1. Введение в системы версионирования

Раздел познакомит с основными процессами совместной работы над файлами и кодом, а также с инструментами, которые были созданы для обеспечения этих процессов.

1.1. Совместная работа над данными

Предположим, что у вас есть черновик важного письма в простом текстовом документе, над которым вы хотите работать совместно с коллегами. Каждый из них будет ответственен за различные аспекты этого черновика. В случае возникновения вопросов необходим способ определения автора фрагмента.

Attention

Каким образом можно организовать подобный процесс?

Варианты решения

- внешний накопитель
- электронная почта
- общий сетевой диск

Attention

Как вы думаете, какой из этих способов обладает наибольшим и наименьшим количеством возможностей?

Рассмотрим каждый из способов, задав себе четыре вопроса:

1. Можно ли работать с данными одновременно?
2. Можно ли работать с данными автономно?
3. Можно ли отследить историю изменений?
4. Можно ли установить авторство изменений?

Классификация инструментов для совместной работы

Внешний накопитель

Основная копия файла всегда находится на диске, который можеть быть в произвольный момент времени только у одного человека. Получив внешний накопитель, пользователь может работать с файлом автономно. Историю изменений и их авторство невозможно отследить, т.к. дата, время и авторство сохраняется только для последнего изменения.

- 🔴 Одновременная работа
- 🟢 Автономная работа
- 🔴 История изменений
- 🔴 Авторство

Электронная почта

Основная копия файла пересылается в электронном письме. Во избежание конфликтов каждый пользователь работает над файлом по очереди. Получив письмо, пользователь может работать с файлом автономно. Историю правок можно отследить путем сравнения нескольких версий файла, отправленных в разное время, а авторство можно установить по отправителю письма с версией, где искомый фрагмент подвергся изменению.

- 🔴 Одновременная работа
- 🟢 Автономная работа
- 🟢 История изменений
- 🟢 Авторство

Общий сетевой диск

Основная копия файла доступна всем участникам, однако в силу того, что файл является простым текстовым документом, одновременное внесение правок может привести к потере части изменений. Файл также не доступен без постоянной связи с сетевым диском, что исключает возможность автономной работы. Историю изменений и их авторство невозможно отследить, т.к. дата, время и авторство сохраняется только для последнего изменения.

- Одновременная работа
- Автономная работа
- История изменений
- Авторство

Итоговое сравнение

Электронная почта	🔴🟢🟢🟢
Внешний накопитель	🔴🔴🔴🟢
Общий сетевой диск	🔴🔴🔴🔴

Все перечисленные требования равноценно применимы к любому процессу совместной работы над кодом. Для построения подобных процессов были созданы инструменты, обеспечивающие частичное или полное выполнение всех упомянутых требований.

1.2. Почему Git?

1. **Полная децентрализованность и автономность**
   Git не требует наличия централизованного хранилища и позволяет осуществлять обмен данными путем передачи обновлений между автономными репозиториями.
2. **Исчерпывающая история изменений с указанием авторства**
   Механизм позволяет восстановливать хронологию модификации кода и осуществлять возврат к произвольной её точке.
3. **Гарантии целостности хранимых данных**
   Консистентность данных и истории обеспечивается с помощью криптографических операций.
4. **Быстрое переключение контекста**
   Возможность хранить изолированные контексты позволяет работать над несколькими задачами параллельно без пересечения изменений.
5. **Популярность**
   Git используется в подавляющем большинстве компаний, связанных с разработкой, в т.ч. Google, Facebook, Microsoft, Twitter, Netflix и других.

Tip

Забавный факт: На Github есть зеркало ↗ официального репозитория Apache Subversion.

Подробнее ознакомиться с отличительными чертами Git можно на сайте проекта ↗.

1.3. Другие системы версионирования

Помимо Git существуют другие системы версионирования кода. Обсуждение их преимуществ и недостатков выходит за рамки данного курса. В этом разделе собраны ссылки на источники информации о них.

<details>
<summary>SVN</summary>

- Subversion — официальный сайт ↗
- Wikipedia — Apache Subversion ↗
- Perforce — Git vs. SVN – What Is The Difference? ↗
- Tabnine — SVN vs. Git: Which is right for you in 2020? ↗
</details>

<details>
<summary>Mercurial</summary>

- Mercurial - официальный сайт ↗
- Wikipedia - Mercurial ↗
- Atlassian - Git vs. Mercurial: why Git? ↗
- Perforce - Mercurial vs. Git: How Are They Different? ↗
- Хабр - Сходство и различие между Mercurial и Git ↗
</details>

<details>
<summary>Сравнения трех и более систем контроля версий</summary>

- Wikipedia — Comparison of version-control software ↗
- Medium — Version Control Software Comparison: Git, Mercurial, CVS, SVN ↗
</details>

2. Основные концепции Git

Раздел познакомит с терминологией Git, технической реализацией основных концепций, и предоставит примеры использования базовых команд.

