GIT

История коммитов

Сохранение истории изменений или история коммитов - одна из самых важных частей git. В истории сохраняются все коммиты, по которым можно посмотреть автора коммита, commit message, дату коммита и его хэш. А также можно увидеть измененные файлы и изменения в каждом файле. То есть git хранит буквально все, от самого начала проекта.

Команда git log

За просмотр истории коммитов отвечает команда git log. В сочетании с различными параметрами эта команда выводит историю по-разному. Есть много различных вариантов и комбинаций параметров, посмотрим некоторые из них

git log, просмотр истории по умолчанию

```
$ git log
```

Copy

Показывает все коммиты от новых к старым. Для каждого коммита выводится

- хэш
- автор
- дата
- сообщение (commit message)

git log -p, расширенный вывод истории

```
$ git log -p
```

Copy

Выводит то же, что и git log, но еще и с изменениями в файлах

git log --oneline, короткая запись

```
$ git log --oneline
```

Copy

Вывод коммитов в одну строку. Показывает только хэш коммита и commit message

git log --stat --graph, история в виде дерева

```
$ git log --stat --graph
```

Copy

Выводит коммиты в виде дерева, в командной строке псевдографикой. Плюс выводит список измененных файлов. К дереву коммитов мы вернемся, когда

будем работать с ветками.

Сортировка и фильтрация истории

Есть множество команд, которые позволяют сортировать и фильтровать историю коммитов в командной строке. В том числе в сочетании с линуксовыми командами. Рассмотрим некоторые из них

Поиск по коммитам

Команда grep - мощный инструмент, который помогает работать в том числе и с git. Например, искать по коммитам

```
    git log --oneline | grep revert # поиск упоминания revert
    git log --oneline | grep -i revert # независимо от регист
pa
```

Copy

Коммиты, затронувшие один файл

```
    git log index.html
```

Copy

Поиск по автору

```
    git log --author webdev
```

Copy

В опции --author можно указать имя или email, необязательно целиком, можно только часть.

Поиск по диапазону дат

Опции --after и --before задают начальную и конечную даты коммитов

```
     git log --after='2020-03-09 15:30' --before='2020-03-09
16:00'
```

Copy

Комбинация команд и опций

Команды и опции git можно комбинировать и дополнять их линуксовыми командами

```
    git log --author=webdev --oneline | grep footer # все ком
миты от автора, в которых упоминается footer
    git log --oneline | wc -l # количество коммитов
```

Copy

Какие еще есть варианты

Мы рассмотрели базовые примеры, но в документации по git log есть много различных опций. Все их рассматривать нет смысла, при необходимости изучайте документацию.

```
    git log --help
```

Copy

Просмотр отдельного коммита, git show

Чтобы просмотреть отдельный коммит, нужно узнать его хэш. Хэш коммита выводится в любой команде git log, с параметрами или без. Например,

```
$ git log --oneline

7b7d7fa Fixed footer
26812f9 Revert "Fixed footer"
0f90ae7 Revert "Fixed styles"
...
a1f3c45 Added footer
a65aa43 Added new block students to main page
0b90433 Initial commit
```

Copy

Смотрим второй коммит

```
$ git show 43f6afc
```

Copy

Выводится подробная информация о коммите:

- хэш
- автор
- дата
- commit message
- список измененных файлов
- изменения в каждом файле

Короткий хэш коммита

Хэш коммита 40-символьный, но можно использовать короткую запись - первые 7 символов хэша. Команда git log --oneline выводит именно короткий хэш. Для других операций с коммитами достаточно использовать первые 4

символа. Например, 3 команды ниже покажут содержимое одного и того же коммита

```
$ git show 051f75475cb1dca3cd08c1c7367a3308671ccf7b
$ git show 051f754
$ git show 051f
```

Copy

История коммитов в PhpStorm

В окне Local Changes, на вкладке Log показывается вся история коммитов, в левой половине вкладки. В списке коммитов показываются их commit message, автор и дата. Клик на нужный коммит откроет в правой части вкладки список измененных файлов. Клик на нужном файле и Ctrl/Cmd+D покажет все изменения в этом файле точно так же, как и git diff.

В тексте объяснить работу с историей в PhpStorm сложно, смотрите видеоурок.

