Prise en main de Git & GitLab sur un projet Laravel

Objectif : être autonome sur Git et GitLab au quotidien - pas pour coder en Laravel, mais parce que toute l'infra qu'on gère (déploiement, config serveur, secrets, CI/CD) passe par des dépôts Git.

Règle du jeu : je ne te donne aucune commande à copier-coller. Tu cherches toi-même - git help <commande>, la doc officielle de Git, la doc GitLab, un moteur de recherche.

Captures d'écran : pour chaque étape numérotée ci-dessous, prends une capture d'écran montrant la commande que tu as tapée **et** son résultat dans le terminal. Enregistre-les dans un dossier `captures/` à côté de ton `journal.md`, nommées avec le numéro de l'étape. Si une étape contient plusieurs commandes distinctes à montrer, ajoute un suffixe.

Note dans `journal.md` chaque commande que tu utilises, une phrase sur ce qu'elle fait et les capture d'écran associées. Ce journal, c'est ton livrable en plus du repo et des captures.

Phase 0 - Mise en place

0.1 - Crée-toi un compte GitLab si ce n'est pas déjà fait.

0.2 - Génère une paire de clés SSH et associe la clé publique à ton compte.

0.3 - Crée un nouveau projet GitLab vide.

0.4 - Génère un projet Laravel sur ta machine et relie-le à ton dépôt GitLab distant.

Phase 1 - Git tout seul, sans branche, sans réseau

Le but ici : que les gestes de base deviennent des réflexes. Fais chaque point plusieurs fois, sur des fichiers différents du projet Laravel, jusqu'à ce que tu n'aies plus besoin de rechercher la commande.

1.1 - Observer l'état du dépôt

- Vérifier comment Git décrit l'état de ton dépôt avant tout ajout.
- Modifie le fichier README.md et observe comment il est signalé.

- Crée un nouveau fichier (jamais vu par Git) et observe comment il est signalé.
- Modifie deux fichiers différents et observe comment Git les liste tous les deux.

1.2 - Voir les différences

- Regarde la différence exacte, ligne par ligne, entre ce que tu as modifié dans ton dossier de travail et ce qui est enregistré dans le dernier commit.
- Fais la même chose mais uniquement sur un seul fichier précis, pas sur tout le dépôt.

1.3 - Indexer

- Ajoute un seul fichier précis à l'index (staging area), pas les autres.
- Regarde la différence entre ce qui est indexé et ce qui ne l'est pas.
- Retire un fichier de l'index sans perdre la modification qu'il contient.
- Indexe maintenant tous les fichiers modifiés d'un coup.

1.4 - Committer

- Fais un commit avec un message clair et descriptif.
- Fais-en un deuxième juste après une petite correction, puis regarde comment "corriger" le tout dernier commit (message ou contenu) sans en créer un nouveau.

1.5 - Consulter l'historique

- Affiche l'historique des commits en version courte.
- Affiche l'historique en version détaillée, avec les fichiers modifiés à chaque commit.
- Affiche le contenu exact d'un commit précis.
- Compare deux commits entre eux
- Compare l'état actuel avec un commit plus ancien.
- Trouve qui a modifié une ligne précise d'un fichier et dans quel commit.

1.6 - Annuler des choses

- Modifie un fichier, puis annule cette modification pour revenir à la version du dernier commit (sans avoir committé).
- Fais un commit "de test" exprès, puis annule-le de deux façons différentes et prends le temps de bien observer la différence entre les deux.
    - une fois en gardant les modifications dans ton dossier de travail,
    - une fois en les supprimant complètement.

- Regarde aussi comment annuler proprement un commit déjà partagé avec d'autres (une méthode qui ajoute un nouveau commit d'annulation plutôt que de réécrire l'historique).

1.7 - Mettre de côté sans committer

- Modifie un fichier, puis mets cette modification de côté temporairement sans la committer ni la perdre. Vérifie que ton dossier de travail redevient parfaitement propre.
- Fais une autre modification par-dessus, committe-la.
- Récupère ensuite ta modification mise de côté et observe comment elle vient s'appliquer par-dessus ce qui existe déjà.

1.8 - Ignorer des fichiers

- Crée un fichier .gitignore et fais en sorte qu'un dossier généré automatiquement par Laravel (comme vendor/ ou node_modules/) ne soit jamais suivi par Git, même si tu fais un "ajouter tous les fichiers".
- Vérifie qu'un fichier déjà suivi par erreur avant la création du .gitignore ne disparaît pas tout seul, et remédie à la situation.

1.9 - Étiqueter

- Crée un tag sur un commit précis, pour marquer par exemple une "version" du projet.
- Liste les tags existants et affiche les infos d'un tag précis.

Phase 2 - Branches et fusion, toujours en local

Toujours sans GitLab. On ajoute juste les branches et la notion de dépôt distant.

2.1 - Créer et naviguer

- Crée plusieurs branches à partir de la branche principale, liste-les, bascule de l'une à l'autre.
- Renomme une bra nche.
- Supprime une branche que tu n'utilises plus

2.2 - Fusionner

- Fais des commits différents sur deux branches distinctes.

