Rapport sur Git et GitHub

Guide Complet des Notions de Base et Commandes

Documentation Technique

Nour El Houda Aljane

7 janvier 2026

Table des matières

1 Introduction

Git est un système de contrôle de version distribué créé par Linus Torvalds en 2005. **GitHub** est une plateforme d'hébergement de code qui utilise Git pour le versionnement.

$$\text{Git} + \text{Cloud Hosting} + \text{Collaboration Tools} = \text{GitHub} \tag{1}$$

Git permet aux développeurs de :
— Suivre l'historique des modifications
— Collaborer efficacement en équipe
— Expérimenter sans risque avec les branches
— Revenir à des versions antérieures du code

2 Notions de Base

2.1 Dépôt (Repository)

Un dépôt est un espace de stockage pour votre projet contenant tous les fichiers et l'historique des versions.

$$\text{Repository} = \{\text{Files}, \text{History}, \text{Branches}\} \tag{2}$$

2.2 Commit

Un commit est un instantané de votre projet à un moment donné. Chaque commit possède :
— Un identifiant unique (hash SHA-1)
— Un message descriptif
— Un auteur et une date
— Un ou plusieurs commits parents

$$\text{Commit}_n = f(\text{Changes}, \text{Message}, \text{Author}, \text{Timestamp}) \tag{3}$$

2.3 Branches

Les branches permettent de travailler sur différentes versions du code simultanément sans affecter la branche principale.

$$\text{Branch}_{\text{feature}} \parallel \text{Branch}_{\text{main}} \tag{4}$$

2.4 Les Trois États de Git

Git possède trois états principaux dans lesquels les fichiers peuvent se trouver :

$$\text{Working Directory} \xrightarrow{\text{git add}} \text{Staging Area} \xrightarrow{\text{git commit}} \text{Repository} \tag{5}$$

1. **Working Directory** : Répertoire de travail contenant les fichiers modifiés
2. **Staging Area** : Zone d'index où on prépare les fichiers pour le commit
3. **Repository** : Base de données Git contenant l'historique complet

3 Installation et Configuration

3.1 Installation

Windows :

```
# Telecharger depuis git-scm.com
winget install --id Git.Git -e --source winget
```

Linux (Ubuntu/Debian) :

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

MacOS :

```
brew install git
```

3.2 Configuration Initiale

```
# Configuration de l'identite
git config --global user.name "Votre Nom"
git config --global user.email "votre.email@example.com"

# Configuration de l'editeur par defaut
git config --global core.editor "code --wait"

# Verifier la configuration
git config --list
```

$$\text{Identity} = (\text{user.name}, \text{user.email}) \tag{6}$$

4 Commandes Git Essentielles

4.1 Initialisation et Clonage

```
# Initialiser un nouveau depot local
git init

# Cloner un depot existant
git clone <url-du-depot>
git clone https://github.com/username/repository.git

# Cloner une branche specifique
git clone -b <branch-name> <url-du-depot>
```

4.2 Commandes de Base

```
# Verifier le statut des fichiers
git status

# Ajouter des fichiers a la zone de staging
git add <fichier>
git add .                    # Ajouter tous les fichiers modifies
git add *.js                 # Ajouter tous les fichiers .js

# Creer un commit
git commit -m "Message de commit descriptif"
git commit -am "Message"     # Add + Commit pour fichiers suivis

# Voir l'historique des commits
git log
git log --oneline            # Format condense
git log --graph --all        # Affichage graphique
git log --author="Nom"       # Commits d'un auteur specifique
```

$$\text{Staging Area Size} = \sum_{i=1}^{n} \text{size}(\text{file}_i) \tag{7}$$

4.3 Synchronisation avec le Dépôt Distant

```
# Ajouter un depot distant
git remote add origin <url-du-depot>

# Voir les depots distants
git remote -v

# Pousser les modifications vers le depot distant
git push origin <branch-name>
git push -u origin main      # Premier push + tracking

# Recuperer les modifications depuis le depot distant
git pull origin <branch-name>
git pull                     # Pull depuis la branche trackee

# Recuperer sans fusionner
git fetch origin
```

$$\text{Remote} \xrightarrow{\text{fetch}} \text{Local} \xrightarrow{\text{merge}} \text{Working Directory} \tag{8}$$

