



SOUTENANCE POEI INGÉNIEUR LOGICIELS C++ :

Projet de GPS *TraceMyPlane*

IDENTIFIANTS DU DOCUMENT

Auteur	Aurélien PLAZZOTTA
Date de création	2023-01-12
Dernière mise à jour	2022-01-16
Status	Travail en cours
Version	0.0.1
License	ISC



II. DÉPLOIEMENT VIA GIT ET 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.

3.1 NOTATION BACKUS-NAUR

Un metalanguage a été défini par le mathématicien nord-américain du XX^e siècle Backus et le mathématicien perse du XI^e siècle Naur. Il permet la communication d'expressions logico-mathématiques couramment utilisées et est adoptée dans le présent document afin de faciliter la transmission d'idées riches avec une densité d'informations plus élevée.



En voici les cinq éléments fondamentaux:

Symbol	Definition
::=	Défini comme suit
	OU logique
< >	concept (<i>i.d. nom d'objet, valeur</i>)
[]	possible
{ }	<i>item</i> à définir

3.2 CONFIGURATION

Il existe trois options de configuration impactant le périmètre de git sur la machine:

1. **--system** : modifie la configuration pour tous les utilisateurs ayant un compte sur la machine physique
2. **--global** : le paramètre est modifié pour le compte utilisateur actuel
3. **--local** : seul le dépôt actuel bénéficie de cette modification

Les priorités sont définies du plus restreint au plus général (e.g. **--local** supplante **--system**).

Nota Bene: les exemples de commandes qui suivent utilisent l'option "**--global**" à titre purement indicatif, libre à vous d'utiliser **--system** ou **--local** selon vos besoins.

Les 3 options pouvant être cumulées sur une même machine, au travers de multiples dépôts.

Dans le but de préparer l'espace de travail à la connexion et l'alimentation d'un dépôt distant (*i.e.* "remote") et ainsi partager les nouvelles itérations du code source à toutes les personnes impliquées dans le développement.

Il est requis de renseigner au minimum un nom et une adresse électronique pour identifier son compte utilisateur.

1.

Saisir et valider avec la touche [Return]:

```
git config --global user.name <user_name>
```

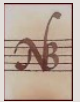
```
git config --global user.email <email>
```

```
→ git config --global user.name aurèle
```

```
~  
→ git config --global user.email aurelien.plazzotta@tutanota.com
```



Nota bene:



D'autres options peuvent être personnalisées:

- un éditeur de texte: par défaut, lancé automatiquement lorsque vous omettez le paramètre `-m <message>` lors de l'exécution d'une commande `commit`.
- Un paginateur peut être défini pour remplacer les commandes `less` et `more`.
- Un nom de branche par défaut pour vos futurs créations de dépôts.
- Des alias pour faciliter la saisie de commandes couramment employées
- Une signature électronique (pour utiliser implicitement l'option `-S` de la commande `commit`)

2.

3.1.3 Pour afficher la liste de vos paramètres personnalisés, saisir:

```
git config --list
```

```
→ git config --list
user.name=aurele
user.email=aurelien.plazzotta@tutanota.com
user.signingkey=ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AA
init.defaultbranch=master
alias.last=log -1 HEAD
alias.st=status -s
core.editor=nvim
core.pager=nvimpager
color.pager=true
commit.gpgsign=false
gpg.format=ssh
pull.ff=only
push.autosetupremote=true
push.default=simple
```

Nota bene 2:

Pour créer un alias, et faciliter l'emploi d'une commande régulièrement usitée:

```
git config --global alias.<alias_name> <command>
e.g. git config --global alias.st "status -s"
```

Nota bene 2:

L'option `--show-origin` vous permet d'afficher en sus, le chemin d'accès de votre fichier de configuration globale `git`.



```
→ git config --list --show-origin
```



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 en local

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

2.

Initialisation du dépôt git pour démarrer le pistage des fichiers. Cette commande crée un sous-dossier .git contenant les fichiers de fonctionnement interne à git lui-même.

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

3.

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 réglemente 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 cherchant 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 finaux du projet (e.g. dossiers de tests internes, fichiers temporaires, recherches documentaires utiles seulement aux contributeurs dans le cadre du développement du projet ou de bibliothèques partagées).

```
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
```

4.

