 <b>MASTER</b> <b>DÉVELOPPEMENT</b> <b>LOGICIEL</b>	M1 Info
	Développement collaboratif, Qualité

## TP 1 – Prise en main Git et GitHub

Remarque si vous utilisez votre propre machine :

Vous devrez installer Git (cf. diapo **18/58** du cours « DéQo2 - Introduction à Git »)

Remarque si vous utilisez les machines de l'UPS :

Les PC de l'Université ont un dual-boot Windows / Linux Fedora.

Ce TP1 ne nécessite pas d'IDE particulier (juste un terminal en ligne de commande et un éditeur de texte), mais **à partir du TP2 nous utiliserons IntelliJ IDEA Ultimate (logiciel édité par JetBrains) qui est préinstallé sous Linux.**

*Note :* en cas de besoin vous pouvez sauvegarder certains fichiers sur le cloud de Moodle (100 Mo d'espace de stockage sur <https://moodle.univ-tlse3.fr/user/files.php>)

Concernant les notions demandées dans ce sujet, vous trouverez les commandes Git nécessaires dans le support du cours « DéQo2 - Introduction à Git ».

Créer un nouveau dossier (nommé « TP-DeQo » par exemple) dans votre espace de travail.

### Exercice 1 – Création d'un compte GitHub

1. Rendez vous sur le site de GitHub : <https://github.com> et créez un compte « gratuit ». Vous recevrez ensuite un e-mail avec un lien pour vérifier votre adresse (sinon vous pourrez aller sur <https://github.com/settings/emails> pour ré-envoyer cet e-mail de vérification).
2. Rendez vous sur le projet GitHub <https://github.com/DCLL-MDL/ExemplesMoodleXML> et effectuez un fork du projet.
3. Dans votre dossier de travail, en utilisant un terminal et les commandes “git ...” vues en cours, clonez **votre** projet ExemplesMoodleXML résultant du fork exécuté précédemment. Le clone doit être effectué en utilisant le protocole **https** (préfixe de l'URL du projet : https://...). Vérifiez à l'aide de quelques commandes Git (git status, git branch -a, git remote -v) que votre projet est bien disponible localement.

### Exercice 2 – Travail en solo

1. Configurer votre identité Git en utilisant les commandes suivantes :  

```
git config --global user.name "mon_user_name"
```

```
git config --global user.email "mon_adresse"
```

 où « mon\_user\_name » est à remplacer par votre nom d'utilisateur GitHub et « mon\_adresse » est à remplacer par votre adresse mail utilisée pour la création de votre compte GitHub. Ensuite, pour vérifier, vous pouvez utiliser la commande



Ce(tte) oeuvre est mise à disposition selon les termes de la [Licence Creative Commons Paternité - Pas d'Utilisation Commerciale 3.0 France](https://creativecommons.org/licenses/by-nc/3.0/fr/).



`git config --global -l` (vue en cours).


