GeekBrains

Git. Базовый курс

Урок 5

История Git-проекта

Просмотр истории проекта

Хэш-сумма коммита

Поиск по дате

Поиск коммита по строке и автору

Игнорирование

Пустые папк

Используемые источники

© geekbrains.ru

Просмотр истории проекта

При создании коммит Git запоминает текущее состояние проекта. Посмотреть историю изменений можно при помощи команды **git log**:

```
$ git log
commit 9992e6df2da3270347f77bb8f9ed6d76dd34aedb (HEAD -> master)
Author: Igor Simdyanov <igorsimdyanov@gmail.com>
Date:   Sun Sep 8 22:06:46 2019 +0300

      И з м е н е н и я  в README-ф а й л е

commit e0c800c0646d885a64385da9d6943ab96c72ef50
Author: Igor Simdyanov <igorsimdyanov@gmail.com>
Date:   Sun Sep 8 22:04:56 2019 +0300

      Д о б а в л я е м  ф а й л readme.txt

commit dca0e8e4ed179eac45b4c607011aa21e208cf164
Author: Igor Simdyanov <igorsimdyanov@gmail.com>
Date:   Sun Sep 8 21:59:11 2019 +0300

      Comment

commit c24ec381a9eac257f93a40275b095f617ceb5ac3
Author: Igor Simdyanov <igorsimdyanov@gmail.com>
Date:   Sun Sep 8 21:56:34 2019 +0300

      П е р в ы й  к о м м и т
```

В результате выполнения команды выводится список коммитов в хронологическом порядке: от более свежих к более ранним. Если коммитов много, они не убираются на экране, двигаться дальше можно, нажимая на пробел. Выйти из режима просмотра можно, нажав q.

По умолчанию для каждого коммита выводится хэш, автор, время добавления и комментарий к коммиту.

Ограничить количество выводимых коммитов можно при помощи параметра -2.

```
$ git log -2
```

Таким образом, выводится лишь два коммита.

По умолчанию в отчете команды выводится только метаинформация. Чтобы вывести детальные изменения для каждого из коммитов, команде следует передать параметр **-p**.

```
$ git log -2 -p
```

Если нужна статистика для каждого коммита, можно воспользоваться параметром **--stat**:

```
$ git log -2 --stat
```

Под каждым из коммитов выводится список измененных файлов, а также количество строк, которое было добавлено и удалено в каждом из файлов. Каждая добавленная строка обозначается зеленым плюсом, а каждая удаленная — красным минусом.

Еще больше сократить вывод команды **git log** можно при помощи параметра **--pretty**, который изменяет формат вывода:

```
$ git log --pretty=oneline
9992e6df2da3270347f77bb8f9ed6d76dd34aedb (HEAD -> master) И з м е н е н и я  в
README-ф а й л е
e0c800c0646d885a64385da9d6943ab96c72ef50 Д о б а в л я е м  ф а й л  readme.txt
dca0e8e4ed179eac45b4c607011aa21e208cf164 Comment
c24ec381a9eac257f93a40275b095f617ceb5ac3 П е р в ы й  к о м м и т
```

Например, таким образом можно добиться, чтобы коммиты выводились в одну строку. Можно последовательно почитать комментарии к коммитам в хронологическом порядке.

Если вместо oneline использовать format, можно сформировать свой собственный формат логов:

```
$ git log --pretty=format:"%h - %an, %ar : %s"
9992e6d - Igor Simdyanov, 13 minutes ago : И з м е н е н и я  в README-ф а й л е
e0c800c - Igor Simdyanov, 14 minutes ago : Д о б а в л я е м  ф а й л  readme.txt
dca0e8e - Igor Simdyanov, 20 minutes ago : Comment
c24ec38 - Igor Simdyanov, 23 minutes ago : П е р в ы й  к о м м и т
```

Полный список параметров форматирования и их значения можно обнаружить в документации к git.

Хэш-сумма коммита

Как мы видели в выводе команды **git log**, каждый коммит снабжается хэш-суммой. Механизм, которым пользуется Git при вычислении хеш-сумм, называется SHA-1 хеш. Это строка длинной в 40 символов в шестнадцатеричном формате, каждая позиция может принимать либо числовое значение от 0 до 9, либо английский символ от a до f. Т. е., каждая из позиций кодирует 16 значений.

Хэш вычисляется на основе содержимого файла или структуры каталога. К коммиту всегда можно обратиться по этой хэш-сумме. Невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом, он постоянно вычисляет хэш-суммы объектов: папок и файлов. Как только выясняется, что хэш-сумма изменилась, значит изменилось состояние какого-то файла, в каталог был добавлен новый или удален существующий файл.

