GIT

Tags

GIT — это распределенная система контроля версий и система управления исходным кодом (SCM), предназначенная для быстрой и эффективной обработки небольших и крупных проектов.

▼ Что такое репозиторий в GIT?

Репозиторий содержит каталог с именем .git, где git хранит все свои метаданные репозитория. Содержимое каталога .git является личным для git.

▼ Каковы преимущества использования GIT?

а) Избыточность и репликация данных

б) Высокая доступность

в) Только один каталог .git для каждого репозитория.

г) Превосходное использование диска и производительность сети.

д) Дружественное сотрудничество

e) Любые проекты могут использовать GIT.

▼ Как вы узнаете в GIT, что ветка уже объединена с основной?

```
git checkout main
git fetch origin
git branch --merged
```

▼ Какова функция git clone?

Команда git clone создает копию существующего репозитория Git. Чтобы получить копию центрального репозитория, программисты чаще всего используют «клонирование».

▼ Что такое ветвление (branching) в Git и для чего оно используется.

Ветвление в Git позволяет создавать независимые линии разработки внутри одного репозитория. Это полезно для работы над новыми функциями, исправлениями ошибок или экспериментами без риска повредить основную (стабильную) кодовую базу. Ветви можно объединять (сливать) обратно, интегрируя изменения в основную ветвь.

▼ Как создать новую ветку и переключиться на нее?

Создать новую ветку и сразу переключиться на нее можно с помощью команды:

```
git checkout -b имя_ветки
```

Эта команда сочетает в себе создание ветки (`git branch имя_ветки`) и переключение на нее (`git checkout имя_ветки`).

▼ Как объединить две ветки в Git?

Чтобы объединить ветку `feature` в текущую ветку (например, `main`), выполните следующие команды:

```
git checkout main          # Переключиться на ветку main
git merge feature          # Объединить ветку feature с mai
n
```

При слиянии Git автоматически объединит изменения. Если есть конфликты, их нужно разрешить вручную.

▼ Что такое `git stash` и когда его используют?

`git stash` временно сохраняет незакоммиченные изменения в рабочей директории, очищая ее до последнего коммита. Это полезно, когда нужно переключиться на другую ветку или выполнить срочную задачу, не теряя текущих изменений.

Основные команды:

- Сохранить изменения: `git stash`
- Просмотреть список сохраненных изменений: `git stash list`
- Восстановить последние сохраненные изменения: `git stash apply` или `git stash pop` (последнее удаляет изменения из стэка)

▼ Объясните разницу между `git fetch` и `git pull`.

- `git fetch` загружает изменения из удаленного репозитория в локальный, но не объединяет их с текущей веткой. Это позволяет просмотреть изменения перед слиянием.
- `git pull` выполняет `git fetch`, а затем автоматически сливает (merge) изменения с текущей веткой. Это обновляет локальную ветку до состояния удаленной.

▼ Как отменить последний коммит без удаления изменений в коде?

Чтобы отменить последний коммит, сохранив изменения в рабочей директории:

```
git reset --soft HEAD~1
```

Опция `--soft` перемещает указатель ветки на предыдущий коммит, оставляя изменения в индексе (staging area).

▼ Что такое `git rebase` и когда его следует использовать?

Ответ:

`git rebase` — это команда для переноса серии коммитов на новую базу. Она перезаписывает историю коммитов, делая ее более линейной. Используется для

интеграции изменений из одной ветки в другую без создания дополнительного коммита слияния.

Пример:

```
git checkout feature
git rebase main
```

Важно: Не следует использовать `git rebase` на ветках, которые уже опубликованы и доступны другим разработчикам, так как это изменяет историю и может привести к конфликтам.

Пример использования

Сценарий

Представьте следующую историю веток:

```
A --- B --- C  [main]
       \
        D --- E --- F  [feature]
```

- `main`: содержит коммиты `A`, `B`, `C`.
- `feature`: ответвилась от `B` и содержит коммиты `D`, `E`, `F`.

Задача

Обновить ветку `feature` последними изменениями из `main`, чтобы она выглядела так, как будто была создана от коммита `C`.

Шаги

1. **Переключитесь на ветку `feature`:**

   ```
   git checkout feature
   ```

2. **Выполните `git rebase`:**

```
git rebase main
```

Что происходит?

- Git перенесет коммиты `D`, `E`, `F` и применит их поверх `C`.
- История ветки `feature` будет выглядеть так:

```
A --- B --- C  [main]
              \
               D' --- E' --- F'  [feature]
```

- Обратите внимание, что хэши коммитов изменились (`D'`, `E'`, `F'`), так как они были переписаны.

▼ Как разрешать конфликты при слиянии веток?

Ответ:

При слиянии веток могут возникать конфликты, если изменения затрагивают одни и те же строки кода. Для разрешения конфликтов:

1. Выполните слияние:

```
git merge feature
```

2. Git сообщит о конфликтующих файлах. Откройте каждый из них и вручную разрешите конфликты, выбрав нужные изменения и удалив маркеры конфликтов (`<<<<<<<`, `=======`, `>>>>>>>`).

3. После разрешения конфликтов добавьте исправленные файлы в индекс:

```
git add конфликтующий_файл
```

4. Завершите слияние коммитом:

```
git commit
```

▼ Что такое `git cherry-pick` и как его использовать?

Ответ:

`git cherry-pick` позволяет выбрать конкретный коммит из одной ветки и применить его в текущую ветку. Это полезно, когда нужно перенести отдельные изменения без слияния всей ветки.

Пример:

```
git checkout main
git cherry-pick commit_hash
```

▼ Как отменить изменения в файле и вернуть его к состоянию последнего коммита?

Ответ:

Чтобы отменить незакоммиченные изменения в файле:

```
git checkout -- имя_файла
```

▼ Как посмотреть историю коммитов в Git?

Ответ:

Используйте команду:

```
git log
```

▼ Как просмотреть изменения в файлах перед коммитом?

Ответ:

Используйте команду:

- Для незакоммиченных изменений: `git diff`
- Для проиндексированных изменений: `git diff --stage`