2. Le projet contient deux fichiers XML (vous pouvez vérifier avec la commande `ls` sur le terminal). Ces fichiers sont mal formés (vous pouvez essayer de les ouvrir avec un navigateur pour constater cela). Corrigez ces fichiers avec votre éditeur de texte préféré, de manière à ce que les documents XML contenus dans ces fichiers soient bien formés. Enregistrez les fichiers corrigés.
3. Reportez les modifications dans votre dépôt local. Rappel des commandes :  
« `git status` » pour vérifier l'état du projet, ensuite « `git add .` » suivi de « `git commit -m "Commentaire..."` » et « `git status` » à nouveau pour vérifier le nouveau état du projet. PS : Voir cette note concernant les messages de commit : <https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html>
4. Créez un nouveau fichier XML intitulé « `quiz.xml` » qui agrège dans un élément racine « `quiz` » les 2 documents XML corrigés précédemment, en vous aidant de la documentation [https://docs.moodle.org/22/en/Moodle\\_XML\\_format#Overall\\_structure\\_of\\_XML\\_file](https://docs.moodle.org/22/en/Moodle_XML_format#Overall_structure_of_XML_file). Enregistrez le fichier.
5. Reportez les modifications dans votre dépôt local.
6. Créez une branche intitulée « `dev/exempleTrueFalse` » et basculez sur cette branche. Dans cette branche, créez un nouveau fichier « `TrueFalseQuestion.xml` » et ajoutez **un exemple de question** de type `true/false` en vous aidant de la documentation [https://docs.moodle.org/22/en/Moodle\\_XML\\_format#True.2Ffalse](https://docs.moodle.org/22/en/Moodle_XML_format#True.2Ffalse) et en vous inspirant du fichier « `MultipleChoiceQuestion.xml` » déjà existant. Enregistrez votre fichier et ajoutez-le dans le dépôt dans votre branche « `dev/exempleTrueFalse` »
7. Visualisez l'état de l'historique avec les commandes « `git log` » et « `gitk` », puis « Envoyez » votre branche « `dev/exempleTrueFalse` » sur votre dépôt GitHub. Rendez vous sur le site GitHub pour vérifier que vos modifications sont bien prises en compte sur le dépôt distant.
8. Rebasculez sur la branche `master` et effectuez un merge de la branche « `dev/exempleTrueFalse` ».  
**Est-ce le merge effectué est du type « `fast-forward` » ? Qu'est-ce que cela veut dire un merge `fast-forward` ?**
9. « Envoyez » les modifications sur la branche `master` sur votre dépôt GitHub. Vérifiez sur le site GitHub que vos modifications sont bien prises en compte.

### Exercice 3 – Travail en duo

Vous allez maintenant voir comment Git supporte le développement collaboratif.

1. Choisissez un binôme. Un des 2 binômes doit être désigné comme titulaire du projet partagé GitHub. Le titulaire du projet partagé doit, sur GitHub, ajouter son binôme à la liste des contributeurs de son projet. L'autre binôme doit alors effectuer dans son dossier de travail un clone du projet correspondant au projet partagé. Une



 <b>MASTER</b> <b>DÉVELOPPEMENT</b> <b>LOGICIEL</b>	M1 Info
	Développement collaboratif, Qualité

fois les manipulations réalisées, les 2 binômes exécutent la commande « `git remote -v` » pour vérifiez que les 2 binômes pointent sur le même dépôt distant.

2. Chaque binôme doit alors :
  1. créer une branche dans son dépôt local. Chaque binôme choisit un nom de branche distinct.
  2. Basculer sur la nouvelle branche.
  3. Ajouter un fichier XML illustrant un nouvel exemple de question au format XML Moodle en s'appuyant sur la documentation Moodle (cf. lien ci-dessus).
  4. Enregistrer et commiter le nouveau fichier sur sa propre branche.
  5. « Envoyer » la branche en question sur le dépôt GitHub partagé.
  6. Vérifier sur GitHub que les 2 branches sont bien dans le dépôt partagé.

#### Exercice 4 – Résolution de conflits

1. Chaque binôme doit basculer sur sa branche master et effectuer un merge de la branche précédemment créée.
2. Pour simuler un conflit (amicale) entre binômes, chaque binôme modifie dans son éditeur de texte préféré le même endroit du même fichier dans sa branche « master ».  
Enregistrez et committez le fichier sur vos branches.
3. Un des binômes « envoie » les modifications sur la branche master du dépôt GitHub partagé.
4. L'autre binôme essaie « d'envoyer » les modifications sur la branche master du dépôt. Vérifier qu'un message d'erreur apparaît suggérant de faire « `git pull` ». Faites-le et vérifiez qu'un conflit est détecté. Examinez le conflit avec la commande « `git diff` ».
5. Le même binôme résout le conflit en éditant le fichier qui est en conflit, et en commitant. Ensuite, ce binôme peut « envoyer » les modifications sur la branche master du dépôt.
6. Le premier binôme doit maintenant mettre à jour son dépôt avec les modifications apportées par le deuxième binôme sur GitHub.
7. Définissez avec votre binôme une séquence opératoire permettant maintenant d'aboutir à un conflit sur un fichier **entre deux branches d'un même dépôt local**. Pour cela, vous pouvez travailler tous les deux sur la même machine en mode «pair-programming» : choisissez laquelle de vos machines vous allez utiliser, et qui sera chargé de taper les commandes pour simuler le conflit. Mettez en œuvre la séquence opératoire décidée et résolvez le conflit.