4.4 Gestion des Modifications

```
# Voir les differences
git diff                     # Changements non stages
```

```
git diff --staged            # Changements stages
git diff <commit1> <commit2> # Entre deux commits

# Annuler des modifications
git restore <fichier>        # Annuler modifications (working dir)
git restore --staged <fichier> # Retirer du staging
git reset HEAD <fichier>     # Alternative pour unstage

# Annuler un commit
git reset --soft HEAD~1      # Garde les modifications en staging
git reset --mixed HEAD~1     # Garde les modifications (defaut)
git reset --hard HEAD~1      # SUPPRIME les modifications

# Revenir a un commit specifique
git revert <commit-hash>     # Cree un nouveau commit inverse
```

$$\text{Reset Types} = \begin{cases} \text{soft} & \text{Garde staging + working dir} \\ \text{mixed} & \text{Garde working dir} \\ \text{hard} & \text{Supprime tout} \end{cases} \tag{9}$$

5 Branches et Fusion

5.1 Gestion des Branches

```
# Lister les branches
git branch                   # Branches locales
git branch -r                # Branches distantes
git branch -a                # Toutes les branches

# Creer une nouvelle branche
git branch <nom-branche>

# Changer de branche
git checkout <nom-branche>
git switch <nom-branche>     # Nouvelle syntaxe

# Creer et changer de branche
git checkout -b <nom-branche>
git switch -c <nom-branche>

# Supprimer une branche
git branch -d <nom-branche>  # Suppression securisee
git branch -D <nom-branche>  # Suppression forcee

# Renommer une branche
git branch -m <ancien-nom> <nouveau-nom>
```

$$\text{Branches} = \{\text{main}, \text{develop}, \text{feature}_1, \ldots, \text{feature}_n\} \tag{10}$$

5.2 Fusion (Merge)

```
# Fusionner une branche dans la branche courante
git merge <nom-branche>

# Fusionner sans fast-forward (cree toujours un commit de merge)
git merge --no-ff <nom-branche>

# Abandonner une fusion en cas de conflit
git merge --abort
```

$$\text{Merge}(\text{Branch}_A, \text{Branch}_B) = \text{Branch}_{\text{merged}} \tag{11}$$

5.3 Rebase

Le rebase permet de réécrire l'historique en déplaçant ou en combinant des commits.

```
# Rebaser la branche courante sur une autre
git rebase <nom-branche>

# Rebase interactif (pour modifier l'historique)
git rebase -i HEAD~n          # n derniers commits

# Continuer/Abandonner un rebase
git rebase --continue
git rebase --abort
```

$$\text{Rebase} : \text{Branch}_{\text{feature}} \rightarrow \text{Base}_{\text{new}} + \text{Commits}_{\text{feature}} \tag{12}$$

5.4 Résolution de Conflits

```
# Voir les fichiers en conflit
git status

# Apres resolution manuelle
git add <fichier-resolu>
git commit -m "Resolution des conflits"

# Utiliser un outil de merge
git mergetool
```

6 Conclusion

Git et GitHub sont des outils essentiels pour tout développeur moderne. La maîtrise de ces outils permet :

$$\text{Productivite} \propto \text{Maitrise de Git} \times \text{Collaboration} \tag{13}$$

- ✓ Versionnement efficace du code
- ✓ Collaboration en équipe facilitée
- ✓ Historique complet et traçabilité
- ✓ Expérimentation sans risque avec les branches
- ✓ Intégration continue et déploiement automatisé

La pratique régulière est la clé de la maîtrise !

Rapport généré le : 7 janvier 2026
Version : 1.0