Використання 5 команд Git-a: stash, rebase, patch, cherry pick, amend

(відпрацьовуються усіма учасниками проєкту)

1. git stash

Команда "git stash" використовується для тимчасового зберігання змін в робочій директорії. Далі наведено приклади завдань для роботи команди з використанням команди `git stash`:

Тімлід:

- Пояснює розробнику та тестувальнику, як використовувати команду `git stash` для збереження змін у робочій директорії перед внесенням надто складних або ризикованих змін. Проводить ретроспективу, спеціальну зустріч з учасниками проекту.

- Встановлює процедуру використання команди `git stash` у проекті та забезпечує необхідну підтримку для розробника та тестувальника.

Розробник:

- Перед початком роботи над новою функціональністю або виправленням помилок використовує команду `git stash`, щоб зберегти поточні зміни.

- Виконує збережені зміни за допомогою команди `git stash pop` або `git stash apply`, коли готовий повернутися до них.

Тестувальник:

- Використовує команду `git stash` для збереження змін, які можуть вплинути на процес тестування.

- Повертається до збережених змін за потреби, використовуючи команди `git stash pop` або `git stash apply`, коли тестування потребує відновлення попереднього стану.

Команда `git stash` використовується для тимчасового збереження змін у робочій директорії. Вона дозволяє вам прибрати зміни, які ви не готові закомітити, але вам не хочеться їх втрачати.

Використовуйте `git stash` для тимчасового збереження змін, коли ви хочете переключитися на іншу гілку чи вирішити іншу проблему, не комітячи незакінчену роботу.

Використання команди `git stash`:

1. Збереження змін:

Ви можете використовувати `git stash`, щоб прибрати всі зміни у робочій директорії, які ще не готові до коміту. Просто введіть:

```
git stash
```

2. Збереження змін з повідомленням:

Ви можете додати повідомлення, щоб уточнити, що саме ви зберігаєте:

```
git stash save "Ваше повідомлення"
```

3. Застосування змін:

Якщо ви хочете повернутися до збережених змін пізніше, використайте команду `git stash apply` або `git stash pop`:

`git stash pop` також видаляє збережені зміни після їхнього застосування.

Ця команда застосовує останнє збереження до робочої директорії і видаляє його зі стеку.

4. Список збережених змін:

Ви можете переглянути список всіх збережених змін:

```
git stash list
```

5. Повернення до конкретного збереження:

Ви можете застосувати конкретне збереження, вказавши його індекс у списку:

```
git stash apply stash@{n}
```

Де `n` - це індекс збереження у списку.

6. Видалення збережених змін:

Якщо ви більше не потребуєте збережених змін, ви можете видалити їх:

```
git stash drop
```

Ця команда видаляє останнє збереження зі стеку, але не застосовує його до робочої директорії.

Якщо ви хочете видалити конкретне збереження, вкажіть його індекс:

```
git stash drop stash@{n}
```

`git stash` також використовується в робочих процесах вашої команди. Наприклад, залучіть її до процесу розгляду коду (code review), щоб розробники могли тимчасово приховувати зміни для уникнення розбитих (розділених) комітів.

Виправлення проблеми з розбитими комітами може бути частиною процесу роботи з Git. Розбитий коміт – це ситуація, коли один коміт містить зміни, які повинні були бути розділені на кілька окремих комітів для кращої структуризації історії комітів або для кращого розуміння змін.

Наприклад, у випадку, коли ви внесли кілька різних, але не пов'язаних змін у файл, і ви комітите ці зміни разом, це може створити розбитий коміт. Також, якщо ваші коміти містять зміни, які стосуються різних аспектів програмного забезпечення (наприклад, виправлення помилок та вдосконалення функціональності), розділення їх на окремі коміти може бути корисним для кращої зрозумілості історії комітів.

Вирішення проблеми з розбитими комітами може включати розбиття цих комітів на менші, більш узгоджені коміти, а також переписання історії комітів за допомогою команди `git rebase`, щоб виправити цю проблему.

---

2. git rebase

Відпрацьовується усіма учасниками проекту.