Ниже представлен типичный хеш:

```
fd076e9099e04eb66dc0966c9cbd765e76d243e7
```

Оперировать длинным хэшем неудобно, поэтому его можно сократить до размера, когда он еще отличается от других хэшей:

```
fd076e9
```

Как правило, 7-8 символов вполне достаточно.

Поиск по дате

Для поиска коммитов в определенном временном интервале можно использовать параметры **--since** и **--until**:

- **--until** позволяет задать дату, с которой мы начинаем поиск,
- **--since** — задать, до какого момента в прошлом мы ищем коммиты.

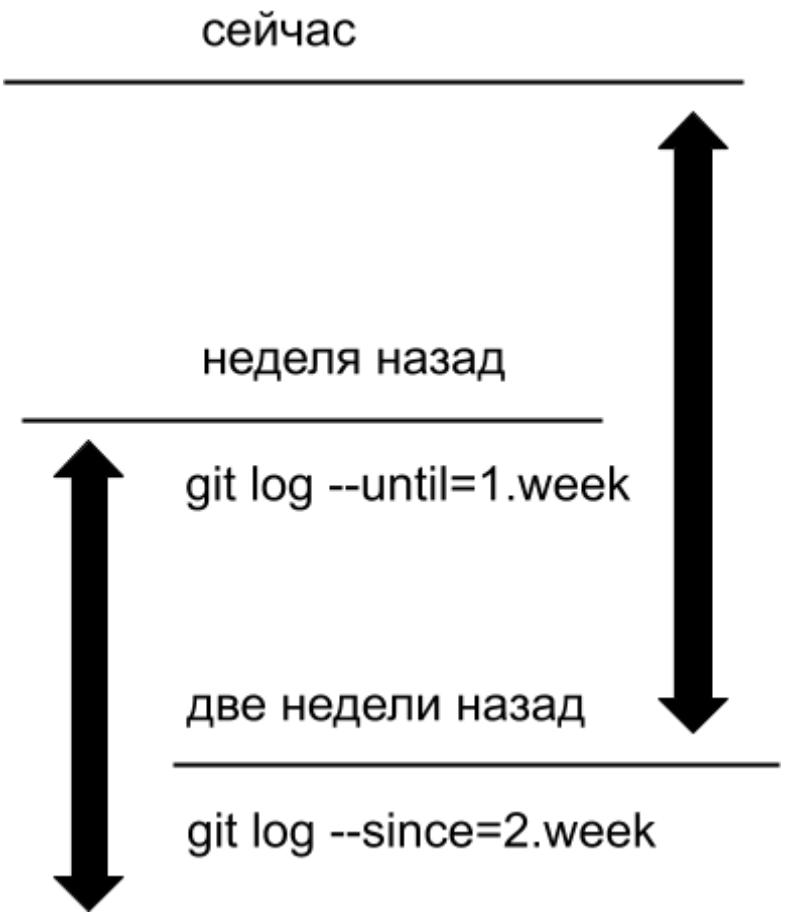
Команда ниже позволяет искать коммиты, начиная двумя неделями назад и до текущего момента:

```
$ git log --since=2.week
```

Следующая команда ищет коммиты с начала проекта до даты недельной давности:

```
$ git log --until=1.week
```

Можно одновременно использовать оба параметра, чтобы задать интервал поиска в прошлом.



Допускается использовать и точные даты.

```
$ git log --since=2019-03-01 --until=2019-03-25
```

Поиск коммита по строке и автору

При помощи параметра **-S** можно найти коммит, в котором была добавлена или удалена заданная строка:

```
$ git log -SGit -p
```

Команда выше ищет коммиты, в которых упоминается слово Git.

Иногда необходимо найти коммиты, которые принадлежат конкретному разработчику. Сделать это можно при помощи параметра **--author**:

```
$ git log --author="Igor Simdyanov"
```

Иногда стоит обратная задача, нужно, наоборот, выяснить, кто автор этой строки в файле и в каком коммите она в последний раз менялась Для этого предназначена команда **git blame**:

```
$ git blame readme.txt
e0c800c0 (Igor Simdyanov 2019-09-08 22:04:56 +0300 1) Новый проект
9992e6df (Igor Simdyanov 2019-09-08 22:06:46 +0300 2) Другая строка
```

В качестве параметра команда принимает имя файла. В результате выводится содержимое файла, в котором каждая строка помечена хэшем коммита, автором и датой последнего изменения.