2.1. Система отслеживания изменений

Представим производственную линию завода, на которой каждый день есть один сменяемый дежурный. По мере выполнения каждой операции дежурный заносит информацию о ней в блокнот, а вечером переносит содержимое блокнота в журнал. В результате получается три источника информации о выполненных работах:

1. Фактическое состояние линии, где часть работ может быть не завершена
2. Список выполненных работ в блокноте дежурного
3. Журнал изменений, хранящий последовательную историю всех работ

Аналогично этому примеру, Git хранит три состояния файлов.

1. **Рабочая директория (working directory)**
   Директория проекта, в которой осуществляется работа с файлами.
2. **Индекс (staging area)**
   Перечень файлов, подготовленных к отправке в репозиторий.
3. **Репозиторий (repository)**
   Финализированная последовательная история изменений.

2.2. Структура репозитория

Git изначально задумывался, как децентрализованная система, поэтому всё необходимые для полноценной работы репозитория хранится непосредственно в директории проекта внутри поддиректории `.git`

Tip

Любой репозиторий — ни что иное, как набор директорий и файлов.

У каждого пользователя может содержаться полная или частичная копия репозитория, позволяющая работать в условиях полной автономности.

Создание репозитория осуществляется с помощью команды `git init` ↗

```
$ mkdir /tmp/git
$ cd /tmp/git
```

```
$ git init .
Initialized empty Git repository in /tmp/git/.git/
```

Tip

Точка в конце строки является указателем на текущую директорию. Вместо точки можно использовать путь к произвольной директории.

В результате выполения команды создается структура из директорий и файлов, необходимых для функционирования репозитория.

```
$ tree -aF -I '*sample'
.
└── .git/
    ├── HEAD
    ├── config
    ├── description
    ├── hooks/
    ├── info/
    │   └── exclude
    ├── objects/
    │   ├── info/
    │   └── pack/
    └── refs/
        ├── heads/
        └── tags/

9 directories, 4 files
```

Tip

Git создает файлы шаблонов в директории `hooks`, однако в рамках курса они будут скрываться до момента, когда курс затронет их напрямую.

2.3. Операции в рабочей директории

В процессе внесения изменений в файлы в директории проекта состояние репозитория не меняется. В этом можно убедиться, создав файл и повторив операцию листинга файлов.

```
$ echo 'Arbitrary text' > file1.txt
```

```
$ tree -aF -I '*sample'
.
├── .git/
│   ├── HEAD
│   ├── config
│   ├── description
│   ├── hooks/
│   ├── info/
│   │   └── exclude
│   ├── objects/
│   │   ├── info/
│   │   └── pack/
│   └── refs/
│       ├── heads/
│       └── tags/
└── file1.txt

9 directories, 5 files
```

Tip

Git не выполняет операции самостоятельно и не отслеживает состояние файловой системы. Все действия над репозиторием выполняются путем запуска команд.

2.4. Подготовка к сохранению

После завершения работ над файлом необходимо добавить его в индекс, тем самым пометив его к сохранению в репозиторий. Для этого используется команда `git add` ↗

```
$ git add file1.txt
```

```
$ tree -aF -I '*sample'
.
├── .git/
│   ├── HEAD
│   ├── config
│   ├── description
│   ├── hooks/
│   ├── index
│   ├── info/
│   │   └── exclude
│   ├── objects/
│   │   ├── 8e/
│   │   │   └── 4b04b468c5350cad218004489cc896e80df946
│   │   ├── info/
│   │   └── pack/
│   └── refs/
│       ├── heads/
│       └── tags/
└── file1.txt
```

В результате запуска команды в директории `.git` появились два файла:

- `.git/index`
- `.git/objects/8e/4b04b468c5350cad218004489cc896e80df946`

Первый файл является списком, в котором содержится перечень всех файлов, помеченных к сохранению. Для просмотра индекса используется команда `git ls-files --stage` ↗

```
$ git ls-files --stage
100644 8e4b04b468c5350cad218004489cc896e80df946 0   file1.txt
```

В выводе команды присутствуют тип исходного файла, его имя, права доступа, а также идентификатор из символов шестнадцатеричной системы, который является хэш-суммой от содержимого исходного файла. Он совпадает с именем одного из файлов в директории `.git/objects`, внутри которого в формате `gzip` находится содержимое исходного. Убедиться в этом можно с помощью команды `git cat-file` ↗. Первый запуск определит тип объекта, а второй отобразит содержимое.

```
$ git cat-file -t 8e4b04b468c5350cad218004489cc896e80df946
blob
```

```
$ git cat-file blob 8e4b04b468c5350cad218004489cc896e80df946
Arbitrary text
```

Tip

`blob` расшифровывается как `binary large object`

Tip

Первые два символа хэш-суммы переносятся в имя директории с целью разделения файлов на группы. Это предотвращает замедление файловой системы.

