Навигация по коммитам

Хеш — идентификатор коммита

Информация о коммите — это набор данных: когда был сделан коммит, содержимое файлов в репозитории на момент коммита и ссылка на предыдущий, или родительский (англ. parent), коммит.

Git хеширует (преобразует) информацию о коммите с помощью алгоритма SHA-1 (от англ. Secure Hash Algorithm — «безопасный алгоритм хеширования») и получает для каждого коммита свой уникальный хеш — результат хеширования.

Обычно хеш — это короткая (40 символов в случае SHA-1) строка, которая состоит из цифр 0—9 и латинских букв A—F (неважно, заглавных или строчных). Она обладает следующими важными свойствами:

* если хеш получить дважды для одного и того же набора входных данных, то результат будет гарантированно одинаковый;
* если хоть что-то в исходных данных поменяется (хотя бы один символ), то хеш тоже изменится (причём сильно).

Чтобы убедиться в этом, можно поэкспериментировать с SHA-1 [на этом сайте](https://emn178.github.io/online-tools/sha1.html) (https://emn178.github.io/online-tools/sha1.html) — попробуйте ввести в поле input (англ. «ввод») разные символы, слова или предложения и понаблюдайте, как меняется хеш в поле output (англ. «вывод»).

**Хеш — основной идентификатор коммита**

Git хранит таблицу соответствий хеш → информация о коммите. Если вы знаете хеш, вы можете узнать всё остальное: автора и дату коммита и содержимое закоммиченных файлов. Можно сказать, что хеш — основной идентификатор коммита.

При работе с Git хеши будут встречаться вам регулярно. Их можно будет передавать в качестве параметра разным Git-командам, чтобы указать, с каким коммитом нужно произвести то или иное действие.

Все хеши и таблицу хеш → информация о коммите Git сохраняет в служебные файлы. Они находятся в скрытой папке .git в репозитории проекта.

**Исследуем лог**

Лог содержит описание коммита: хеш, автор, дата, сообщение.

Сокращенный лог помогает быстро найти нужный коммит среди множества.

Команда git log --oneline выводит сокращенный лог с хешами и комментариями.

Уникальная длина сокращенных хешей помогает идентифицировать коммит.

Сокращённый хеш (то есть первые несколько символов полного) можно использовать точно так же, как и полный. Для этого команда git log --oneline автоматически подбирает такую длину сокращённых хешей, чтобы они были уникальными в пределах репозитория и Git всегда мог понять, о каком коммите идёт речь.

Если выход из просмотра логов не произошёл автоматически, нажмите клавишу Q (от англ. ***Q***uit — «выйти») в английской раскладке клавиатуры.

Что означает HEAD

При вызове команды git log вы также могли заметить надпись (HEAD -> master) после хеша последнего коммита.

Файл *HEAD* (англ. «голова», «головной») — один из служебных файлов папки *.git.* Он указывает на коммит, который сделан последним (то есть на самый новый).

В этом можно убедиться с помощью терминала. Перейдите в папку .*git* командой *cd*. Посмотрите содержимое файла *HEAD* командой *cat*.

*$ pwd* # посмотрели, где мы

*$ cd .git*/ # перешли в папку .git

*$ ls* # посмотрели, какие есть файлы

*$ cat HEAD* # команда cat показывает содержимое файла

*ref: refs/heads/master* # в файле вот такая ссылка

Внутри HEAD — ссылка на служебный файл: refs/heads/master (или refs/heads/main в зависимости от названия ветки). Если заглянуть в этот файл, можно увидеть хеш последнего коммита.

**Статусы файлов в Git**

До появления Git системы контроля версий выделяли только два статуса у файлов: «уже закоммичен» и «ещё не закоммичен». Например, в Subversion (самой популярной VCS до эпохи Git) не нужно было выполнять команду — аналог git add, а можно было просто сделать коммит (svn commit). Эта команда по умолчанию добавляла в коммит все новые и изменённые файлы.

Git даёт больше контроля за состоянием файлов.

**Статусы untracked/tracked, staged и modified**

Одна из ключевых задач Git — отслеживать изменения файлов в репозитории. Для этого каждый файл помечается каким-либо статусом.

