📘 Guide des Commandes Git pour Débutants

1️⃣ Qu'est-ce que Git ?

Git est un système de contrôle de version qui permet de :

- Suivre l'historique des modifications de votre code
- Travailler en équipe sur le même projet
- Revenir à une version antérieure si nécessaire
- Gérer différentes versions (branches) du projet

Analogie : Imaginez Git comme une machine à voyager dans le temps pour votre code !

2️⃣ Installation et Configuration

📥 Vérifier si Git est installé

```bash
git --version
```

⚙️ Configuration initiale (à faire une seule fois)

Configurer votre nom :

```bash
git config --global user.name "Votre Nom"
```

Configurer votre email :

```bash
git config --global user.email "votre.email@example.com"
```

Vérifier la configuration :

```bash
git config --list
```

3️⃣ Commandes de Base

🆕 Créer un nouveau dépôt Git

Initialiser un dépôt dans un dossier existant :

```bash
git init
```

Cloner un dépôt existant depuis GitHub/GitLab :

```bash
git clone https://github.com/utilisateur/nom-du-projet.git
```

📊 Vérifier l'état du projet

Voir les fichiers modifiés :

```bash
git status
```

Affichage :

- 🔴 Fichiers modifiés (non ajoutés)
- 🟢 Fichiers prêts à être commités
- ⚪ Fichiers non suivis

➕ Ajouter des fichiers au suivi

Ajouter un fichier spécifique :

```bash
git add nom-du-fichier.txt
```

Ajouter tous les fichiers modifiés :

```bash
```

```
git add .
```

Ajouter plusieurs fichiers :

```bash
git add fichier1.txt fichier2.js fichier3.css
```

💾 Sauvegarder les modifications (Commit)

Créer un commit avec un message :

```bash
git commit -m "Description des modifications"
```

Exemples de bons messages :

```bash
git commit -m "Ajout de la page de connexion"
git commit -m "Correction du bug d'affichage"
git commit -m "Mise à jour de la documentation"
```

Ajouter et commiter en une seule commande :

```bash
git commit -am "Message"
```

⚠️ Fonctionne uniquement pour les fichiers déjà suivis par Git.

📜 Voir l'historique des commits

Afficher tous les commits :

```bash
git log
```

Afficher l'historique en une ligne par commit :

```bash
git log --oneline
```

Afficher un graphique des branches :

```bash
git log --oneline --graph --all
```

Voir les 5 derniers commits :

```bash
git log -5
```

4️⃣ Travailler avec les Branches

🌿 Qu'est-ce qu'une branche ?

Une branche est une version parallèle de votre projet. Cela permet de :

- Développer de nouvelles fonctionnalités sans toucher au code principal
- Tester des idées sans risque
- Travailler sur plusieurs fonctionnalités en même temps

🔀 Commandes de gestion des branches

Voir toutes les branches :

```bash
git branch
```

Créer une nouvelle branche :

```bash
git branch nom-de-la-branche
```

Changer de branche :

```bash
git checkout nom-de-la-branche
```

Créer et changer de branche en une commande :

```bash
git checkout -b nouvelle-branche
```

Ou avec la nouvelle syntaxe :

```bash
git switch -c nouvelle-branche
```

Supprimer une branche :

```bash
git branch -d nom-de-la-branche
```

Supprimer une branche (forcer) :

```bash
git branch -D nom-de-la-branche
```

Renommer une branche :

```bash
git branch -m ancien-nom nouveau-nom
```

🔗 Fusionner des branches (Merge)

Fusionner une branche dans la branche actuelle :

```bash
git merge nom-de-la-branche
```

Exemple pratique :

```bash
# 1. Se placer sur la branche principale
git checkout main

# 2. Fusionner la branche de développement
git merge developpement
```

5️⃣ Synchroniser avec un Dépôt Distant (GitHub/GitLab)

🌐 Lier un dépôt distant

Ajouter un dépôt distant :

```bash
git remote add origin https://github.com/utilisateur/projet.git
```

Voir les dépôts distants :

```bash
git remote -v
```

⬆️ Envoyer des modifications (Push)

Envoyer les commits vers GitHub/GitLab :

```bash
git push origin nom-de-la-branche
```

Exemple pour la branche main :

```bash
git push origin main
```

Envoyer et créer la branche sur le dépôt distant :

```bash
```

```
git push -u origin nouvelle-branche
```