Переключение на старый коммит, зачем это нужно

Нужно это обычно в двух случаях:

1. При неудачном деплое, когда вскрылась критичная бага. Если бага сложная и пофиксить ее быстро не удается, можно откатиться на рабочий коммит, задеплоить рабочую версию и уже потом чинить багу.

2. При отладке. Когда в код закралась бага и мы постепенно продвигаемся "назад в прошлое" и ищем, в какой момент что-то сломалось

Как переключиться на коммит в терминале

Первое - узнать хэш нужного коммита. Например, имеем такую историю

```
$ git log --oneline
```

```
7b7d7fa Fixed footer
26812f9 Revert "Fixed footer"
0f90ae7 Revert "Fixed styles"
...
a1f3c45 Added footer
a65aa43 Added new block students to main page
0b90433 Initial commit
```

Copy

Хотим переключиться на предпоследний коммит. Коммиты идут в порядке убывания, поэтому нам нужен второй сверху - 26812f9. Переключаемся на него командой

```
$ git checkout 26812f9
```

Copy

Все, вернулись в прошлое. Проверим историю, теперь коммит, на который мы переключились - последний

```
$ git log --oneline

26812f9 Revert "Fixed footer"
0f90ae7 Revert "Fixed styles"
...
a1f3c45 Added footer
a65aa43 Added new block students to main page
0b90433 Initial commit
```

Copy

Уйдем еще дальше, переключимся на первый коммит. Так как коммиты упорядочиваются по убыванию даты, то первый коммит - это последний в списке - 0b90433 Initial commit

```
$ git checkout 0b90433
```

Copy

Проверяем историю

```
$ git log --oneline

0b90433 Initial commit
```

Copy

Чтобы вернуться обрано, в исходное состояние, нужно набрать

```
$ git checkout master
```

Copy

master - это ветка, в которой мы работаем по умолчанию. О ветках поговорим через пару уроков

Добавление и коммит нескольких файлов

Обычно в Git мы используем команду

```
git add *
```

, чтобы подготовить все изменённые файлы для последующего коммита. Чтобы закоммитить эти изменения, используется команда

```
git commit -m "commitMessage"
```

. Однако существует более удобная команда, которая выполняет обе задачи за один шаг:

```
git commit -am "commitMessage"
```

Флаг

```
-am
```

позволяет не только подготовить эти изменения, но и закоммитить их одной эффективной операцией.

Создание и переключение на ветку Git

Как и в предыдущем случае, есть команда, которая объединяет функциональность этих двух команд. Вместо того, чтобы использовать две отдельные команды —

```
git branch branchName
```

для создания ветки и

```
git checkout branchName
```

для переключения на неё — обе задачи можно выполнить за один шаг с помощью следующей команды:

```
git checkout -b branchName
```

Флаг

```
-b
```

в команде

```
git checkout
```

позволяет не только создать новую ветку, но и сразу же переключить вас на неё.

Удаление Git-ветки

Чтобы удалить ветку в Git, можно использовать команду

```
git branch -d
```

или

```
git branch -D
```

. Опция

```
-d
```

предназначена для безопасного удаления: удаление ветки произойдёт только в том случае, если было проведено слияние (merge) с текущей веткой. Опция

```
-D
```

предназначена для принудительного удаления, при котором ветка будет удалена независимо от того, полностью она слилась или нет. Вот команды:

Безопасное удаление (проверяет слияние):

```
git branch -d branchName
```

Принудительное удаление (не проверяет слияние):

```
git branch -D branchName
```

Переименование Git-ветки

Чтобы переименовать ветку, можно использовать команду

```
git branch -m
```

, за которой следует текущее имя ветки и новое желаемое имя. Например:

```
git branch -m oldBranch newBranch
```

Однако если вы хотите переименовать текущую ветку, в которой вы сейчас работаете, без явного указания текущего имени, вы можете использовать следующую команду:

```
git branch -m newBranchName
```

Здесь вам не нужно указывать старое имя ветки, потому что Git предположит, что вы хотите переименовать текущую ветку.

Удаление определённого файла

Когда-нибудь вам понадобится удалить определённый файл из области подготовки, что позволит внести дополнительные изменения перед коммитом. Для этого используйте команду:

```
git reset filename
```

Это удалит файл из области подготовки файлов, сохранив при этом изменения.

Отмена изменений в определённом файле

Если вам нужно полностью отменить изменения, внесённые в определённый файл, и вернуть его к последнему закомиченному состоянию, используйте команду:

```
git checkout -- filename
```

Эта команда гарантирует, что файл вернётся к своему предыдущему состоянию с отменой всех последних изменений. Это полезный способ

начать работу над конкретным файлом заново, не затрагивая остальные изменения.

