

II. DÉPLOIEMENT GITHUB

1. ABSTRACT

Tout projet professionnel de génie logiciel implique plus d'un intervenant. Différents acteurs aux coeurs de métier différents interviennent selon des périmètres variés au sein du projet dépendamment de leur rôle, grade et attentes en termes de valeur ajoutée.

Qu'il s'agisse de produire un système, soit un logiciel qui s'adresse à d'autres logiciels ou un applicatif, dont l'utilisateur final est un opérateur humain, la complexité est telle qu'il est nécessaire de faire appel à des outils de travail collaboratif afin d'absorber cette complexité technique et humaine et garantir les chances de succès du sus-dit projet.

Le déploiement en environnement de productif n'est qu'un symptôme; le critère du succès est la valeur utile rendue accessible aux professionnels métier l'exploitant dans le cadre de leur opérations quotidiennes afin de remplir leur rôle en société.

Le cycle de vie normal du projet requiert l'élaboration de phases en amont du projet avant sa mise en production, savoir:

- Les spécifications techniques
- Analyse
- Conception
- Implementation
- Tests
- Documentation
- Déploiement

Après son déploiement en environnement de production livré chez client, ou rendu accessible à distance pour lui et ses collaborateurs, le projet logiciel entre alors en phase de maintenance évolutive.

Les besoins métiers du client peuvent et changeront sans doute, et le logiciel subit régulièrement des nouvelles itérations logicielles incluant une nouvelle fois les étapes susmentionnées.

Un logiciel est donc un projet vivant qui regroupe de nombreux protagonistes aux jeux de compétences, objectifs et besoins très variés. C'est pourquoi il est nécessaire de rendre le code source du logiciel disponible auprès de différentes équipes travaillant en parallèle, qu'il s'agisse d'un même site physique ou d'équipes délocalisées sur plusieurs continents et oeuvrant sur des fuseaux horaires variés.

2. RATIONALE

A ce motif, des outils de gestion de projet collaboratif et de contrôle de version ont vu le jour au début du XXI^e siècle afin de palier aux besoins des programmeurs pour opérer concomitamment sur un projet en constante évolution. Le logiciel est un flux interrompu de mises à jour et de corrections qui impose un cadre strict de gestion des accès et de politique de modification et d'historisation, afin que chacun puisse contrôler les choix d'implémentation technique ainsi que la direction générale vers laquelle s'oriente le projet.

Après des logiciels comme BitBucket (toujours actif sur le marché), Mercurial et Subversion (qui sont tous des deux des outils de gestion de version centralisés, avec les

problèmes que cela implique; un homme du nom de Linus Torvalds, auteur du noyau GNU/Linux, lui-même basé sur Minix, fruit des efforts de Andrew S. Tanenbaum, publie en ligne une première version à source ouverte et gratuite d'un nouvel entrant sur le marché en 2005: git.

Git est un système de version de contrôle distribué permettant d'assurer l'intégrité d'échanges de données et flux de travaux non-linéaires.

Il est accessible en ligne de commande et s'avère très utile pour normaliser le développement du projet logiciel, y compris pour des projets qui seraient menés par une seule personne.

3. ADOPTION DE GIT

Git est adopté par l'équipe en raison de son immense popularité, tant à la fois pour les projets open source individuels que professionnels, et au sein de l'industrie civile et de la sphère académique.

Ses barrières à l'entrée sont faibles:

- ses documentations gratuites et payantes sont nombreuses,
- il existe une très grande communauté active de programmeurs du monde entier, qui s'avère disponible pour renseigner et aider 24-7 sur les canaux IRC et discord.com.
- son utilisation est gratuite (financièrement).

Git est un outil qui incarne le socle fondamental de nombreux acteurs sur le marché de la gestion de projets logiciels communautaires.


4. CRÉATION D'UN DÉPÔT

Un espace de travail local (sur la machine physique du programmeur), s'appelle un dépôt. Chaque dépôt correspond donc à un système à développer et maintenir, ou même à un sous-système selon la complexité du projet.

