

3. История и отслеживание изменений

Раздел познакомит с принципами построения линейной истории, механизмами обеспечения ее целостности и инструменты поиска по ней.

3.1. Структура линейной истории

Создадим новый файл, добавим его в индекс и сохраним в репозиторий.

```
$ echo 'Less arbitrary data' > file3.txt
```

```
$ git add .
```

```
$ git commit -m 'Add file3.txt'
[master ad9a212] Add file3
1 file changed, 1 insertion(+)
create mode 100644 file3.txt
```

💡 Tip

Команда `git add` ↗ может рекурсивно добавлять содержимое директорий.

Рассмотрим созданный коммит.

```
$ git cat-file -p ad9a212
tree 08f82821f9970ea05eb6b8797124d8ba702ea1d4
parent 60492349e637ccee64ceded6894b88f8e7e8bd72
author Aleksei Sokolov <Aleksei_Sokolov2@epam.com> 1626338153 +0300
committer Aleksei Sokolov <Aleksei_Sokolov2@epam.com> 1626338153 +0300

Add file3.txt
```

💡 Tip

Идентификатор объекта можно сокращать до 4 символов, пока сокращение уникально.

В содержимом коммита появилась новая строка с заголовком `parent`. Она указывает на коммит, который был основой для создания текущего. Каждое последующее изменение отправляет к предыдущему и далее до первого коммита в истории.

⚠ Attention

С какой целью каждый коммит отправляет к своему родителю?

Причины наличия указателя на предыдущий коммит

Указатель на родительский коммит хэшируется вместе с коммитом, обеспечивая гарантию целостности и неразрывности истории изменений.

⚠ Attention

Каково содержимое дерева в этом коммите?

Содержимое дерева

В дереве хранятся ссылки на все объекты, находившиеся в репозитории и индексе на момент создания коммита.

```
$ git cat-file -p 08f82821f9970ea05eb6b8797124d8ba702ea1d4
100644 blob 8e4b04b468c5350cad218004489cc896e80df946    file1.txt
100644 blob 8e4b04b468c5350cad218004489cc896e80df946    file2.txt
100644 blob fe5cc1589456736495519b147d6141c1eb188f90    file3.txt
```

Хранение полного дерева для каждого коммита позволяет оперативно сравнивать произвольные точки в истории изменений.

3.2. Просмотр истории

Git предоставляет возможность отследить всю последовательность изменений, начиная с текущего коммита. Отобразить историю можно с помощью команды

`git log` ↗.

```
$ git log
commit ad9a2121363fb677b8b32c843616f019497f2b3e (HEAD -> master)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date: Thu Jul 15 11:40:37 2021 +0300

    Add file3

commit 60492349e637ccee64ceded6894b88f8e7e8bd72
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date: Thu Jul 15 11:33:44 2021 +0300

    Add file1 and file2
```

Просмотр истории позволяет отследить последние изменения вместе с авторством и получить представление о сути внесенных изменений.

3.3. Поиск источника изменений

Иногда необходимо выяснить, в какой момент и кем были внесены изменения. К примеру, это может помочь связаться с автором кода, либо понять, в какой момент в коде появился баг. Найти коммит, привнесший изменения, позволяет команда

`git blame` [↗](#)

```
$ git blame file1.txt
^6049234 (Aleksei Sokolov 2021-07-15 11:33:44 +0300 1) Arbitrary text
```

В каждой строке можно увидеть идентификатор коммита, упоминание автора и даты.

3.4. Введение в указатели

До этого момента все операции ограничивались манипуляциями с бинарными объектами, деревьями и коммитами. Вне зависимости от этого в репозитории появились другие объекты, которые имеют непосредственную важность для использования функционала Git. Обратимся к листингу файлов.

```
$ tree -aF -I '*sample|objects'
```

```
.
├── .git/
│   ├── COMMIT_EDITMSG
│   ├── HEAD
│   ├── config
│   ├── description
│   ├── hooks/
│   ├── index
│   ├── info/
│   │   └── exclude
│   ├── logs/
│   │   ├── HEAD
│   │   └── refs/
│   │       └── heads/
│   │           └── master
│   └── refs/
│       ├── heads/
│       │   └── master
│       └── tags/
├── file1.txt
├── file2.txt
└── file3.txt
```

9 directories, 12 files

Первым делом рассмотрим файл `.git/HEAD`. Это текстовый файл.

```
$ cat .git/HEAD
ref: refs/heads/master
```

`HEAD` — это указатель (`reference`), определяющий состояние, в которое была приведена рабочая директория. Однако, он содержит не идентификатор коммита, а указатель на другой файл. По этому пути расположен файл, ссылающийся на конкретный коммит.


```
$ cat .git/refs/heads/master
ad9a2121363fb677b8b32c843616f019497f2b3e
```

ⓘ Attention

Для чего используется файл HEAD?



Предназначение указателя HEAD



Файл HEAD служит указателем на текущий коммит, относительно которого строится история рабочей директории. `git status`  будет сравнивать состояние файлов в рабочей директории с деревом этого коммита. Родительским для следующего коммита станет указанный в `HEAD` коммит.

Подробнее указатели будут рассмотрены в следующем разделе.

3.5. Итоги раздела

- История формируется из последовательно связанных коммитов
- Каждый коммит указывает на дерево со списком файлов на момент его создания
- `HEAD` указывает на состояние, в которое была приведена рабочая директория
- Команды для работы с историей:
 - `git log`  — просмотр истории коммитов
 - `git blame`  — поиск источника изменений