

🔧 Gérer le Versioning d'une Application

PHP, Node.js, C++, C# - Outils et Bonnes Pratiques

Sommaire

- [1 Introduction au Versioning](#)
- [2 Outils de Versioning](#)
- [3 Versioning d'une Application PHP](#)
- [4 Versioning d'une Application Node.js](#)
- [5 Versioning d'une Application C++](#)
- [6 Versioning d'une Application C#](#)
- [7 Conclusion](#)
- [8 Exercices Pratiques](#)

1 Introduction au Versioning

Le **versioning** (ou **gestion de version**) est le processus de suivi et de gestion des modifications apportées à un logiciel ou à un projet. Cela permet de :

- Suivre l'**évolution** du code source.
- Revenir à une **version antérieure** en cas de problème.
- Travailler en **équipe** sans écraser le travail des autres.
- Créer des **releases** (versions stables).

 **Outil le plus populaire :** **Git** est le système de versioning le plus utilisé dans l'industrie du développement logiciel.

2 Outils de Versioning

Voici les outils les plus courants pour gérer le versioning :

Outil	Type	Description
Git	Système de versioning distribué	Le plus populaire. Fonctionne localement et peut être synchronisé avec des serveurs distants (GitHub, GitLab, Bitbucket).
SVN (Subversion)	Système de versioning centralisé	Moins populaire que Git, mais encore utilisé dans certains environnements.
Mercurial	Système de versioning distribué	Alternative à Git, mais moins répandue.

 **Recommandation :** **Git** est le standard de facto. Il est essentiel de le maîtriser.

3 Versioning d'une Application PHP

Pour une application PHP, **Git** est l'outil de choix. Voici comment procéder :

1. Initialiser un dépôt Git

```
cd /chemin/vers/votre/application_php  
git init
```

2. Créer un fichier ` .gitignore`

Ce fichier permet d'exclure certains fichiers/dossiers du versioning (ex: fichiers de log, uploads temporaires, `vendor/` de Composer).

```
# Fichiers de configuration sensibles  
config/db.php  
.env  
  
# Dossiers de cache  
cache/  
logs/  
  
# Dossiers d'uploads  
uploads/  
  
# Dépendances PHP (si gérées avec Composer)  
vendor/  
  
# Fichiers temporaires  
*.tmp  
*.log
```

3. Premier commit

```
git add .  
git commit -m "Initial commit - Structure de base de l'application PHP"
```

4. Connecter à un dépôt distant (ex: GitHub)

```
git remote add origin https://github.com/votre_compte/votre_repo_php.  
git branch -M main  
git push -u origin main
```

 **Astuce :** Si vous utilisez des dépendances PHP (ex: Composer), veillez à ne pas les pousser sur GitHub. Le fichier **composer.lock** est utile, mais pas **vendor/**.

4 Versioning d'une Application Node.js

Les applications Node.js utilisent **Git** de la même manière que PHP, mais avec quelques spécificités :

1. Initialiser un dépôt Git

```
cd /chemin/vers/votre/application_nodejs  
git init
```

2. Fichier ` .gitignore` pour Node.js

```
node_modules/  
.env  
npm-debug.log*  
coverage/  
.nyc_output/  
dist/  
*.log
```

3. Premier commit

```
git add .  
git commit -m "Initial commit - Projet Node.js avec Express"
```

4. Connecter à un dépôt distant

```
git remote add origin https://github.com/votre_compte/votre_repo_node  
git branch -M main  
git push -u origin main
```

 **Important :** **node_modules/** (dossier des dépendances) est **toujours exclu** du versioning. Les autres développeurs utilisent **npm install** pour les installer.

5 Versioning d'une Application C++

Les applications C++ utilisent également **Git** pour gérer le code source :

1. Initialiser un dépôt Git

```
cd /chemin/vers/votre/application_cpp  
git init
```

2. Fichier ` .gitignore ` pour C++

```
# Fichiers objets  
*.o  
*.obj  
# Exécutables  
*.exe  
*.out  
# Dossiers de build  
build/  
Debug/  
Release/  
# Fichiers temporaires de l'IDE  
.vs/  
.idea/  
*.swp  
*.swo
```

3. Premier commit

```
git add .  
git commit -m "Initial commit - Projet C++ avec Makefile"
```

4. Connecter à un dépôt distant

```
git remote add origin https://github.com/votre_compte/votre_repo_cpp.  
git branch -M main  
git push -u origin main
```

 **Attention :** Ne pas versionner les fichiers compilés (.exe, .o, etc.) ni

les fichiers temporaires de l'IDE.

6 Versioning d'une Application C#

Les projets C# (ex: avec .NET) utilisent **Git** avec des spécificités Microsoft :

1. Initialiser un dépôt Git

```
cd /chemin/vers/votre/application_csharp  
git init
```

2. Fichier ` .gitignore ` pour C# (.NET)

```
# Fichiers binaires et objets  
bin/  
obj/  
# Fichiers de configuration utilisateur  
*.user  
*.suo  
.vs/  
# Fichiers de build  
*.dll  
*.exe  
*.pdb  
*.nupkg  
# Dossiers de packages NuGet  
packages/  
# Fichiers temporaires  
*.tmp  
*.log
```

3. Premier commit

```
git add .  
git commit -m "Initial commit - Projet C# avec ASP.NET Core"
```

4. Connecter à un dépôt distant

```
git remote add origin https://github.com/votre_compte/votre_repo_csha  
git branch -M main  
git push -u origin main
```

 **Astuce :** Utilisez [ce modèle de .gitignore pour Visual Studio](#) pour des projets C#/.NET.

7 Conclusion

Git est l'outil universel pour gérer le versioning de vos projets, qu'ils soient écrits en **PHP, Node.js, C++, C# ou tout autre langage**.

- Il est **indispensable** pour tout développeur moderne.
- Il permet de **collaborer efficacement** et de **maintenir un historique** de votre code.
- Chaque langage ou framework a ses **fichiers à ignorer** via **.gitignore**.
- Des plateformes comme **GitHub, GitLab, Bitbucket** permettent de **héberger** vos dépôts.

8 Exercices Pratiques

Exercice 1 : Initialiser un dépôt Git pour votre projet PHP

Allez dans le dossier de votre application PHP, initialisez un dépôt Git, créez un fichier ` `.gitignore` , et faites votre premier commit.

Exercice 2 : Créer un dépôt distant sur GitHub

Sur github.com, créez un nouveau dépôt vide et liez-y votre projet local avec **git remote add** et **git push**.

Exercice 3 : Modifier un fichier et faire un nouveau commit

Changez un fichier de votre projet, ajoutez-le à la zone de staging avec **git add**, puis commitez avec **git commit**.

Exercice 4 : Utiliser des branches

Créez une nouvelle branche nommée **develop**, basculez-y, faites-y une modification, puis fusionnez-la avec la branche **main**.

```
git checkout -b develop
# (Faire une modification)
git add .
git commit -m "Nouvelle fonctionnalité"
git checkout main
git merge develop
```

Exercice 5 : Générer un fichier ` .gitignore ` pour votre projet Node.js

Créez un fichier ` .gitignore ` approprié pour votre projet Node.js en utilisant les règles vues dans ce tutoriel.