Configuration de GIT

Objectifs R2-03 : Apprendre à utiliser un outil de gestion de versions

« Git est un logiciel de gestion de versions décentralisé permettant de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. »

Dans cet exercice, nous utiliserons l'IDE IntelliJ pour réaliser la gestion de versions avec GIT. **Toutes les commandes présentées dans ce document ont leur équivalent en ligne de commande.**

Exercice 1 : créer et utiliser un dépôt local

1. Créer le dépôt local

Créer le dépôt local GIT : **`VCS > Create GIT > Create GIT repository`** … dans le dossier de votre projet.

Un dossier **`.git`** a été créé à la racine de votre projet. Il contiendra la description de votre historique de versionning (versions, branches, tag, etc.). Attention, le dossier **`.git`** n'est pas visible dans l'IDE, par contre il est visible dans votre gestionnaire de fichier.

Créer un fichier **`.gitignore`** à la racine de votre dossier. L'IDE vous demande d'ajouter votre fichier à votre dépôt, répondez oui.

Un fichier **`.gitignore`** est un fichier texte placé dans votre référentiel git qui indique à git de ne pas suivre certains fichiers et dossiers que vous ne souhaitez pas télécharger dans votre dépôt.

Remplir votre fichier **`.gitignore`**. Pour se faire, utilisez le lien « gitignore.io » qui vous redirige vers https://www.toptal.com/developers/gitignore. La page web proposée permet de créer les exclusions de votre suivi GIT. Dans notre cas, donnez les mots clés « intellij » et « java », puis **Copier-coller le code obtenu** dans votre fichier **`.gitignore`**.

Extrait des exclusions du `.gitingore` pour les mots clés intellij et java

```
# Created by https://www.toptal.com/developers/gitignore/api/intellij,java
# Edit at
https://www.toptal.com/developers/gitignore?templates=intellij,java

### Intellij ###
# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm,
CLion, Android Studio, WebStorm and Rider
# Reference: https://intellij-support.jetbrains.com/hc/en-
us/articles/206544839

# User-specific stuff
.idea/**/workspace.xml
.idea/**/tasks.xml
.idea/**/usage.statistics.xml
.idea/**/dictionaries
.idea/**/shelf

# AWS User-specific
.idea/**/aws.xml
…
```

2. Premier commit

C'est quoi un commit ? Un commit capture l'état d'un projet à un point dans le temps. Il regroupe les dernières modifications avec un message descriptif.

Pour réaliser votre premier commit - votre première version : **`GIT > commit`**

AIDE : Vous pouvez faire un commit même si votre projet a des warnings. Évitez de le faire avec des erreurs.

L'IDE ouvre une interface comprenant entre autres :

- Les fichiers que vous souhaitez ajouter à votre dépôt,
- Le message de commit : **toujours mettre un message descriptif clair !**
- Les différences avec les versions précédentes.

Concernant les fichiers à ajouter, sélectionnez-les tous pour le premier dépôt. Pour les futurs dépôts, vous ferez attention aux fichiers que vous souhaitez ajouter à vos commits. Rappel, le fichier **`.gitignore`** a permis d'éviter de « commiter » des fichiers non essentiels (comme des librairies qui peuvent être facilement re-téléchargeable).

Vous venez de réaliser votre premier commit !

3. Futur commit

Quand faire un commit ? à la fin d'une réalisation fonctionnelle ! Par exemple, dans notre cas à la fin de la réalisation d'un exercice. Ou, dans le cadre d'un projet, à la fin de la réalisation d'une fonctionnalité (même petite), testée et vérifiée !

Pourquoi mettre un message de commit clair ? pour comprendre votre progression dans la création de votre historique de version : un commit = une version.

J'ai fait une erreur, je veux effacer ma version ? NON dans la plupart des cas vous devez simplement corriger votre erreur et faire un nouveau commit avec un message de commit expliquant l'erreur et la correction.

4. L'historique

Pour voir votre historique, **`Git > Show Git Log`**

Vous verrez les différentes versions décrites par leur message de commit et les fichiers ajoutés et modifiés.