Contrôler l’état actuel du dossier de travail:

```
git status [-s]
```

```
myplane on ↗ master [!X!+?]
→ git status -s
M Classe_Aeroport.h
D Fonctions_Diverses.h
A testing.cpp
A tools.cpp
?? Fonctions_Diverses.h
```

Nota bene:

L’option --s (pour “short”) permet d’augmenter la densité de l’information afin de réduire la pollution informationnelle.

Lexikon:

- Rouge : action utilisateur requise	- Vert : nouvel état valide	- A : ajouté
	- ?? : non pisté	- M : modifié

5.

Ajouter des fichiers à pister par git:

```
git add <file(s)>
```

```
TraceMyPlane on ↗ master
→ git add {LICENSE,README.md,.gitignore}
```



6.

Soumettre du code dans le dépôt local:

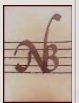
```
TraceMyPlane on master [+]
→ git commit -m "Adding LICENSE, README.md and .gitignore files"
[master (root-commit) d4df0b6] Adding LICENSE, README.md and .gitignore files
3 files changed, 52 insertions(+)
create mode 100644 .gitignore
create mode 100644 LICENSE
create mode 100644 README.md
```

```
git commit -m <message>
```

Nota bene:

L'option `--a` (pour "all") permet d'outrepasser la zone tampon (étape intermédiaire "stage").

Tous les nouveaux fichiers sont automatiquement ajoutés puis soumis ("commit"). La perte d'un filet de sûreté s'accompagne d'un gain de temps et évite les omissions.



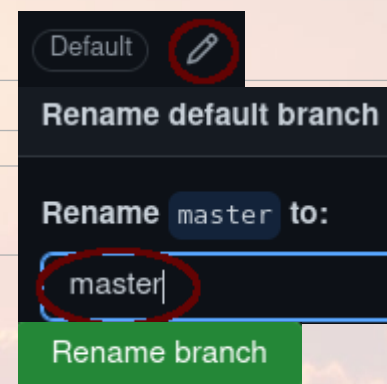
[RENOMMER UNE BRANCHE PAR DÉFAUT]

1. Rendez-vous sur la page suivante: (pensez à modifier le nom d'utilisateur), dans l'onglet "Overview".

<https://github.com/kenaryn/TraceMyPlane/branches> Overview

2. Cliquez sur l'icône en forme de stylo.

3. Saisir le nouveau nom de branche par défaut puis cliquez sur le bouton "Rename branch" pour confirmer l'action.



4. Actualisez la nouvelle branche du dépôt distant vers votre espace de travail local:

Commande	Rôle
<code>git branch -M <branch></code>	Créer une nouvelle branche, écraser le nom de l'actuelle et basculer dessus.
<code>git fetch origin</code>	Configure le flux pour rappatrier les données localement



	depuis le dépôt distant "origin"
<code>git branch -u <upstream>/<remote_branch> <local_branch></code>	Mettre à jour le pistage de référence de la branche distante master du dépôt "origin" depuis la branche locale locale "master"
<code>git remote set-head origin -a</code>	Actualiser le pointage du curseur vers le nouveau nom

```
→ git branch -M master
→ git fetch origin
→ git branch -u origin/master master
→ git remote set-head-origin -a
```

5. CHOISIR SA SOLUTION POUR PARTAGER SES TRAVAUX VERS UN DÉPÔT DISTANT

Partager ses travaux vers un dépôt public ou privé pour rendre disponible l'instant des fichiers d'en-tête, les fichiers source, les images et les documentations liées à la compréhension du projet requiert l'hébergement de ces données vers un serveur web disponible en permanence.

La location des serveurs leur administration peut être à la fois coûteux en temps et en argent. C'est pourquoi une solution gratuite d'hébergement dédiée à la gestion décentralisée des flux de travaux collaboratifs s'est imposée à l'équipe.

De plus, les différents acteurs du marché intègrent tous une interface graphique utilisateur pour aider à l'usage des solutions d'hébergement pour les utilisateurs les moins avertis, et ainsi faciliter son adoption auprès du plus grand nombre, renforçant par la même occasion la continuité d'activité du service, et *in fine*, la base d'utilisateurs actifs à même d'en assurer la promotion et pourquoi des documentations liées à la plate-forme.