⬇️ Récupérer des modifications (Pull)

Récupérer et fusionner les modifications :

```bash
git pull origin nom-de-la-branche
```

Récupérer sans fusionner (fetch) :

```bash
git fetch origin
```

6️⃣ Annuler des Modifications

↩️ Annuler les modifications locales

Annuler les modifications d'un fichier (avant add) :

```bash
git checkout -- nom-du-fichier.txt
```

Ou avec la nouvelle syntaxe :

```bash
git restore nom-du-fichier.txt
```

Annuler tous les fichiers modifiés :

```bash
git restore .
```

🔙 Annuler un add (unstage)

Retirer un fichier de la zone de staging :

bash

```
git reset nom-du-fichier.txt
```

Ou avec la nouvelle syntaxe :

bash

```
git restore --staged nom-du-fichier.txt
```

⏪ Annuler un commit

Annuler le dernier commit (garder les modifications) :

bash

```
git reset --soft HEAD~1
```

Annuler le dernier commit (supprimer les modifications) :

bash

```
git reset --hard HEAD~1
```

Annuler les 3 derniers commits :

bash

```
git reset --soft HEAD~3
```

🔄 Revenir à un commit précis

Voir l'historique avec les identifiants :

bash

```
git log --oneline
```

Revenir à un commit spécifique :

bash

```
git reset --hard abc1234
```

7️⃣ Commandes Avancées

🏷️ Étiqueter des versions (Tags)

Créer un tag :

```bash
git tag v1.0.0
```

Créer un tag avec message :

```bash
git tag -a v1.0.0 -m "Version 1.0.0 - Première release"
```

Voir tous les tags :

```bash
git tag
```

Envoyer un tag vers le dépôt distant :

```bash
git push origin v1.0.0
```

Envoyer tous les tags :

```bash
git push origin --tags
```

📦 Sauvegarder temporairement (Stash)

Mettre de côté les modifications en cours :

```bash
```

```
git stash
```

Récupérer les modifications mises de côté :

```
bash

git stash pop
```

Voir la liste des stash :

```
bash

git stash list
```

Appliquer un stash spécifique :

```
bash

git stash apply stash@{0}
```

🔍 Recherche et Différences

Voir les différences avant commit :

```
bash

git diff
```

Voir les différences d'un fichier spécifique :

```
bash

git diff nom-du-fichier.txt
```

Voir les différences entre deux branches :

```
bash

git diff branche1 branche2
```

Rechercher dans l'historique :

🧹 Nettoyer le dépôt

Supprimer les fichiers non suivis :

```bash
git clean -n  # Prévisualiser
git clean -f  # Supprimer
```

Supprimer les fichiers et dossiers non suivis :

```bash
git clean -fd
```

8️⃣ Fichier .gitignore

🚫 Ignorer des fichiers

Créez un fichier `.gitignore` à la racine de votre projet :

```bash
```

```
# Fichiers de configuration
.env
config.json

# Dépendances
node_modules/
vendor/

# Fichiers système
.DS_Store
Thumbs.db

# Fichiers de build
dist/
build/
*.log

# Fichiers IDE
.vscode/
.idea/
```

9️⃣ Workflow Git Typique

📋 Flux de travail quotidien

bash

```
# 1. Récupérer les dernières modifications
git pull origin main

# 2. Créer une nouvelle branche pour votre fonctionnalité
git checkout -b feature/nouvelle-fonctionnalite

# 3. Faire vos modifications dans le code

# 4. Voir ce qui a changé
git status

# 5. Ajouter les fichiers modifiés
git add .

# 6. Créer un commit
git commit -m "Ajout de la nouvelle fonctionnalité"

# 7. Envoyer la branche vers GitHub
git push origin feature/nouvelle-fonctionnalite

# 8. Créer une Pull Request sur GitHub

# 9. Une fois approuvée, fusionner dans main
git checkout main
git merge feature/nouvelle-fonctionnalite

# 10. Supprimer la branche locale
git branch -d feature/nouvelle-fonctionnalite
```

🔟 Résolution de Conflits

⚠️ Quand un conflit survient ?

Lors d'un `git merge` ou `git pull`, si deux personnes ont modifié les mêmes lignes.