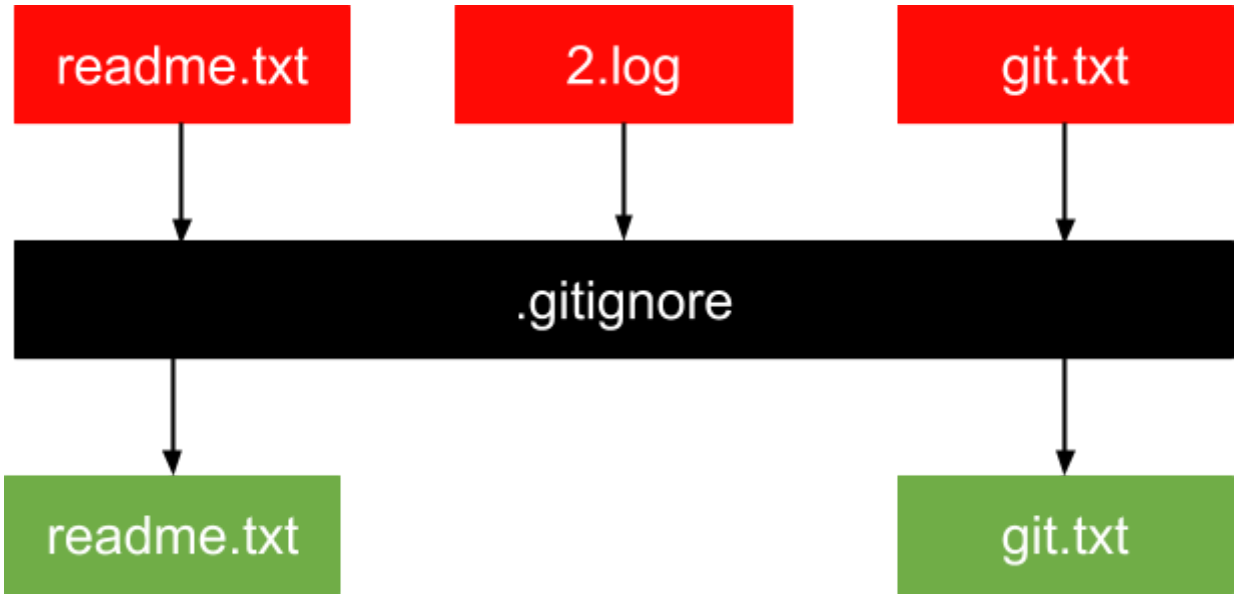
Игнорирование

Все, что помещается в .git-репозиторий, хранится в истории проекта. Если в коммит попал гигабайтный файл, последующее удаление его не приводит к удалению файла из истории. Пользователи, выполняющие клонирование репозитория, будут вынуждены каждый раз выгружать всю историю, включая этот гигантский файл. Чтобы Git игнорировал нежелательные файлы, предусмотрен специальный конфигурационный файл .gitignore, который помещается в корень проекта.

Внутри файла прописывается шаблон: файлы, которые удовлетворяют им, игнорируются Git.

```
*.log
tmp/*
```

Например, так мы будем игнорировать все файлы, которые завершаются расширением log, а также файлы в папке tmp.



Давайте добавим log-файл и поместим какой-либо файл в папку tmp. Кроме того, давайте создадим файл git.txt, который не должен попадать под действие шаблонов из файла .gitignore.

Если теперь запросить статус проекта:

```
$ git status
```

Мы видим, что log-файлы и содержимое папки tmp git вообще не воспринимает.

Ресурс http://gitignore.io/ позволяет автоматически формировать содержимое .gitignore-файла. Достаточно указать ключевые слова, например Sublime, Yii2, vim, и на их основе будет предложена заготовка для .gitignore, которая исключает временные и вспомогательные файлы.

Пустые папки

Git ориентируется на файлы, папки попадают под отслеживание, если только в них находятся файлы, пустые папки — игнорируются.

Давайте добавим пустую папку src в проект и запросим статус проекта при помощи команды **git status**:

```
$ git status
```

В результате папка не попадает в индекс. Это не всегда удобно, иногда папки нужны для функционирования проекта. Например, приложение может ожидать наличие папки log, но так как мы запретили log-файлы, такая папка рассматривается как пустая и не попадает в индекс git-проекта. Чтобы избежать этого, в папку часто добавляют файл .keep. Это просто обычный пустой файл, он не является частью экосистемы git и вместо него можно использовать любой другой файл. Файл начинается с точки, поэтому в UNIX-подобных операционных системах он рассматривается как скрытый. В Windows это не так, однако это название получило широкое распространение в тех случаях, когда важно сохранить пустую папку в git-проекте.

Используемые источники

Для подготовки методического пособия мы использовали эти ресурсы:

- Официальная документация Git **https://git-scm.com/book/ru/v2.**