Команда `git rebase` використовується для перебазування (переміщення) комітів на іншу гілку або для перебазування комітів на нову базову точку. Це може бути корисно для реорганізації історії комітів або для інтеграції змін з однієї гілки на іншу. Під час використання команди `git rebase` для розробки програмного забезпечення зазвичай виконуються наступні кроки:

1. Перевірка поточної гілки:

Переконайтеся, що ви на гілці, яку потрібно перебазувати. Використовуйте команду `git status`, щоб перевірити поточний стан репозиторію.

2. Витягнення змін з віддаленої гілки (якщо потрібно):

Якщо ви працюєте з віддаленою гілкою, спочатку витягніть оновлення з віддаленого репозиторію за допомогою команди `git pull` або `git fetch`.

3. Застосування команди `git rebase`:

Застосуйте команду `git rebase` з необхідними параметрами. Наприклад, якщо ви хочете перебазувати поточну гілку на гілку `main`, виконайте команду:

```
git rebase main
```

4. Вирішення конфліктів (якщо вони виникли):

Під час перебазування можуть виникнути конфлікти. Вирішуйте ці конфлікти, відкриваючи відповідні файли, редагуючи їх, видаляючи конфліктні мітки та додаючи зміни за допомогою команд `git add` та `git rebase --continue`.

5. Продовження перебазування:

Після вирішення конфліктів продовжте перебазування за допомогою команди `git rebase --continue`.

6. Перевірка результатів:

Після завершення перебазування перевірте робочий стан репозиторію за допомогою команд `git status` та `git log`, щоб переконатися, що все пройшло успішно.

7. Тестування коду:

Переконайтеся, що ваш код працює як очікувалося після перебазування. Запустіть тести та перевірте правильність роботи програми.

8. Завершення перебазування:

Після успішного перебазування та перевірки коду закінчіть операцію, якщо все в порядку, за допомогою команди `git rebase --continue`.

9. Оновлення віддаленого репозиторію (за потреби):

Якщо ви працюєте з віддаленою гілкою, оновіть віддалений репозиторій за допомогою команди `git push`.

Це загальний порядок дій під час використання `git rebase` для роботи з розробкою програмного забезпечення. Деталі можуть варіюватися в залежності від конкретного сценарію роботи та ваших потреб.

3. git patch

Відпрацьовується усіма учасниками проекту.

Використання команди `git patch` для розробки програмного забезпечення може бути корисним у випадках, коли потрібно застосувати певні зміни з однієї гілки або коміту до іншої без використання повного механізму `git cherry-pick`.

Порядок дій під час використання команди `git patch`:

1. Створення патча:

- Створіть патч для змін, які ви хочете застосувати. Це можна зробити за допомогою команди `git diff` або `git format-patch`. Наприклад:

```
git diff <початковий коміт> <кінцевий коміт> > my_patch.patch
```

- Використовуйте команду `git diff` для створення патча, який містить зміни між двома комітами, гілками або робочим деревом. Наприклад:

```
git diff > my_changes.patch
```

2. Переключення на цільову гілку:

- Переключіться на гілку, до якої ви хочете застосувати зміни за допомогою команди `git checkout`:

```
git checkout <цільова гілка>
```

3. Застосування патча:

- Застосуйте патч до вашої цільової гілки за допомогою команди `git apply`:

```
git apply my_patch.patch
```

- Якщо патч створений за допомогою `git diff --cached`, можна використати команду `git apply` з параметром `--cached`, щоб застосувати зміни до індексу, а не до робочої гілки.

- Якщо патч створений за допомогою `git format-patch`, його можна застосувати за допомогою команди `git am`. Це корисно, коли патч містить інформацію про автора та іншу мета-інформацію. Наприклад:

```
git am my_changes.patch
```

4. Коміт змін:

- Закомітьте застосовані зміни до цільової гілки за допомогою команди `git commit`:

```
git commit -m "Застосування змін з патча"
```

5. Перевірка результатів:

- Переконайтеся, що зміни були правильно застосовані та всі тести пройшли успішно.

6. Оновлення віддаленого репозиторію (за потреби):

- Якщо ви працюєте з віддаленою гілкою, оновіть віддалений репозиторій за допомогою команди `git push`.

7. Кінець роботи з патчем:

- Після застосування змін та оновлення віддаленого репозиторію звільніть ресурси, пов'язані з патчем.

