Organisation GitHub Nuit de l’Info

# Rappels Git

**git add**

* Permet d’ajouter un/des fichiers dans la « staging area » (zone contenant les fichiers qui vont être inclus dans le prochain commit)
* <fichier> : Sélectionne un fichier précis
* <dossier> : Sélectionne tout les fichiers d’un dossier
* . ou \* : Sélectionne tous les fichiers déjà suivis (lorsqu’ils viennent d’être créés, il faut les ajouter un à un)

**git commit**

* Fabriquer un commit avec les fichiers suivis
* -m « descriptif » : Pour mettre le message du commit
* -a : pour inclure tous les fichiers déjà suivis dans le commit

**git cherry-pick <idCommit>**

* Récupère le commit dont l’id est en paramètres, peu importe sa branche.

**git switch <branche>**

* Se déplace dans la branche
* -c : pour fabriquer la branche en se déplaçant dedans

**git branch**

* Affiche les branches locales
* -a : affiche également les branches distantes
* -b : supprimer la branche locale
* -B : force la suppression globale (lorsqu’il pense, à tort, que la branche locale est en avance sur la distante et empêche la suppression)

**git merge <branche>**

* Fusionne la branche sélectionnée DANS la branche courante
* Aucun : fusionne les commits uns à uns en les gardant tous
* --squash : fusionne toutes les modifications des 2 branches en un seul commit (voir schéma)

Une image contenant diagramme, texte, cercle, croquis

Description générée automatiquement

# Structure adoptée

Une image contenant texte, diagramme, cercle, capture d’écran

Description générée automatiquement

* **master** : versions fonctionnelles et stables, qui servira de vitrine (seuls les responsables y touchent)
* **preMaster** : Chaque commit = une tâche entière gérée par une équipe (fonctionnalité, réparation…) ayant été validée par le biais d’une pull request.
* **documentation** : contient tous les fichiers de doc (comme les fichiers binaires style loopings). Attention : pour modifier une documentation, il faut aller modifier l’unique version sur la branche Documentation, la commit, la push et ENSUITE récupérer une copie sur sa branche avec un cherry-pick. TOUJOURS préfixer le commit avec DOCS
* **Branches temporaires** : ces branches sont libres, vous utilisez la nomenclature et la structure que vous préférez. Elles peuvent être décomposées en plusieurs branches si nécessaire.

# Fabrication d’une pull request

1. **Squash and merge** la branche de destination dans la sienne
2. **Créer** la pull request sur GitHub (UNIQUEMENT vers preMaster)
3. Rédiger une **description** détaillée et un **titre** de commit de la forme :   
   <N°DEFI>-<TYPE COMMIT> : <DESCRIPTION>
   1. N°DEFI: identifiant du défi tel que D1 pour le défi 1, D2 pour le défi 2….
   2. TYPE COMMIT : on choisit ce qui définit GLOBALEMENT au mieux le commit (c’est pas grave si on a un peu de fix dans un feat par exemple)
      1. FEAT : Fonctionnalité développée
      2. FIX : réparation
      3. REFACTOR : changement code
      4. STYLE : modifications graphiques
      5. DOCS : documentation
4. **Attendre la validation** d’autres membres du même défi (au minimum 2)
5. Lorsque c’est validé, demander la dernière validation du **responsable** en l’appelant.
6. Si c’est bon, **valider le Squash and merge** sur preMaster.

# Règles

* TOUJOURS **pull avant de push** pour ne pas écraser de travail
* Activer le **formatage** **automatique** lors des CTRL S
  + pour limiter le nombre de modifications inutiles dans les commits.
  + Sur VSCode : Activer « Format on Save » dans les paramètres
* **Supprimer** les branches en locales lorsqu’on n’y touche pas / plus.
* Suivre les règles de fabrication de **pull request** ci-dessus.
* Ne pas hésiter à **demander de l’aide** au premier problème