- Fusionne une branche dans une autre en local. Fais-le une première fois dans une situation où ça se résout tout seul sans rien te demander, puis observe une situation où Git crée un commit de fusion en plus

2.3 - Dépôt distant

- Vérifie quels dépôts distants sont configurés sur ton projet.
- Récupère les dernières modifications du dépôt distant sans les fusionner automatiquement, regarde ce que ça a rapporté, puis fusionne-les toi-même. Ensuite, fais la même chose mais avec la commande qui fait les deux étapes en une seule.
- Envoie tes branches locales vers GitLab.

Phase 3 - Gérer un conflit (toujours en local)

3.1 - Crée deux branches différentes qui modifient la **même ligne** d'un même fichier (par exemple une valeur dans config/app.php ou une ligne du README.md).

3.2 - Fusionne la première branche sans problème, en local.

3.3 - Essaie de fusionner la seconde : tu vas obtenir un conflit. Résous-le à la main, puis termine la fusion.

Phase 4 - Collaboration via GitLab

4.1 - Envoie une de tes branches vers GitLab et ouvre une **Merge Request** vers la branche principale.

4.2 - Ajoute-moi comme reviewer.

4.3 - Configure une protection sur la branche principale : personne ne doit pouvoir y pousser directement, tout doit passer par une MR.

4.4 - Une fois la MR approuvée, fusionne-la depuis l'interface, puis mets à jour ton dépôt local pour récupérer le résultat.

Phase 5 - Volet sécurité

Un projet Laravel contient un fichier .env avec des secrets (clés d'API, mots de passe de base de données...).

5.1 - Vérifie si le .env du projet est suivi par Git ou non. Si besoin, corrige la situation pour qu'il ne soit **jamais** commité, sans supprimer le fichier lui-même.

5.2 - Cherche comment révoquer une clé SSH ou un token d'accès personnel sur GitLab, et dans quel cas on le ferait.

5.3 - Regarde la différence entre un **Personal Access Token**, un **Project Access Token** et un **Deploy Token** dans GitLab.

Questions de synthèse (à répondre par écrit dans journal.md)

Ces questions reprennent des points croisés tout au long de l'exercice. Réponds-y avec tes propres mots, en quelques phrases chacune - ce n'est pas un contrôle de vocabulaire, c'est pour vérifier que tu as compris et pas juste exécuté des commandes. Pas de capture d'écran nécessaire pour cette section, ce sont des réponses rédigées.

Q1 (réf. 0.2) - Entre l'authentification SSH et HTTPS avec token, laquelle as-tu choisie pour te connecter à GitLab, et pourquoi ?

Q2 (réf. 1.3) - Quelle est la différence entre indexer un seul fichier parmi plusieurs modifiés et indexer tous les fichiers d'un coup ?

Q3 (réf. 1.6) - Quelle est la différence entre annuler un commit en gardant les modifications dans le dossier de travail, et l'annuler en les supprimant complètement ? Pourquoi cette distinction peut-elle avoir des conséquences sérieuses en environnement pro ?

Q4 (réf. 2.3) - Quelle est la différence entre `fetch` et `pull` ?

Q5 (réf. 2.2) - Quelle est la différence entre une fusion fast-forward et une fusion qui crée un commit de fusion ?

Q6 (réf. 3.3) - Comment as-tu résolu ton conflit ? Comment as-tu choisi quelle version garder ?

Q7 (réf. 4.3, 4.4) - Quelle est la différence entre fusionner une Merge Request depuis l'interface GitLab et fusionner une branche en local ? Pourquoi protège-t-on une branche en environnement de production ?

Q8 (réf. 5.1) - Pourquoi ne faut-il jamais committer un fichier .env, même avec des valeurs de test anodines ?

Q9 (réf. 5.2) - Dans quel cas faudrait-il révoquer une clé SSH ou un token d'accès personnel sur GitLab ?

Q10 (réf. 5.3) - Quelle est la différence entre un Personal Access Token, un Project Access Token et un Deploy Token ? Lequel utiliserais-tu pour un serveur de production qui doit seulement récupérer le code (lecture seule), et pourquoi ?

Q11 (réf. 1.4) - Dans quel cas est-il utile de corriger le tout dernier commit plutôt que d'en créer un nouveau ?

Q12 (réf. 1.5) - En quoi la commande qui retrouve qui a modifié une ligne précise et dans quel commit peut-elle t'être utile dans ton futur métier ?

Q13 (réf. 1.6) - Dans quel cas dois-tu annuler un commit déjà partagé avec une méthode qui ajoute un nouveau commit d'annulation, plutôt que de réécrire l'historique ?

Q14 (réf. 1.8) - Pourquoi un fichier déjà suivi par Git avant la création du .gitignore continue-t-il d'apparaître dans le suivi malgré le .gitignore ? Comment y remédier proprement ?

Q15 (réf. 2.1) - Que se passe-t-il si tu essaies de supprimer une branche qui contient des commits jamais fusionnés ailleurs ? Pourquoi Git te protège-t-il ainsi ?

Rendu final

- Le lien vers ton projet GitLab (accès en lecture pour moi).
- Le dossier `captures/` avec une capture par étape numérotée (commande + résultat).
- Ton journal.md qui constitue le rapport complet de l'exercice, avec les commandes utilisées, leur rôle, les références aux captures correspondantes, et les réponses aux questions de synthèse de la phase 5.