Обов'язково враховуйте потреби вашого конкретного проекту та команди, а також слідкуйте за найкращими практиками роботи з Git.

---

4. git cherry-pick

Відпрацьовується усіма учасниками проекту.

Використання команди `git cherry-pick` дозволяє застосувати окремий коміт з однієї гілки до поточної гілки. Це може бути корисно, коли потрібно взяти лише певні зміни з однієї гілки та застосувати їх до іншої.

Порядок дій під час використання `git cherry-pick`:

1. Вибір цільової гілки:

- Переконайтеся, що ви перебуваєте на гілці, до якої ви хочете застосувати зміни. Використовуйте команду `git checkout` для переключення на цільову гілку:

```
git checkout <цільова гілка>
```

2. Вибір коміту для cherry-pick:

- Визначте ідентифікатор (хеш) коміту, який ви хочете взяти. Цей ідентифікатор можна знайти, переглянувши історію комітів за допомогою `git log`.

3. Виконання cherry-pick:

- Застосуйте команду `git cherry-pick` з ідентифікатором цільового коміту:

```
git cherry-pick <ідентифікатор коміту>
```

4. Вирішення конфліктів (якщо потрібно):

- Якщо під час cherry-pick виникають конфлікти, вирішіть їх, редагуючи файли, що містять конфлікти, та застосовуючи їх за допомогою `git add`. Після цього продовжте cherry-pick за допомогою команди `git cherry-pick --continue`.

5. Перевірка результатів:

- Після cherry-pick перевірте стан репозиторію за допомогою `git status` та переконайтеся, що зміни були застосовані коректно.

6. Тестування коду:

- Переконайтеся, що ваш код працює як очікувалося після застосування змін.

7. Оновлення віддаленого репозиторію (за потреби):

- Якщо ви працюєте з віддаленою гілкою, оновіть віддалений репозиторій за допомогою команди `git push`.

8. Кінець роботи з cherry-pick:

- Після успішного застосування змін та оновлення віддаленого репозиторію звільніть ресурси, пов'язані з cherry-pick.

5. `git commit --amend`

Відпрацьовується усіма учасниками проекту.

Загальний порядок дій під час використання команди `git commit` в процесі розробки програмного забезпечення:

1. Додавання змін:

- Внесіть зміни в код програмного забезпечення або/і документацію.

2. Перевірка стану змін:

- Використовуйте команду `git status`, щоб перевірити, які файли були змінені та які з них готові до коміту.

3. Додавання файлів до індексу:

- Використовуйте команду `git add`, щоб додати змінені файли до індексу. Наприклад:

```
git add файл_змінений.txt
```

4. Коміт змін:

- Використовуйте команду `git commit`, щоб зберегти зміни в репозиторії. Під час коміту можна додати опис змін, використовуючи флаг `-m`, або відкриється текстовий редактор для введення повідомлення про коміт.

```
git commit -m "Опис змін"
```

5. Перевірка результатів:

- Переконайтеся, що коміт пройшов успішно та всі зміни зафіксовані.

6. Оновлення віддаленого репозиторію (за потреби):

- Якщо ви працюєте з віддаленою гілкою, оновіть віддалений репозиторій за допомогою команди `git push`.

Команда `git commit --amend` використовується для внесення змін до останнього коміту.

Загальний порядок дій під час використання команди `git commit --amend`:

1. Внесення змін:

- Внесіть необхідні зміни в код програмного забезпечення або/і документацію.

2. Додавання змін до індексу:

- Використовуйте команду `git add`, щоб додати змінені файли до індексу.

```
git add змінений_файл.txt
```

3. Внесення змін до останнього коміту:

- Використовуйте команду `git commit --amend`, щоб внести зміни до останнього коміту.

```
git commit --amend
```

Після виконання цієї команди відкриється текстовий редактор або командний рядок для введення нового повідомлення про коміт. Збережіть зміни та закрийте редактор.

4. Перевірка результатів:

- Переконайтеся, що коміт був змінений і нові зміни включені.

5. Оновлення віддаленого репозиторію (за потреби):

- Якщо ви працюєте з віддаленою гілкою, оновіть віддалений репозиторій за допомогою команди `git push`.