Exercice 2 : exporter ses modifications sur un dépôt distant

Pour le moment, votre dépôt est créé en local sur votre machine (dans le dossier **`.git`**). Nous souhaitons qu'il soit conservé à distance pour deux raisons : sauvegarde de votre travail et accès pour d'autres utilisateurs (dans votre cas vos enseignants). Bien entendu, un dépôt distant servira également à travailler avec d'autres personnes, vous aurez un TP plus détaillé à ce sujet dans la ressource R2.01b.

Vous utiliserez le gestionnaire de version de l'université https://gricad-gitlab.univ-grenoble-alpes.fr/ (LDAP UGA, login et mot de passe UGA). Ce site permettra d'héberger vos dépôts GIT dans le cadre de notre ressource.

1. Créer un projet Gricad et son dépôt associé

Sur Gricad, le responsable de la ressource R2.01a a créé un groupe dont vous êtes le propriétaire :

https://gricad-gitlab.univ-grenoble-alpes.fr/iut2-info-stud/2023-s2/r2.01.03/VOTRE_DEMI_GROUPE/VOTRE_LOGIN

Modifier dans l'url ci-dessus le nom de votre demi-groupe (a1, a2, b1, etc.) et votre login, pusi charger cette URL dans votre navigateur.

Créer un nouveau projet dans votre groupe : **`New project > Create blanck project`**

Donner un nom de projet clair, par exemple **`TPs-R2.01a-VOTRE_NOM-VOTRE_PRENOM`**

Vous venez de créer votre projet. La page de votre projet permet de voir le dépôt (Repository), la liste des problèmes/demandes (issues), etc.

2. Publier de votre historique : push

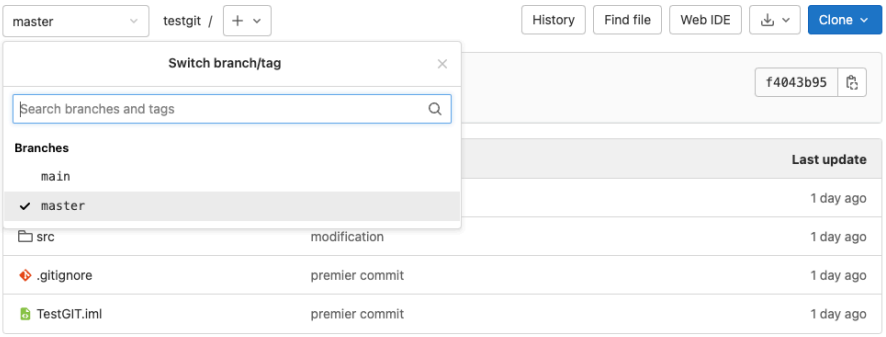
C'est quoi un push ? Cette commande permet de publier les changements locaux sur le dépôt distant.

De retour sur votre IDE, publier votre historique : **`Git > push`**

Pour le premier push, l'IDE vous demande de configurer votre « remote » **`master → define remote`** (votre dépôt distant). Pour récupérer l'adresse du remote (**NON** ce n'est pas l'URL de votre projet), retournez sur votre projet sur Gricad sur la partie repository et copier l'url présent dans **`Clone > Clone with HTTPS`**

Vous venez de faire votre premier push. Vérifiez sur Gricad que le dépôt c'est bien passé ! Attention de bien sélectionner la branche que vous venez de publier la branche **`master`**, voir image ci-après. En effet, deux branches coexistent pour le moment **`main`** et **`master`**,

l'exercice qui suit permet de changer la branche par défaut et la suppression de la branche **`main`**.



