Le Remisage avec Git

¶

Note pour ce TP :

- **Travail en classe (10 points)**
- **Compte-rendu (10 points) à rendre dans 7 jours**

Compte-rendu :

Afin de valider votre travail sur ce TP, il vous est demandé de produire un compte-rendu respectant les critères suivants :

1. Réponses aux Questions :

- Assurez-vous de répondre de manière précise et claire à toutes les questions posées lors du TP.
- Chaque réponse doit être argumentée. Si vous avez des doutes ou des incertitudes, mentionnez-les, cela peut faire partie du processus d'apprentissage.

2. Impressions Écrans :

- Intégrez au compte-rendu des captures d'écran (impressions écrans) illustrant les étapes cruciales ou les résultats obtenus.
- Veillez à ce que ces captures soient lisibles et de bonne qualité.

3. Explications Complémentaires :

- Si une partie du TP vous a semblé complexe, ambiguë, ou si vous avez apporté des modifications notables, expliquez vos choix et démarches.
- Vos explications permettront d'évaluer votre compréhension et votre capacité à analyser et à justifier vos actions.

4. Mise en Forme :

- Veillez à la clarté et à la propreté de votre document. Une présentation soignée facilite la lecture et montre votre investissement.
- Organisez votre compte-rendu de manière logique, en suivant l'ordre du TP ou en regroupant les informations par thématique.

5. Envoi du Compte-rendu :

- Nommez votre fichier selon le format suivant : CR_TP_Nom_Prénom.
- format PDF.

I. A quoi sert *git stash* ? ~~(C.R.)~~

1. Présentation

La commande ***git stash*** est un outil puissant pour gérer des modifications temporaires, en vous permettant de basculer entre différentes tâches et de maintenir une flexibilité dans votre flux de travail.

Elle permet de mettre de côté ("stasher") temporairement des modifications que vous avez faites mais que vous ne souhaitez pas commit immédiatement. Elle est souvent utilisée dans des situations où vous êtes au milieu d'un travail (et n'avez donc pas un état propre pour commit), mais avez besoin de changer de branche ou de réaliser une autre tâche sans perdre vos modifications en cours.

2. Voici quelques scénarios typiques où git stash est utile :

a) Changer de tâche rapidement :

Imaginez que vous travaillez sur une nouvelle fonctionnalité, mais soudainement, un bug critique est signalé et vous devez changer de branche pour le corriger. Au lieu de commit des travaux inachevés, vous pouvez utiliser ***git stash*** pour sauvegarder temporairement vos modifications, changer de branche, corriger le bug, puis revenir et récupérer vos modifications.

b) Obtenir un état propre :

Parfois, vous voulez simplement obtenir un état propre (par exemple, pour créer une nouvelle branche ou pour tirer les dernières modifications du dépôt distant) sans perdre vos modifications en cours.

c) Mélanger des changements entre branches :

Si vous avez travaillé dans une branche mais réalisez ensuite que ces modifications seraient mieux placées dans une autre branche, ***git stash*** peut aider à transférer ces modifications.

3. Comment ça marche ?

Voici une explication rapide de quelques commandes ***git stash*** courantes :

- ***git stash*** ou ***git stash push*** : Met de côté vos modifications. Vous pouvez donner une description pour vous rappeler ce que vous avez mis de côté avec : ***git stash push -m "Description de mes modifications"***.

- ***git stash list*** : Montre la liste des modifications mises de côté.

- ***git stash pop*** : Récupère (et supprime) les modifications de la dernière pile de stash et les applique à votre branche actuelle.

- ***git stash apply*** : Similaire à pop, mais ne supprime pas les modifications de la pile de stash.

- * ***git stash h drop stash@{n}*** : Supprime les modifications mises de côté à la position n (par exemple stash@{0} pour le premier élément).

- ***git stash clear*** : Supprime toutes les entrées mises de côté.

II. Exemple d'utilisation~~(C.R.)~~

1. Scénario :

Vous développez actuellement une fonctionnalité dans une application web qui permet aux utilisateurs de s'inscrire à une newsletter. Pour cela, vous êtes dans une branche spécifique nommée **feature-newsletter**. Vous avez déjà effectué quelques modifications sur le fichier **newsletter.html** et le fichier **styles.css** pour cette nouvelle fonctionnalité. Cependant, avant de pouvoir terminer, vous apprenez qu'un bug critique doit être corrigé de toute urgence sur la branche main.

Plutôt que de committer un travail inachevé dans la branche **feature-newsletter**, vous décidez d'utiliser ***git stash*** pour mettre de côté vos modifications actuelles, de sorte à pouvoir changer de branche et traiter le bug.

Une fois le bug corrigé, vous souhaitez revenir à votre branche de fonctionnalité et reprendre votre travail là où vous l'avez laissé

2. Étapes :

1. Effectuer les dernières modifications sur votre travail

Avant de passer à une autre tâche, vous terminez votre travail concernant, par exemple, les fichiers **newsletter.html** et **styles.css**. Mettez des commentaires afin de vous rappeler où vous en étiez.