Le marché présente de nombreuses solutions directement concurrentes:

Solution	Site web officiel	Particularités
Gitea	gitea.io/en-us	Léger, facile à prendre en main
Fossil	fossil-scm.org/	Executable portable, stocke ses fichiers sous SQLite, préserve le véritable historique
Gogs	gogs.io	S'exécute sur tout environnement supportant le Go, faible consommation de ressources matérielles
* Github	github.com	Gigantesque base d'utilisateurs
Gitlab	gitlab.com	Intègre nativement des outils de développement/integration continus, métriques



		avancés
Bitbucket	bitbucket.org	Premier acteur historique, nombreuses offres pro, intègre Jira
Sourcehut	sr.ht	Granularité fine des accès utilisateurs et des dépôts
Gitly	gitly.org	Ecrit en V, expérimental, extrêmement léger, fonctionne sans JS, rapide

Le choix de l'équipe s'est porté sur **github** en raison de la connaissance préalable du service de certains membres de l'équipe.


6. TÉLÉVERSEMENT DES TRAVAUX VERS UN DÉPÔT DISTANT

Deux scénarii se présentent à chaque nouveau membre d'une équipe de développement:



1. A

Le dépôt n'existe pas. Créer d'un nouveau dépôt vierge distant ("remote", sur un réseau en-ligne) *via* l'interface graphique de l'hébergeur sur la page des dépôts du compte **github.com**

 <https://github.com/kenaryn?tab=repositories>

2.

Cliquez sur le bouton vert "New"

 New



3.

1.3 Définir un nom de dépôt et valider

Repository name *

FollowMyLead|



Create repository

1. B

Le dépôt existe déjà. Télécharger son contenu sur sa machine locale *via* la commande git clone:

**git clone git@github.com:<user>/<repo_name> [--depth=n]
[<custom_repository_name>]**

```
~/dev/SII  
→ git clone git@github.com:WendyLaSirene/TraceMyPlane_Common.git sii_project  
Cloning into 'sii_project'...  
Enter passphrase for key '/home/aurele/.ssh/id_ed25519':  
remote: Enumerating objects: 51, done.  
remote: Counting objects: 100% (51/51), done.  
remote: Compressing objects: 100% (31/31), done.  
remote: Total 51 (delta 19), reused 51 (delta 19), pack-reused 0  
Receiving objects: 100% (51/51), 72.39 KiB | 692.00 KiB/s, done.  
Resolving deltas: 100% (19/19), done.
```

Nota bene:

l'option **--depth** permet de cloner le dépôt en tronquant tous les commit à l'exception des "n" derniers commit définis en argument de la commande de la forme **--depth=n** (où "n" est un nombre naturel).

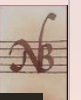


5.

1.4 Enregistrer la nouvelle adresse distante dans votre espace de travail pour pouvoir synchroniser localement:

git remote add git@github.com:<user>/<repository.git>

```
TraceMyPlane on ↵ master [??]  
→ git remote add origin git@github.com:kenaryn/TraceMyPlane.git|
```



6.

Si au coeur du cycle de vie applicatif, le dépôt distant vient à changer de nom et/ou d'adresse, votre espace de travail doit pouvoir interroger la nouvelle adresse pour y accéder:

git remote set-url --add git@github.com:<user>/<new_url>

```
TraceMyPlane on ↵ master [??]  
→ git remote set-url --add git@github.com:wendy/TraceMyPlane.git|
```




7.

Téléverser la dernière itération des travaux locaux vers le dépôt github.com



Nota bene:

git compute le *delta* (différence entre l'existant en local et la version des travaux en-ligne) et téléverse ("upload") les fichiers objets mis à jour.

Un nouveau commit avec son identification par hashage est généré et ajouté à la branche pointée par le curseur.

```
TraceMyPlane on ↩ master [↑]
→ git push -u origin master
Enter passphrase for key '/home/aurele/.ssh/id_ed25519':
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 270.43 KiB | 2.73 MiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:kenaryn/TraceMyPlane.git
    7c5d777..30ff4f3  master -> master
branch 'master' set up to track 'origin/master'.
```



Nota bene:

L'identification via une phrase de passe (plus complexe à déchiffrer qu'un mot de passe vous êtes demandé si vous avez configuré une clé publique associée à une clé privée).