3. Changer la branche courante

Sur Gricad, après votre premier push, deux branches coexistent **`main`** et **`master`**. La branche **`main`** crée par défaut lors de la création de votre projet ne sert plus, vous pouvez la supprimer. Sur la page de votre projet Gricad :

- Modifier à branche par défaut **`master`** : **`Setting > Repository > Default Branch`**
- Supprimer la branche **`main`** : **`Repository > Branch`**

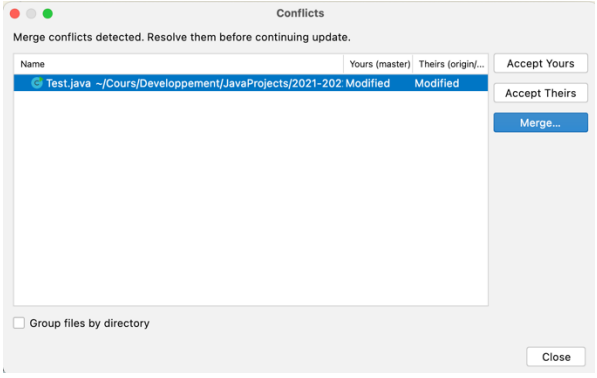
4. Récupérer un historique distant : update project ou pull (optionnel)

C'est quoi un pull ? Permet de récupérer le contenu d'un dépôt distant et pour le télécharger, puis pour mettre à jour immédiatement le dépôt local qui correspond à ce contenu.

Normalement, dans notre module, vous n'aurez pas de modification de votre dépôt distant vu que vous travaillez seul. Mais dans les projets modules (ressources ou saé) vous travaillerez surement en groupe et partagerez donc votre code à l'aide de GIT. La commande **`Git > Update Project`** (ou **`pull`**) va récupérer les dernières versions sur le serveur distant et l'ajouter à votre historique.

Si vous souhaitez tester la commande, vous pouvez **exceptionnellement** modifier un fichier sur le dépôt distant avec l'interface web d'édition proposée par Gricad. Cela va créer un nouveau commit (version) sur le serveur distant. Ensuite faite une récupération sur votre IDE **`Git > Update Project`** (ou **`pull`**) pour voir la modification apparaitre.

Si vous êtes aventureux, vous pouvez créer artificiellement un « **conflit** », c'est à dire le même fichier modifié sur le dépôt distant ET sur le dépôt local. Lors du pull, l'IDE détectera le conflit et vous demandera de le résoudre, voir image ci-après.



À vous de déterminer, si vous souhaitez accepter les modifications du dépôt distant tel quel ou les écraser avec vos modifications ou alors **combiner** de manière plus fine les modifications proposées et les vôtres, nous vous conseillons cette dernière solution. Après votre résolution de conflit, allez voir votre nouvel historique : `Git > Show Git Log`

C'est quoi un merge ? Cela permet de combiner les commits en un historique unifié.

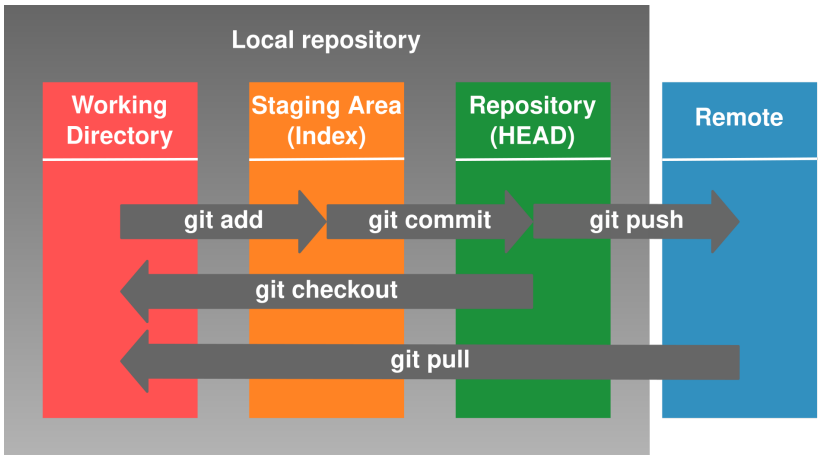
Quelle différence entre update project, pull ou fetch sur IntelliJ :

https://www.jetbrains.com/help/idea/sync-with-a-remote-repository.html#pull

---

Pour résumer

Pour résumer de manière très simplifiée :



Source : https://neurathsboat.blog/post/git-intro/

IMPORTANT : Maintenant, pour la ressource R2.01a vous devez faire des commits et des push réguliers de votre travail. Vous serez évalué sur cette régularité.