* **untracked** (англ. «неотслеживаемый»)   
  Git «видит», что такой файл существует, но не следит за изменениями в нём. У untracked-файла нет предыдущих версий, зафиксированных в коммитах или через команду git add.
* **staged** (англ. «подготовленный»)

После выполнения команды git add файл попадает в **staging area** (от англ. *stage* — «сцена», «этап [процесса]» и *area* — «область»), то есть в список файлов, которые войдут в коммит. В этот момент файл находится в состоянии staged.

**Staging area, index и cache**

Staging area также называют **index** (англ. «каталог») или **cache** (англ. «кеш»), а состояние файла staged иногда называют indexed или cached.

Все три варианта могут встречаться в документации и в качестве флагов команд Git. А также в интернете — например, в вопросах и ответах [на сайте Stack Overflow](https://stackoverflow.com/).

* **tracked** (англ. «отслеживаемый»)   
  Состояние tracked — это противоположность untracked. Оно довольно широкое по смыслу: в него попадают файлы, которые уже были зафиксированы с помощью git commit, а также файлы, которые были добавлены в staging area командой git add. То есть все файлы, в которых Git так или иначе отслеживает изменения.
* **modified** (англ. «изменённый»)   
  Состояние modified означает, что Git сравнил содержимое файла с последней сохранённой версией и нашёл отличия. Например, файл был закоммичен и после этого изменён.

Для файлов в состояниях staged и modified обычно не указывают, что они также tracked, потому что это состояние подразумевается.

**Про staged и modified**

Команда git add добавляет в staging area только текущее содержимое файла. Если вы, например, сделаете git add file.txt, а затем измените file.txt, то новое содержимое файла не будет находиться в staging.

Git сообщит об этом с помощью статуса modified: файл изменён относительно той версии, которая уже в staging. Чтобы добавить в staging последнюю версию, нужно выполнить git add file.txt ещё раз.

**Типичный жизненный цикл файла в Git**

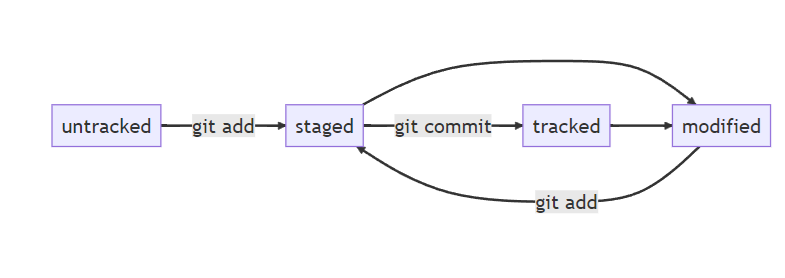
У большинства файлов вполне предсказуемый путь.

1. Файл только что создали. Git ещё не отслеживает содержимое этого файла. Состояние: untracked.
2. Файл добавили в staging area с помощью git add. Состояние: staged (+ tracked).
   * Возможно, изменили файл ещё раз. Состояния: staged, modified (+ tracked). Обратите внимание: staged и modified у одного файла, но у разных его версий.
   * Ещё раз выполнили git add. Состояние: staged (+ tracked).
3. Сделали коммит с помощью git commit. Состояние: tracked.
4. Изменили файл. Состояние: modified (+ tracked).
5. Снова добавили в staging area с помощью git add. Состояния: staged (+ tracked).
6. Сделали коммит. Состояния: tracked.
7. Повторили пункты 4−7 много-много раз.

Изобразим жизненный цикл проекта на схеме mermaid в маркдауне.

* Блоки кода в маркдауне начинаются и заканчиваются тремя символами ```. После первых трёх ``` можно указать, какой именно код будет внутри блока. Например: ```mermaid , ```bash, ```python, ```javascript и так далее. Если ничего не указать, GitHub будет считать весь код простым текстом. Перед блоком и после него нужны пустые строки, иначе GitHub может не понять, что это блок кода.
* Два символа %% обозначают в mermaid строку-комментарий.
* Чтобы сделать схему, нужно указать формат, например, graph LR. **Graph** — это простейший тип схем.
* Чтобы добавить элементы и связи (стрелки), используют строки вида A --> B. Такая строка создаст квадратные блоки А и B и соединит их стрелкой.   
  Дополнительно можно указывать текст на стрелке. Например, так: A -- "text" --> B.