Обновление последнего коммита

Представьте, что вы только что сделали коммит, но потом поняли, что забыли включить в него одно изменение, или, возможно, вы хотите исправить комментарий к коммиту. При этом создавать новый коммит для этого небольшого изменения вы не хотите. Вместо этого вы хотите добавить его в предыдущий коммит. В таком случае можно использовать команду:

```
git commit --amend -m 'message'
```

Эта команда изменяет последний коммит. Это объединяет все новые изменения в области подготовки (добавленные с помощью

```
git add
```

) с новым комментарием, чтобы создать обновлённый коммит.

Следует помнить, что если вы уже отправили коммит в удалённый репозиторий, то для обновления удалённой ветки вам нужно будет принудительно отправить изменения с помощью

```
git push --force
```

. Потому что стандартная команда

```
git push
```

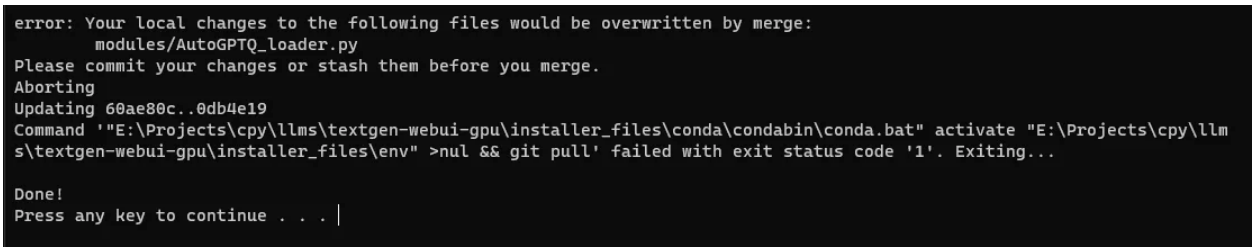
добавляет новый коммит в ваш удалённый репозиторий, а не изменяет последний коммит.

Хранение отложенных изменений (stashing)

Представьте, что вы работаете над двумя разными ветками, A и B. Во время внесения изменений в ветку A ваша команда просит вас исправить баг в ветке B. Когда вы пытаетесь переключиться на ветку B с помощью

```
git checkout B
```

, Git не дает этого сделать, выдавая ошибку:



Невозможно переключить ветку

Можно закоммитить изменения, как предлагает сообщение об ошибке. Но это скорее фиксированная точка во времени, а не постоянная работа. Для хранения отложенных изменений используем команду

```
git stash
```

, которая приведёт к тому, что отложенные изменения будут сохранены.

```
git stash
```

временно сохраняет изменения, которые вы не готовы закоммитить, позволяя вам переключать ветки или работать над другими задачами, не делая коммита незаконченной работы.

Чтобы повторно применить записанные изменения в ветке, можно использовать

```
git stash apply
```

или

```
git stash pop
```

. Обе команды восстанавливают последние отложенные изменения. Команда

```
stash apply
```

просто восстанавливает изменения, а pop восстанавливает изменения и удаляет их из хранилища (stash). Подробнее о хранении отложенных изменений можно прочитать

здесь

.

Откат изменений в коммитах

Представьте, что вы работаете над Git-проектом и обнаружили, что в определённый коммит были внесены некоторые нежелательные изменения. Вам нужно отменить эти изменения, не стирая коммит из истории. Чтобы отменить этот конкретный коммит, можно использовать следующую команду:

```
git revert commitHash
```

Это безопасный и недеструктивный способ исправить ошибки или нежелательные изменения в проекте.