🛠️ Étapes pour résoudre

1. Git vous avertit du conflit :

```
bash

CONFLICT (content): Merge conflict in fichier.txt
```

2. Ouvrez le fichier, vous verrez :

```
<<<<<<< HEAD
Votre version
=======
La version de l'autre personne
>>>>>>> branche-à-fusionner
```

3. Modifiez le fichier pour garder la bonne version

4. Ajoutez le fichier résolu :

```bash
git add fichier.txt
```

5. Finalisez le merge :

```bash
git commit -m "Résolution du conflit"
```

1️⃣1️⃣ Commandes Utiles en Un Coup d'Œil

Commande	Action
`git init`	Créer un nouveau dépôt
`git clone <url>`	Cloner un dépôt distant
`git status`	Voir l'état des fichiers
`git add .`	Ajouter tous les fichiers
`git commit -m "message"`	Créer un commit
`git push origin main`	Envoyer vers GitHub
`git pull origin main`	Récupérer depuis GitHub
`git branch`	Lister les branches
`git checkout -b <nom>`	Créer et basculer vers une branche
`git merge <branche>`	Fusionner une branche
`git log --oneline`	Voir l'historique
`git diff`	Voir les modifications
`git stash`	Mettre de côté les modifications

Commande	Action
`git reset --hard`	Annuler tout

1️⃣2️⃣ Bonnes Pratiques Git

✅ À faire :

- Faire des commits réguliers et petits
- Écrire des messages de commit clairs et descriptifs
- Utiliser des branches pour chaque fonctionnalité
- Tirer (pull) avant de pousser (push)
- Vérifier avec `git status` avant chaque commit

❌ À éviter :

- Commiter des fichiers sensibles (.env, mots de passe)
- Faire des commits trop gros avec beaucoup de changements
- Travailler directement sur la branche `main`
- Oublier de faire des `git pull` régulièrement
- Utiliser `git push --force` sans précaution

1️⃣3️⃣ Conventions de Nommage

📝 Messages de commit

Format recommandé :

```
Type: Description courte

Type peut être:
- feat: nouvelle fonctionnalité
- fix: correction de bug
- docs: documentation
- style: formatage du code
- refactor: restructuration du code
- test: ajout de tests
- chore: tâches diverses
```

Exemples :

```bash
git commit -m "feat: ajout du système de connexion"
git commit -m "fix: correction du bug d'affichage sur mobile"
git commit -m "docs: mise à jour du README"
```

🌿 Noms de branches

Format recommandé :

```
type/description-courte

Exemples:
- feature/login-system
- bugfix/header-display
- hotfix/security-issue
- release/v1.2.0
```

1️⃣4️⃣ Aide et Documentation

Obtenir de l'aide sur une commande :

```bash
git help <commande>
git <commande> --help
```

Exemples :

```bash
git help commit
git merge --help
```

1️⃣5️⃣ Glossary Git

- **Repository (Dépôt)** : Dossier contenant votre projet et son historique
- **Commit** : Sauvegarde d'un état du projet
- **Branch (Branche)** : Version parallèle du projet
- **Merge (Fusion)** : Combiner deux branches
- **Clone** : Copier un dépôt distant
- **Fork** : Copier le projet de quelqu'un d'autre sur votre compte
- **Pull Request** : Demande pour fusionner votre code dans le projet principal
- **Remote** : Dépôt distant (GitHub, GitLab)
- **Origin** : Nom par défaut du dépôt distant principal
- **HEAD** : Pointeur vers le commit actuel
- **Staging Area** : Zone temporaire avant le commit

✅ Checklist pour Démarrer avec Git

- [ ] Installer Git sur votre ordinateur
- [ ] Configurer votre nom et email
- [ ] Créer un compte GitHub/GitLab
- [ ] Initialiser votre premier dépôt (`git init`)
- [ ] Faire votre premier commit
- [ ] Créer un dépôt sur GitHub
- [ ] Lier votre dépôt local au dépôt distant
- [ ] Faire votre premier `git push`
- [ ] Créer votre première branche
- [ ] Faire votre premier `git merge`

💡 **Conseil final :** Git peut sembler complexe au début, mais avec la pratique, ces commandes deviendront naturelles. Commencez par les commandes de base et progressez petit à petit !

Guide créé pour une impression facile - Format A4