1. Création du dossier de travail

```
~/dev/SII  
→ mkdir TraceMyPlane; cd TraceMyPlane|
```

2. Initialisation du dépôt git pour démarrer le pistage des fichiers

```
dev/SII/TraceMyPlane  
→ git init  
Initialized empty Git repository in /home/aurele/dev/SII/TraceMyPlane/.git/  
  
TraceMyPlane on  master  
→ |
```

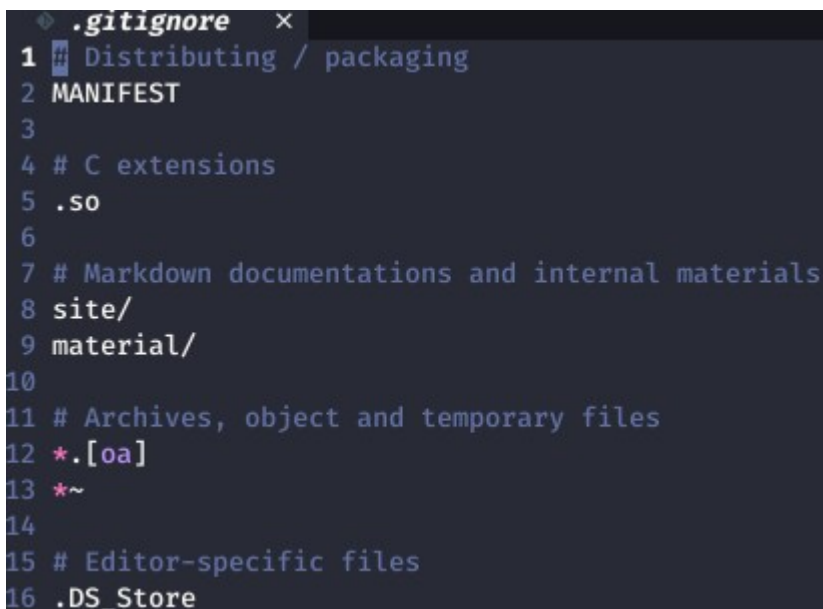
3. Préparation du projet avec création des fichiers administratifs et logistiques:

- **LICENSE**: la license retenue est ISC. Originellement utilisée par l'Université de Berkeley en California, elle fût depuis adoptée par le projet de système d'exploitation OpenBSD. Elle est approuvée par la fondation Open Source Initiative, consortium qui supervise, promeut et règlemente les licences open sources à l'échelle internationale.


La license ISC est recommandée par les mainteneurs du système d'exploitation FreeBSD pour les nouveaux projets et protège la propriété intellectuelle des entrepreneurs de la vampirisation de la base de code face aux très grandes entreprises et conglomérats qui voudraient absorber la substance du projet pour l'intégrer à ses solutions payantes et propriétaires.

- **README.md** : le fichier "Lisez-moi" au format Markdown. Il présente le projet et récapitule son contenu, son rôle et ses aspirations. Devenu au fil du temps plus publicitaire que technique, il permet de mettre en avant les avantages d'un projet et de susciter l'intérêt dans l'esprit des visiteurs désireux d'exploiter une nouvelle solution logicielle qui répond mieux à leurs besoins actuels, ou même de nouveaux contributeurs en quête de défis techniques pour améliorer leurs compétences en programmation et/ou gestion de projet (selon les flux de travail autorisés par le créateur).
- **.gitignore**: le fichier caché ".gitignore" permet de lister des fichiers, répertoires et ou une nomenclature des deux précédents devant être ignorés par la fonctionnalité de pistage de git. Cela permet d'éviter de téléverser au dépôt distant (hébergé en-ligne) des fichiers inutiles aux utilisateurs du projet.


Il peut s'agir de dossiers de tests internes, de fichiers temporaires, de recherches documentaires utiles seulement aux contributeurs dans le cadre du développement du projets ou de bibliothèques partagées.

A screenshot of a code editor showing the content of a .gitignore file. The file is titled ".gitignore" with a close button (X) in the top right corner. The content is as follows:


```
1 Distributing / packaging
2 MANIFEST
3
4 # C extensions
5 .so
6
7 # Markdown documentations and internal materials
8 site/
9 material/
10
11 # Archives, object and temporary files
12 *.[oa]
13 *~
14
15 # Editor-specific files
16 .DS_Store
```

```
TraceMyPlane on  master [?]  
→ git status -s  
?? .gitignore  
?? LICENSE  
?? README.md
```

5. Ajout des fichiers à pister par git

```
TraceMyPlane on  master [?]  
→ pwd  
/home/aurele/dev/SII/TraceMyPlane
```

6. Soumission du code dans le dépôt local

```
TraceMyPlane on  master [+]  
→ git commit -m "Adding LICENSE, README.md and .gitignore files"  
[master (root-commit) d4df0b6] Adding LICENSE, README.md and .gitignore files  
3 files changed, 52 insertions(+)  
create mode 100644 .gitignore  
create mode 100644 LICENSE  
create mode 100644 README.md
```

```
git clone git@github.com:WendyLaSirene/TraceMyPlane_Common --depth=1  
tracemyplane
```