Plus d'informations ici: <https://docs.github.com/en/authentication/connecting-to-github-with-ssh/working-with-ssh-key-passphrases>



3.2 METTRE À JOUR LA BRANCHE DE DEVELOPPEMENT PRINCIPALE

Un membre de l'équipe a intégré dans son espace de travail une nouvelle itération du code intégrant l'ajout ou la modification d'une fonctionnalité.

Ce nouveau code est révisé soit lors d'une session de programmation en binôme, soit leur d'une révision technique par un pair. Une fois validé, le gestionnaire du dépôt doit fusionner la branche dédiée à cette fonctionnalité ("feature") avec les travaux principaux (la branche "master").

1.

S'assurer de rattrier en local la dernière version du dépôt distant:

```
git fetch  
origin
```

```
TraceMyPlane on ↗ master [!] took 7s  
→ git fetch origin  
Enter passphrase for key '/home/aurele/.ssh/id_ed25519':
```

2.

Fusionner les modifications de la branche <feature> pour mettre à jour la branche maîtresse

```
git merge <feature_branch>
```

```
myplane on ↗ master  
→ git merge deployment  
Updating fb72003..ceb49b8  
Fast-forward  
CMakeLists.txt | 17 ++++++  
1 file changed, 17 insertions(+)  
create mode 100644 CMakeLists.txt
```

Nota bene:

Une autre commande fusionne les modifications à l'instar de merge:

```
git rebase <base_branch> <topic_branch>
```

Toutefois, un unique parent est ajouté au journal de commits, comme si la mise à jour est effectuée de manière séquentielle au lieu de présenter 2 branches traitées de manière parallèle.



[RENOMMER UNE BRANCHE PAR DÉFAUT]

Lorsque l'espace de travail local est désynchronisé avec le dépôt distant, toute fusion devient impossible. git tente ainsi d'éviter les pertes irrécouvrables de données: Tenter de fusionner les modifications échoue.

```
TraceMyPlane on ↗ master [!]  
→ git push -u origin master  
Enter passphrase for key '/home/aurele/.ssh/id_ed25519':  
To github.com:kenaryn/TraceMyPlane.git  
! [rejected] master -> master (non-fast-forward)  
error: failed to push some refs to 'github.com:kenaryn/TraceMyPlane.git'  
hint: Updates were rejected because the tip of your current branch is behind  
hint: its remote counterpart. Integrate the remote changes (e.g.  
hint: 'git pull ...') before pushing again.
```




Afficher le statut présente un diagnostic de la situation suivi d'un moyen de résoudre le conflit.

```
TraceMyPlane on ↵ (git)-[master|merge]- [!]=
→ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)

Unmerged paths:
(use "git add <file>..." to mark resolution)
    both modified:    design/github_deployment.pdf
```

Pour résoudre le conflit, taper:

git add <file>

git commit -m <message>

Afficher la liste des dernières soumissions de code selon des critères de date
git log --oneline --since 1months --until 3days --graph

```
TraceMyPlane on ↵ master [!]=
→ git log --oneline --since 5days --until now --graph
* 45199f0 (HEAD -> master) Creating Classe_Aeroport.cpp and Coeur_Vol.cpp's first versions
* 595c56b (origin/deployment, deployment) Adding CMakeLists.txt
* 26f5b25 (origin/master, origin/HEAD) Merge remote-tracking branch 'refs/remotes/origin/master'
| \
| * 30ff4f3 Upgrading git/github deployment design document to v0.0.2
* | 1a15188 Upgrading git/github deployment design document to v0.0.2
| /
* 7c5d777 Adding 'git deployment's design document first draft
* 9473851 Adding deployment via git's design document first draft.
* 472ff25 Removing deployment via git.odt
* 8793dc3 Adding deployment via git's design document first draft.
* 4f1dd2d Upgrading use case nominal scenariii to v0.0.2
* b5e29d6 Altering primary use case in the Nominal Use case diagram
```

UTILISATION DE CMAKE POUR NORMALISER COMPILATION ET DEPLOIEMENT

<capture d'écran>

brève documentation sur son usage.



VISION TECHNIQUE

utilisation de notion.so
roadmap

SPECIFICATIONS TECHNIQUES

AJOUTER DIAGRAMMES SysML:

- cas d'utilisation
- définition de bloc
- séquence