**1) Pourquoi utiliser Git ?**

* Git est **rapide**.
* Il permet de **travailler avec des branches** facilement.
* Il est un peu complexe au début mais **très puissant**.
* Des versions existent avec des **interfaces graphiques** (Windows, Mac) pour simplifier son utilisation.
* Des sites comme **GitHub** ou **Gitorious** permettent de collaborer en ligne.

**2) Vocabulaire de base**

* **Dépôt (repository)** : dossier où Git stocke ton projet et tout son historique.
* **Commit** : un point de sauvegarde, une version enregistrée.
* **Branche (branch)** : une ligne de développement indépendante.
* **HEAD** : le pointeur qui indique sur **quelle branche** ou **quel commit** on est positionné.

**3) Workflow de collaboration : le Forking**

1️⃣ Le propriétaire crée le **dépôt officiel** (remote).  
2️⃣ Les contributeurs font un **fork** (copie) de ce dépôt dans leur propre dépôt distant.  
3️⃣ Chacun **clone** son fork en local sur son ordinateur.  
4️⃣ Ils travaillent dessus, font des **commits**.  
5️⃣ Ils **pushent** leurs modifications dans leur fork en ligne.  
6️⃣ Ils proposent au propriétaire d’**intégrer** leurs modifications (**pull request**).  
7️⃣ Le propriétaire révise et fusionne les modifications dans le dépôt officiel.

**4) Branches et commandes principales**

* Chaque **commit** a un identifiant unique (**SHA**).
* Une **branche** pointe vers un commit.
* On peut **créer une branche** depuis n’importe quel commit.
* Le pointeur **HEAD** indique la branche courante.
* checkout : changer de branche ou revenir à un ancien commit.
* merge : fusionner deux branches.
* rebase : réécrire l’historique pour le rendre plus linéaire et lisible.
* **Astuce** : pour tester une idée ➝ créer une branche ➝ tester ➝ fusionner si l’idée est validée.

**5) Annuler un commit**

* ⚠️ Ne pas utiliser reset pour annuler un commit déjà partagé.
* Utiliser revert ➝ cela **crée un nouveau commit** qui annule **uniquement** les changements du commit ciblé.
* Cela garde l’historique **propre et compréhensible**.

**6) Gérer les conflits**

* Quand deux branches modifient **la même partie d’un fichier**, Git génère un **conflit**.
* Git ajoute des marqueurs (<<<<<<<, =======, >>>>>>>) pour montrer les différences.
* Il faut choisir quelle version garder ou les combiner manuellement.
* Ensuite, on **supprime les marqueurs**, puis on valide la résolution avec un **commit**.
* Des outils comme **SmartGIT** peuvent aider à résoudre visuellement les conflits.

**7) Bonnes pratiques avec Git**

✅ Commiter des modifications **cohérentes et liées entre elles**.  
✅ **Tester le code** avant chaque commit.  
✅ Travailler **avec des branches** pour séparer les fonctionnalités.  
✅ Commiter **souvent** pour garder un historique précis.  
✅ Rédiger des **messages clairs et précis** pour les commits.  
✅ Éviter de commiter du travail **non terminé**.  
✅ Rappelle-toi : Git est un **outil de versionnage**, pas une **sauvegarde**.

**Conclusion**

* **Git** garde tout l’historique du projet.
* Il facilite le **travail en équipe**.
* Il permet de **tester des idées sans risque** grâce aux branches.
* On peut toujours revenir à une version antérieure **proprement**