TP - MDI

git, outil de build & integration continue

Objectifs

En 6h :

- Se familiariser avec git
- Aborder les pratiques industrielles de la gestion d'une base de code partagée
- Automatiser la récupération des dépendances, du build et du packaging du projet avec un outil de build (qui dépendra du langage utilisé)
- Automatiser l'exécution des tests, la vérification de la qualité du code, la génération de rapports et la mise en ligne de votre logiciel

Vous aurez besoin de ...

- Un compte GitHub
- Un nouveau repository vide sur GitHub
- Maven si vous faites le TP en Java, Node + NPM si vous le faites en JS, Cmake (ou autre de votre choix) si vous choisissez C++
- Ces memos sur Git

Template à respecter pour vos commits

Vos commits doivent suivre le template suivant :

```
[BREAKING-API-CHANGE]? <Short description>

<Why this commit (before, after)>

This commit does the following:

* <Modification done>
* <Modification done>
```

Les petits commits sont plus faciles à relire par votre équipe.

Pull Requests (PR)

Avant de créer la Pull Request

- Vérifiez que la fonctionnalité développée fonctionne dans tous les cas d'usages représentatifs de son utilisation future
- Relisez votre code et modifiez le quand il contient des éléments non désirables :
    - Commentaires pour le développeur qui a écrit
    - println et console.log
    - duplications
    - etc.
- Si vous avez des tests unitaires, faites les tourner et vérifiez qu'ils passent bien

Créer la Pull Request

Suivez le même template que celui des commits pour apporter de la lisibilité au relecteur.

Ajoutez en plus les scénarios à tester (quality assurance : QA). Par exemple :

- Create a complete mission
- Test api errors on:
    - mission creation
    - mobilier add
    - poste add

Le TP !

Un minimum de développement.

Vous allez faire une calculatrice ... super nulle :). Elle interagira en ligne de commande.

Votre calculatrice doit avoir 2 modes : normale et scientifique. Elle doit retenir le résultat précédent et permettre de repartir à 0 ainsi que changer de mode.

Pas d'autres contraintes. L'objectif est de ne faire beaucoup de dev. Chaque personne du binôme développera un mode en parallèle de l'autre.

La todo

1. Initialisez un projet avec l'outil de build que vous avez choisi, et votre IDE. Faites cela dans un nouveau dossier. Si vous utilisez maven, vous pouvez utiliser l'archétype maven-archetype-quickstart
2. Initialisez git et faire un commit initial avec votre projet tout juste créé. Pour choisir les fichiers à mettre dans votre commit, utilisez la commande git add.
3. Importez une librairie de calcul mathématique via votre outil de build. Cela vous permet de n'avoir dans votre projet que son code et son descripteur. Aucun dépendance n'est nécessaire. L'outil de build se charge de mettre vos dépendances au bon endroit et de les ajouter a votre classpath le cas échéant.
4. Commitez ce changement.
5. Ajoutez un fichier .gitignore à la racine de votre projet. A quoi sert-il ? Que mettre dedans ?
6. Développez ensemble le début de votre calculatrice : le choix du mode (normal ou scientifique)
7. Commitez.

8. Il est temps de prendre chacun votre ordinateur. Ah mince, le code est encore en local sur la machine que vous venez d'utiliser en binôme. Mettez le en ligne sur GitHub.

9. Récupérez le code de Github sur la seconde machine.

10. Développez chacun 2 nouvelles fonctionnalités, faites le dans une branche par personne. Faites au moins un commit par fonctionnalité, plus si vous en ressentez le besoin. Vous pourrez ignorer le template de commit ici. Vous êtes totalement libre de faire ce que vous voulez, c'est votre branche après tout.

11. Une fois fini, faites votre PR. Relisez celle de votre collègue. **NE MERGEZ PAS**

12. Commencez par la première PR. Au moment de merger, sur GitHub, utilisez Squash and Merge pour n'avoir qu'un seul commit a la fin. Cette fois-ci respectez le template de commit donné ci-dessus.

13. Une fois votre première PR mergée, retournez voir vos branches sur Github. Vous verrez que la branche de la seconde PR est désormais en retard par rapport a master. Placez vous sur cette branche et rebasez la sur master.