2. Utiliser git stash pour mettre de côté les modifications

Avant de changer de branche, vous décidez de "stasher" vos modifications.
git stash push -m "Début du développement de la fonctionnalité newsletter"

3. Changer de branche pour la mise à jour urgente

Vous pouvez maintenant changer sereinement de branche sans emporter vos modifications avec vous.
git checkout main

4. Apporter la correction urgente et la pousser

Après avoir corrigé le problème de sécurité, vous committez et poussez vos changements.

5. Revenir à votre branche de fonctionnalité

git checkout feature-newsletter

6. Récupérer vos modifications mises de côté avec git stash

Pour reprendre là où vous vous étiez arrêté avec votre fonctionnalité, utilisez :
git stash pop

7. Continuer votre développement

Vous constatez que les fichiers **newsletter.html** et **styles.css** sont de nouveau dans l'état où vous les aviez laissés. Vous pouvez maintenant continuer à travailler sur votre fonctionnalité.

III. Premier mini-projet - Guidé - : gestionnaire de tâches en Python

Avec ce TP, vous avez non seulement apprendre à utiliser git stash dans divers scénarios, mais vous avez également réfléchi à son importance et à ses meilleures pratiques dans le processus de développement.

Contexte :
Vous êtes en train de construire un "Gestionnaire de Tâches" en tant qu'application console Python. Pendant le développement, vous vous retrouvez souvent à jongler entre plusieurs modifications.
Découvrez comment **git stash** peut vous aider à sauvegarder, restaurer et examiner ces modifications de manière temporaire.

Partie 1 : Initialisation du projet

Étape N°1 : Créez un nouveau dépôt git : git init.
Étape N°2 : Créez un fichier Python nommé gestionnaire_taches.py.
Étape N°3 : Ajoutez une structure de base pour votre gestionnaire de tâches comme décrit ci-dessus.
Étape N°4 : Committez votre travail initial.

Étape N°5 : Question : Pourquoi est-il utile d'initialiser un dépôt git dès le début d'un projet, même si le projet n'est pas encore complet?

Partie 2 : Introduction à git stash

Étape N°6 : Commencez à développer une fonction pour ajouter des tâches, mais laissez-la inachevée.

Étape N°7 : Mettez de côté ces modifications.

Étape N°8 : Question : Pourquoi utiliser git stash au lieu de simplement créer un nouveau commit pour sauvegarder ces modifications temporaires?

Partie 3 : Utilisation de git stash dans différents scénarios

Étape N°9 : Ajoutez une fonctionnalité pour marquer les tâches comme terminées, mais ne la terminez pas complètement.

Étape N°10 : Sauvegardez ces modifications.

Étape N°11 : Affichez vos stashes avec git stash list.

Étape N°12 : Question : Quelle est la différence entre git stash pop et git stash apply?

Étape N°13 : Appliquez un stash spécifique pour finaliser une fonction.

Étape N°14 : Examinez les modifications dans un stash spécifique.

Étape N°15 : Question : Comment visualiser précisément les modifications d'un stash sans l'appliquer?

Étape N°16 : Supprimez un stash spécifique.

Étape N°17 : Nettoyez tous vos stashes.

Étape N°18 : Question : Dans quel scénario pourriez-vous avoir besoin de supprimer tous vos stashes à la fois?

IV. Deuxième mini-projet : gestion d'employés

Contexte :

Vous travaillez sur un mini-projet Python qui implémente un système de gestion d'employés. Alors que vous développez une nouvelle fonctionnalité, une demande urgente nécessite votre attention sur une autre partie du code.

1. Étape 1 : Mise en place du projet

Étape N°19 : Initialisez un nouveau dépôt git

Étape N°20 : Créez un fichier Python nommé gestion_employes

Étape N°21 : Ajoutez une classe Employe avec des méthodes pour définir le salaire, afficher les détails de l'employé, etc.

- Constructeur avec nom, prenom et salaire
- Méthode afficher_details qui renvoie une chaîne formatée qui insère les valeurs des attributs prenom, nom et salaire
- Une autre méthode definir_salaire permet de définir ou de modifier le salaire d'un employé

Étape N°22 : Créez un autre fichier qui va tester cette classe.

Étape N°23 : Commitez votre travail

2. Étape 2 : Utilisation basique de git stash

Vous commencez à développer une nouvelle méthode pour augmenter le salaire en pourcentage, mais elle reste incomplète :

Étape N°24 : Déclarez cette méthode et y mettre *pass*

Tout à coup, une demande urgente vous oblige à ajouter une méthode pour afficher l'adresse mail de l'employé (qui est simplement prenom.nom@entreprise.com).

Étape N°25 : Avant de passer à cette tâche, utilisez git stash pour mettre de côté votre méthode incomplète.

