# TP sur les logiciels de gestion de versions

Nicolas Fécamp & Adrien Bullich
septembre 2021

## Résumé

L'objectif est d'apprendre à manipuler les commandes classiques de Git et de Gitlab.
Comme exercice de support nous allons coder en langage C les différentes opérations sur les complexes par équipe de 2 ou 3 personnes

## Introduction

Le but du TP est d'explorer les divers cas d'utilisation de Git. Pour cela, nous allons coder, dans le langage C, diverses opérations sur les complexes.
Les nombres complexes furent introduits au XVIe siècle pour définir des solutions à toutes les équations polynomiales à coefficients réels. Ainsi le nombre i, imaginaire, est la solution à l'équation $x^2 = -1$.
Les nombres complexes peuvent s'écrire sous la forme : $a + b * i$, avec a et b des réels.
Pour la suite on notera :

- $z_1 = a_1 + b_1 * i$
- $z_2 = a_2 + b_2 * i$

## Préliminaires

Avant de commencer le travail, il faut installer sur l'ordinateur le paquet correspondant à git si il n'est pas déjà présent :

*sudo apt-get install git*

On pourra utiliser l'éditeur de texte nano en console, ou gedit en interface graphique. L'objectif du TP se limitant à la manipulation de Git, il ne faudra pas hésiter à faire appel au professeur en cas de difficultés sur Linux ou sur le langage C.

## Récupération du dépôt - Prise en main

A partir de maintenant, la classe s'organise en groupes de 2 ou 3, mais chacun sur un ordinateur travaillant en parallèle. Le logiciel de gestion de versions Git et GitLab sera utilisé pour gérer la collaboration.

1. Création d'un projet GitLab par équipe. Constitution de l'équipe avec les bons droits sous GitLab
2. Récupérer le fichier main.c présent sur le projet public de GitLab. Il contient deux programmes : une fonction d'acquisition et une fonction d'édition
3. Sous Git, Configurer les options Git initiales nécessaires. On y ajoutera la commande *git config --global color.ui true*
   pour activer les couleurs dans Git
4. Sous GitLab, taguer le commit en lui donnant pour nom v0
5. Sous Git, Configurer un hook de pre-commit, pour s'assurer que toute version compile. Créer un fichier pre-commit sous .git/hooks avec :

*#!/bin/sh*
*gcc -o main.o main.c –lm*

6. Programmer simplement la fonction d'addition de deux complexes dans le fichier main.c.
   *void additionComplexe(float a1, float b1, float a2, float b2, float*a3, float *b3)*
   Rappel : $z_1 + z_2 = (a_1 + a_2) + (b_1 + b_2) * i$
7. Créer un nouveau commit pour enregistrer vos modifications. (Pour cela, ajouter d'abord le fichier dans l'index.)
8. Taguer le commit en lui donnant pour nom v1.
9. Pour visualiser l'avancée utilisation des commandes :
   *git status*
   *git diff [--cached]*
   *git log --pretty=format:"%h %d : %s" –graph*
   On enregistre un alias sur cette dernière commande pour la réutiliser facilement :
   *git config alias.logg "log --pretty=format:\"%h %d : %s\" --graph"*

## Utilisation des branches

Chaque équipe devra développer 3 fonctionnalités (soit une par personne), en mode collaboratif, le plus simplement possible :

- la multiplication de deux nombres complexes

*void multiplicationComplexe(float a1, float b1, float a2, float b2, float *a3, float *b3)*
Rappel : $z_1 * z_2 = (a_1 * a_2 - b_1 * b_2) + (a_1 * b_2 + a_2 * b_1) * i$

- le module complexe

*float moduleComplexe(float a, float b)*
Rappel : $|z_1| = \sqrt{a_1^2 + b_1^2}$

- l'inverse d'un nombre complexe.

void inverseComplexe(float a1, float b1, float *a2, float *b2)
Rappel : $\frac{1}{z_1} = \frac{a_1}{a_1^2+b_1^2} - \frac{b_1}{a_1^2+b_1^2} * i$

1. Chaque équipe devra élaborer sa stratégie de gestion des branches pour être capable de livrer deux versions stables :
   v2 : contenant v1 + fonctionnalité module
   v3 : contenant l'intégralité des fonctionnalités
   et ceci quel que soit l'ordre de développement des fonctionnalités.
2. Créer les branches nécessaires pour vos développements.
3. Appeler le professeur pour valider la création de vos branches.
4. Développer vos fonctionnalités. Commiter régulièrement les modifications.
5. Appeler le professeur pour valider vos développements
6. Faire les merge nécessaires en suivant votre stratégie de gestion de branches pour créer la version v2.
7. Appeler le professeur pour valider cette version.
8. Faire les merge nécessaires en suivant votre stratégie de gestion de branches pour créer la version v3.
9. Appeler le professeur pour valider cette version.

## Exploration de l’arborescence

La version initiale v1 ne gère que les entiers lors de la saisie utilisateur. Nous allons revenir à cette version et gérer les réels dans une nouvelle branche.

1. Revenir à la version v1
2. Se positionner dans une nouvelle branche nommée gestionReels
3. Modifier la version pour gérer les réels
4. Faire les merges ainsi que les adaptations nécessaires pour créer une version v4 ayant tout le contenu fonctionnel de la v3 avec la gestion des réels.