14. Une fois rebasée, re-pushez la sur GitHub. Vous serez obligés de forcer cette opération. Pourquoi ?

15. Sur vos machines, tapez git status. Que voyez-vous ? Faites désormais git fetch . Que s'est-il passé ? Mettez a jour votre branche master locale. Prenez l'habitude de le faire après chaque PR mergée.

16. Écrivez chacun quelques tests unitaires. Idem, faites des branches et PR pour merger le tout.

17. Faites tourner les tests unitaires avec votre outil de build : trouvez la commande pour le faire.

18. Trouvez la configuration et la commande qui vous permettent de packager votre application via l'outil de build. Une fois le package en main, vous devez pouvoir faire tourner votre application en une commande.

19. Ensemble sur le même ordinateur, dans une nouvelle branche, écrivez un script (peu importe le langage) qui clean votre projet, re-télécharge les dépendances, fait tourner les tests, build le tout et en fait un package. Ne pushez pas la branche sur GitHub !! Mergez la sur master sur votre ordinateur. Allez regarder la liste de vos branches sur GitHub. Pourquoi la branche sur laquelle vous avez créé le script n'apparaît pas ?

20. Faites une branche par personne, dans laquelle vous modifiez le texte qui s'affiche en premier lorsque l'utilisateur ouvre votre logiciel. Mergez la première PR puis rebasez la branche de la seconde sur master. Oh un conflit ... y a plus qu'à le résoudre et trouver comment merger :).

21. Ajoutez quelques commentaires a votre code et générez la documentation via votre outil de build (s'il supporte cette fonctionnalité, sinon voyez s'il existe une alternative).

22. Ajoutez un outil de vérification de la qualité du code. Sur maven, un plugin checkstyle existe. Sur d'autres langage, un *linter* peut être utilisé. Il sert notamment à ajouter la vérification de style aux éditeurs de texte comme Atom, Visual Studio Code ou Sublime Text.

23. Quelle norme de codage est utilisée par défaut ?

24. Comment changer cette norme de codage, pour celle de Google ou Microsoft par exemple ?

25. Ajouter un plugin à votre outil de build pour vérifier la couverture de test et en sortir un rapport

26. Ajoutez volontairement du code mort et du code dupliqué à votre projet puis un plugin pour détecter cela. Sur Maven, vous avez PMD.

27. Intégrez désormais le plugin sonar à votre outil de build.

28. Telechargez Jenkins sur https://jenkins.io/. Choisissez la version **Java Web Archive (.war)**. Démarrez-le avec la commande java -jar jenkins.war —httpPort=9900 . Ouvrez http://localhost:9900

29. Jenkins n'a pas de support de git par défaut. Installez donc le plugin "git plugin". Créez ensuite un job pour votre projet. Dans la

configuration du job, indiquez a Jenkins l'emplacement de vos sources grâce au lien du repository GitHub. La configuration d'un job peut prendre de nombreux paramètres. Par exemple, vous pouvez définir la condition de déclenchement d'un build ("Build Triggers"), vous pouvez limiter la sauvegarde des anciens builds ("Discard old builds"), vous pouvez aussi envoyer des notifications lorsque le build échoue ("E-mail notification" qui nécessite une configuration dans "Configure System").

30. Lancer un Build pour vérifier que vous avez bien configuré votre projet.

31. Ajouter Sonar, Cobertura et PMD a votre Jenkins. Attention pour Cobertura, vous avez besoin de définir le format de sortie en xml. Deux options pour cela avec Maven : *(i)* -Dcobertura.report.format=xm, *(ii)* modifier la configuration dans votre pom et ajouter l'option de configuration appropriée (voir sur la page de Cobertura plugin)

Le plus qui vous fera gagner un temps précieux par la suite

Créez un template de projet (sur Maven c'est un archétype) dans lequel tous ces outils sont déjà intégrés et configurés, selon une configuration par défaut.

Vous le réutiliserez pour votre projet MDI.

Pensez a y mettre un dossier doc a la racine dans lequel vous documenterez les règles d'utilisation de git.

Ce genre de projet sur votre GitHub sera très bien vu lorsque vous chercherez un stage/job. Faites ça proprement et rendez-le public. N'hésitez pas à utiliser git lors de sa création. Ce sera aussi l'occasion de démontrer vos capacités à bien utiliser cet outil.

!!!! À la fin du TP, il est préférable que vous supprimiez le dossier ~/.hudson sauf si vous souhaitez conserver la configuration de votre jenkins