Étape N°26 : Ajoutez rapidement la méthode pour générer l'adresse mail :

Étape N°27 : Committez cette nouvelle méthode

Étape N°28 : Testez votre programme

3. Étape 3 : Exploration de git stash

Étape N°29 : Affichez la liste des modifications mises de côté

Étape N°30 : Récupérez votre travail inachevé

Étape N°31 : Terminez la méthode augmenter_salaire

Étape N°32 : Committez cette méthode.

4. Bilan étapes 1 à 3

Étape N°33 : Si un développeur commence à ajouter une nouvelle méthode à la classe Employe mais décide de ne pas la finaliser, quelle commande git pourrait-il utiliser pour conserver temporairement ces changements ?

Étape N°34 : Après avoir utilisé git stash sur des modifications de la classe Employe, comment le développeur peut-il supprimer ces modifications mises de côté s'il décide de ne jamais les utiliser ?

Étape N°35 : Quel est l'avantage d'utiliser git stash drop lors de la manipulation de la classe Employe, surtout si le développeur a de nombreux stashes ?

Étape N°36 : Comment git stash peut-il aider un développeur à maintenir un historique de commits propre tout en travaillant sur des modifications temporaires pour la classe Employe?

Étape N°37 : Si un développeur a plusieurs stashes liés à des modifications sur la classe Employe, comment peut-il s'assurer que seules les modifications pertinentes sont conservées et que les autres sont éliminées?

5. Étape 3 : Exploration approfondie de git stash

Étape N°38 : Ajoutez une méthode pour définir le poste de l'employé, mais vous ne souhaitez pas encore le committer (*definir_poste(self, poste)*)

Étape N°39 : Mettez cette modification de côté avec un message descriptif

Étape N°40 : Supposons que vous ayez maintenant plusieurs modifications dans votre stash. Affichez une description détaillée de ce qui a été modifié dans chaque stash.

Étape N°41 : Vous souhaitez maintenant récupérer une modification précise de votre stash (pas la dernière). Quelle commandez pouvez-vous utiliser ? Donnez un exemple.

Étape N°42 : Quelle est la commande pour pour appliquer et supprimer simultanément un stash spécifique ? Donnez un exemple.

6. Étape 4 : Gérer les conflits avec git stash

Étape N°43 : Sur la branche actuelle, modifiez la méthode *definir_salaire* pour qu'elle refuse les salaires négatifs :

Étape N°44 : Committez votre travail.

Étape N°45 : Appliquez maintenant une modification mise de côté (stash) qui modifie également la méthode definir_salaire. Par exemple, une version qui change le salaire seulement s'il est augmenté. Par exemple :

```
# Exemple d'utilisation:
employe1 = Employe("Alice", 50000)
```

```
employe1.definir_salaire(52000)  # Affichera : "Le salaire de Alice a été mis à jour à 52000 euros."
employe1.definir_salaire(51000)  # Affichera : "La proposition de salaire pour Alice n'est pas une augmentation. Aucune mise à jour effectuée."
```

Étape N°46 : Vous devriez rencontrer un conflit. Résolvez-le en choisissant la version qui convient le mieux ou en fusionnant manuellement les modifications.

Étape N°47 : Après avoir résolu le conflit, n'oubliez pas de faire un git add . pour marquer le conflit comme résolu.

7. Étape 5 : Nettoyage et bonnes pratiques

Étape N°48 : Avec plusieurs entrées dans votre stash, supposons que vous souhaitez nettoyer toutes les anciennes entrées pour éviter la confusion. Quelle commande devez-vous utiliser ?

En pratique, essayez d'utiliser git stash pour des modifications temporaires et non pour stocker des modifications à long terme. Si une fonctionnalité ou une correction nécessite un temps de développement plus long, envisagez d'utiliser une branche dédiée.

8. Bilan final

Ces questions sont basées sur le paragraphe fourni et ciblent la manière dont les commandes git peuvent être utilisées lors de la manipulation d'une classe spécifique, en l'occurrence Employe.

Étape N°49 : Si un développeur commence à ajouter une nouvelle méthode à la classe Employe mais décide de ne pas la finaliser, quelle commande git pourrait-il utiliser pour conserver temporairement ces changements?

Étape N°50 : Après avoir utilisé git stash sur des modifications de la classe Employe, comment le développeur peut-il supprimer ces modifications mises de côté s'il décide de ne jamais les utiliser ?

Étape N°51 : Quel est l'avantage d'utiliser git stash drop lors de la manipulation de la classe Employe, surtout si le développeur a de nombreux stashes ?

Étape N°52 : Comment git stash peut-il aider un développeur à maintenir un historique de commits propre tout en travaillant sur des modifications temporaires pour la classe Employe?

Étape N°53 : Si un développeur a plusieurs stashes liés à des modifications sur la classe Employe, comment peut-il s'assurer que seules les modifications pertinentes sont conservées et que les autres sont éliminées?