Для построения такой [схемы](https://github.blog/2022-02-14-include-diagrams-markdown-files-mermaid/):



требуется в файле Readme.md напечатать:

```mermaid

graph LR

untracked -- "git add" --> staged;

modified -- "git add" --> staged;

staged --> modified;

staged -- "git commit" --> tracked;

tracked --> modified;

```

Как читать git status

Частая ошибка при использовании Git — закоммитить лишнее или, наоборот, забыть добавить важный файл в коммит. Этого легко избежать, если не забывать проверять статусы файлов с помощью команды git status.

**Какие состояния показывает git status**

Большинство файлов в типичном проекте будут находиться в состоянии tracked (то есть закоммичены и не изменены после коммита). Вы не увидите это состояние в выводе команды git status — иначе она бы каждый раз выводила список вообще всех файлов проекта.

В итоге git status показывает только следующие состояния файлов:

* staged (Changes to be committed в выводе git status);
* modified (Changes not staged for commit);
* untracked (Untracked files).

4 примера состояний, в которых может находиться репозиторий

1. **Нет ни staged-, ни modified-, ни untracked-файлов.**

Если ничего не менять в репозитории после первого коммита, то в нём не должно быть ни изменённых файлов (modified), ни новых (untracked), ни добавленных в список на коммит (staged). Вызовите команду git status. Её вывод будет примерно таким.

$ git status

On branch master

nothing to commit, working tree clean

Это означает, что в репозитории нет новых или изменённых файлов. Последняя строка nothing to commit, working tree clean буквально переводится как «нечего коммитить, рабочая директория чиста».

Первая строка On branch master сообщает, что текущая ветка — master.

1. **Найдены неотслеживаемые файлы.**

$ git status

On branch master

Untracked files: *# найдены неотслеживаемые файлы*

(use "git add <file>..." to include in what will be committed)

fileA.txt

nothing added to commit but untracked files present (use "git add" to track)

Файл fileA.txt отображается в секции неотслеживаемых файлов — Untracked files. Это значит, что он не был добавлен в репозиторий через git add.

Обратите внимание: в самом выводе git status есть подсказка, какую команду использовать, чтобы добавить файл в список на коммит: **Use** git add <file> **to include in what will be committed** (англ. «используйте git add <file>, чтобы добавить в список на коммит»).

Добавьте fileA.txt в staging area с помощью git add и снова запросите git status.

$ git add fileA.txt

$ git status

On branch master

Changes to be committed: *# новая секция*

(use "git restore --staged <file>..." to unstage)

new file: fileA.txt

Теперь fileA.txt находится в секции Changes to be committed (англ. «изменения, которые попадут в коммит»). Если сейчас выполнить коммит, то в репозитории будет зафиксирована текущая версия этого файла.

1. **Найдены изменения, которые не войдут в коммит**

$ git status

On branch master

Changes not staged for commit: *# ещё одна секция*

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: fileA.txt

Файл fileA.txt был изменён, но ещё не добавлен в staging area после этого. Так он оказался в секции Changes not staged for commit (англ. «изменения, которые не подготовлены к коммиту»). Эта секция соответствует статусу modified.

Когда совсем новый файл попадает в staging area, перед его названием указывается new file. Вот так: new file: fileA.txt.

Если файл уже однажды попадал в историю (с помощью коммита) и был изменён, после выполнения git add он будет записан уже так: modified: fileA.txt.

1. **Файл добавлен в staging area, но после этого изменён**

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: fileA.txt

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: fileA.txt

Файл попал и в staged (Changes to be committed), и в modified (Changes not staged for commit). В staging area находится версия файла, которая была подготовлена к коммиту, а в Changes not staged for commit — уже изменённая версия.

Чтобы закоммитить самую свежую версию файла, нужно снова выполнить git add перед коммитом.