Допустим, у вас есть серия коммитов:

- Коммит A
- Коммит B (здесь внесены нежелательные изменения)
- Коммит C
- Коммит D

Чтобы отменить последствия коммита B, нужно выполнить команду:

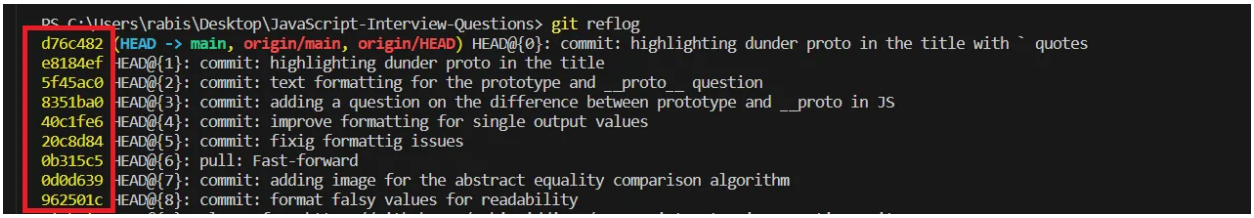
```
git revert commitHashOfB
```

Git создаст новый коммит, назовём его коммит E, который отменяет изменения, внесённые коммитом B. Коммит E станет последним коммитом в вашей ветке, и проект теперь отражает состояние, в котором он находился бы, если бы коммита B не было.

Если вам интересно, как получить хэш коммита, то это можно сделать с помощью команды

```
git reflog
```

. На скриншоте ниже выделенные части представляют собой хэши коммитов, которые можно легко скопировать:



Хэши коммитов

Сброс коммитов

Предположим, вы сделали коммит в своём проекте. Однако после проверки вы понимаете, что вам нужно скорректировать или полностью отменить последний коммит. Для таких случаев Git предоставляет следующие команды:

Soft reset

```
git reset --soft HEAD^
```

Эта команда позволяет вернуться к последнему коммиту, сохранив все изменения в области подготовки. Проще говоря, с помощью этой команды можно легко отменить коммит, сохранив изменения в коде. Это удобно, когда вам нужно пересмотреть последний коммит, возможно, добавить больше изменений перед повторным коммитом.

Mixed reset

```
git reset --mixed HEAD^
```

Это поведение по умолчанию, когда вы используете

```
git reset HEAD^
```

без указания

```
--soft
```

или

```
--hard
```

. Он сбрасывает последний коммит и удаляет изменения из области подготовки. Однако эти изменения остаются в рабочей директории. Это полезно, когда вы хотите отменить последий коммит и внести изменения с нуля, сохранив изменения в рабочей директории перед повторным коммитом.

Hard reset

```
git reset --hard HEAD^
```

Напоследок давайте поговорим о команде

```
git reset --hard HEAD^
```

. Она полностью стирает последний коммит вместе со всеми связанными с ним изменениями из вашей истории Git. Когда вы используете флаг

```
--hard
```

, пути назад уже не будет. Поэтому используйте его с особой осторожностью, только если хотите навсегда удалить последний коммит и все его изменения.

Задание

Задание: Работа с ветками и разрешение конфликтов

Цель:

Закрепить навыки работы с ветками, разрешения конфликтов, слияния изменений и анализа истории коммитов.

Описание:

1. **Создайте репозиторий:**
   - Инициализируйте новый репозиторий Git на своем компьютере.
   - Создайте файл `README.md` и добавьте в него краткое описание проекта. Сделайте первый коммит.
2. **Создайте ветку для новой функции:**
   - Создайте новую ветку `feature/update-readme`.
   - Внесите изменения в файл `README.md`, добавив список целей проекта. Сделайте коммит.
3. **Создайте другую ветку для исправления:**
   - Вернитесь в ветку `main`.
   - Создайте ветку `fix/typos`.
   - Внесите изменения в файл `README.md`, исправив орфографическую ошибку или добавив пример использования проекта. Сделайте коммит.
4. **Имитируйте конфликт:**
   - Вернитесь в ветку `main`.
   - Добавьте и закоммитьте новую строку в файл `README.md`.

5. **Слияние изменений:**
    - Слейте ветку `feature/update-readme` в `main`. Разрешите конфликт, если он возникнет.
    - После успешного слияния слейте ветку `fix/typos` в `main`.
6. **Просмотр истории коммитов:**
    - Выполните команду `git log` и проанализируйте историю изменений.
    - Дополнительно используйте `git log --oneline --graph --all`, чтобы наглядно увидеть ветки и слияния.
7. **Удаление веток:**
    - После успешного слияния удалите ветки `feature/update-readme` и `fix/typos`.

Результат:

После выполнения задания у вас должен быть репозиторий с веткой `main`, где объединены все изменения, и файл `README.md` обновлен с учетом всех изменений и разрешенных конфликтов.

Проверка:

- История коммитов, просмотренная с помощью `git log`, должна показывать правильный порядок действий и слияния.
- Финальный файл `README.md` должен содержать все изменения.