Vim**) pour Git, utilisez la commande :

```
git config --global core.editor "vim"
```

CONFIGURER UN OUTIL DE DIFFÉRENCIATION

Git utilise un **outil de différenciation externe** pour afficher les différences entre les fichiers. Pour configurer cet outil (par exemple, "diff-so-fancy"), utilisez la commande:

```
git config --global core.pager "diff-so-fancy | less --tabs=4 -RFX"
```

CLONER UN DÉPÔT

COMMANDÉ git clone

Pour cloner un **dépôt distant** (par exemple, depuis **GitHub**), utilisez la commande `git clone` suivie de l'URL du dépôt.

```
git clone https://github.com/user/repo.git
```

CLONAGE AVEC HTTPS ET SSH

Il est possible de cloner un dépôt en utilisant une connexion sécurisée **HTTPS** ou **SSH**.

- **HTTPS**: `https://github.com/user/repo.git`
- **SSH**: `git@github.com:user/repo.git`

INITIATION À L'OUTIL DEVOPS GIT

UTILISATION BASIQUE DE GIT

COMMANDES PRINCIPALES

- git add : Ajouter des fichiers à la **staging area**
- git commit : Créer un **commit** avec un message
- git status : Afficher l'état du **dépôt**
- git log : Afficher l'**historique** des commits

git add

- Ajoute des fichiers **modifiés** à la **staging area**
- Prépare les fichiers pour le **prochain commit**
- Exemple: git add fichier1.txt fichier2.txt

git commit

- Enregistre les changements de la **staging area** dans l'historique
- Nécessite un **message** pour décrire les modifications
- Exemple: git commit -m "Ajout de fichiers"

git status

- Montre l'état actuel du dépôt
- Liste les fichiers **modifiés, ajoutés ou supprimés**
- Exemple: git status

git log

- Affiche l'historique des **commits**
- Montre les informations sur l'auteur, la date et le message de chaque commit
- Exemple : git log

TRAVAILLER AVEC DES BRANCHES

- `git branch` : Créer, lister et supprimer des **branches**
- `git checkout` : Changer de **branche** et annuler des modifications

CRÉATION D'UNE NOUVELLE BRANCHE AVEC `git branch`

- Crée une **nouvelle branche** à partir de la branche actuelle
- N'affecte **pas** la branche actuelle
- Exemple : `git branch ma-nouvelle-branche`

LISTE DES BRANCHES AVEC `git branch`

- Affiche la liste des **branches locales**
- Montre la **branche actuelle** avec un astérisque (*)
- Exemple : `git branch`

CHANGER DE BRANCHE AVEC `git checkout`

- Passer à une autre branche en mettant à jour les fichiers du dépôt
- Ramène la **dernière version** de la branche sélectionnée
- Exemple : `git checkout ma-nouvelle-branche`

ANNULER DES MODIFICATIONS AVEC `git checkout`

- **Revenir** à l'état du **dernier commit** sur la branche actuelle
- **Annule** toutes les modifications en cours
- Exemple: `git checkout -- fichier.txt`

COLLABORATION AVEC GIT

COMMANDES DE SYNCHRONISATION

git fetch

Récupérer les **modifications distantes** sans les fusionner.

```
git fetch
```

git pull

Fusionner les modifications **distantes** avec la **branche courante**.

```
git pull
```

git push

Envoyer les **modifications locales** vers le **dépôt distant**.

```
git push
```

FUSIONNER LES BRANCHES

COMMANDÉ git merge

Fusionner deux **branches** ensemble.

```
git merge <branch-name>
```

RÉSOUDRE LES CONFLITS

1. Ouvrir les fichiers avec des **conflits**
2. Modifier les parties en **conflict**
3. Ajouter et valider les fichiers modifiés

```
git add <file-name>
git commit -m "Resolve merge conflict"
```

BONNES PRATIQUES AVEC GIT

MESSAGES DE COMMIT CLAIRS

- Ecrire des messages **compréhensibles** et **informatifs**
- Décrire la raison des modifications en quelques mots
- Utiliser le format "**Sujet + verbe + complément**"

UTILISER DES BRANCHES POUR LES FONCTIONNALITÉS

- Créer une **nouvelle branche** pour chaque fonctionnalité ou correction de bug
- Ne pas travailler directement sur la branche principale (**main** ou **master**)
- Fusionner les branches dans la branche principale une fois terminées

GARDER L'HISTORIQUE PROPRE

- Éviter les **commits trop fréquents** et / ou **trop petits**
- Regrouper les **modifications liées** dans un seul commit
- Utiliser git rebase pour modifier l'historique si nécessaire

RÉGULIÈREMENT SYNCHRONISER AVEC LE DÉPÔT DISTANT

- git fetch: récupérer les modifications des autres **collaborateurs**
- git pull: fusionner les modifications distantes avec la **branche locale**
- git push: envoyer les modifications locales sur le **dépôt distant**

COMMUNICATION ET COORDINATION AVEC L'ÉQUIPE

- Informer le reste de l'équipe des **mises à jour importantes**
- Régler les **conflits** de manière collégiale
- Exprimer clairement les **objectifs** et les **attentes**

RESSOURCES COMPLÉMENTAIRES

- **Tutoriels en ligne**
- **Documentation officielle**
- **Forums et communautés**

RESSOURCES COMPLÉMENTAIRES

RESSOURCES COMPLÉMENTAIRES

Les ressources complémentaires vous permettent d'**approfondir** vos connaissances sur **Git** et de trouver de l'aide en cas de besoin.

- Documentation officielle de Git
- GitHub - Astuces et conseils
- Pro Git - Livre en ligne gratuit
- Atlassian - Tutoriels sur Git

TUTORIELS EN LIGNE

- [Pro Git book](#)
- [Atlassian Git tutorial](#)
- [GitKraken Git tutorial](#)

DOCUMENTATION OFFICIELLE

- [Git-scm.com](#)
- [GitHub Help](#)
- [GitLab Docs](#)

FORUMS ET COMMUNAUTÉS

- **Stack Overflow** Git questions
- **GitHub** Community Forum
- **GitLab** Community

N'hésitez pas à consulter ces ressources pour approfondir vos connaissances et résoudre les problèmes que vous pourriez rencontrer en utilisant Git.