2.5. Отслеживание состояния рабочей директории

Рассмотренная ранее команда `git ls-files` ↗ позволяет просматривать не только список подготовленных для сохранения файлов, но и перечень модифицированных или созданных файлов, не добавленных в индекс. Команда `git status` ↗ отображает все три набора файлов, чем существенно упрощает отслеживание изменений.

Создадим в директории еще один файл и рассмотрим вывод этой команды.

```
$ echo 'Arbitrary text' > file2.txt
```

```
$ git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
    new file:   file1.txt

Untracked files:
(use "git add <file>..." to include in what will be committed)
    file2.txt
```

Команда сравнивает состояние рабочей директории с индексом и сохраненными в репозитории данными, позволяя тем самым отслеживать появление новых файлов, удаление существующих и внесение изменений как до, так и после добавления файлов в индекс.

Tip

Команда `git status` ↗ подсказывает синтаксис команд для работы с индексом.

Добавим второй файл в индекс и сравним вывод команд.

```
$ git add file2.txt
```

```
$ git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
    new file:   file1.txt
    new file:   file2.txt
```

```
$ tree -aF -I '*sample'
.
├── .git/
│   ├── HEAD
│   ├── config
│   ├── description
│   ├── hooks/
│   ├── index
│   ├── info/
│   │   └── exclude
│   ├── objects/
│   │   ├── 8e/
│   │   │   └── 4b04b468c5350cad218004489cc896e80df946
│   │   ├── info/
│   │   └── pack/
│   └── refs/
│       ├── heads/
│       └── tags/
├── file1.txt
└── file2.txt

10 directories, 8 files
```

Не смотря на то, что файл был добавлен в индекс, количество файлов в директории `.git/objects` не изменилось. Это связано с тем, что Git не хранит метаинформацию внутри объекта и хэширует только содержимое файла.

Tip

Хэширование содержимого без метаданных позволяет экономить пространство на диске и сокращать количество сохраняемых файлов.

2.6. Сохранение данных в репозиторий

Для того, чтобы перманентно сохранить текущее содержимое индекса в репозиторий, используется команда `git commit` ↗ Выполнение команды запустит текстовый редактор и предложит изменить сообщение, которое будет добавлено к сохраняемым

данным.

Содержание сообщения должно отражать суть вносимых изменений. Это может быть идентификатор задачи, упоминание исправляемой проблемы, общее описание правок. Git прервет выполнение операции, если сообщение будет пустым.

Tip

Большинство систем визуализации отображают только первую строчку сообщения. Старайтесь делать ее содержимое кратким и емким.

Tip

Комментарий можно внести из командной строки: `git commit -m '<сообщение>'` ↗

После сохранения и закрытия текстового файла Git продолжит выполнять операции и выведет информацию по результатам.

```
$ git commit -m 'Add file1 and file2'
[master (root-commit) 6049234] Add file1 and file2
2 files changed, 2 insertions(+)
create mode 100644 file1.txt
create mode 100644 file2.txt
```

Tip

Объект, создаваемый в результате операции, также называется `commit`

Рассмотрим содержимое директории `.git` после выполнения данной операции.

```
$ tree -aF -I '*sample'
.
├── .git/
│   ├── COMMIT_EDITMSG
│   ├── HEAD
│   ├── config
│   ├── description
│   ├── hooks/
│   ├── index
│   ├── info/
│   │   └── exclude
│   ├── logs/
│   │   ├── HEAD
│   │   └── refs/
│   │       └── heads/
│   │           └── master
│   ├── objects/
│   │   ├── 60/
│   │   │   └── 492349e637ccee64ceded6894b88f8e7e8bd72
│   │   ├── 8e/
│   │   │   └── 4b04b468c5350cad218004489cc896e80df946
│   │   ├── e7/
│   │   │   └── 4a8bdc9580ebed872e6b64a16c075f8194b1db
│   │   ├── info/
│   │   └── pack/
│   └── refs/
│       ├── heads/
│       │   └── master
│       └── tags/
├── file1.txt
└── file2.txt

15 directories, 14 files
```

В результате выполнения команды в директории `.git` появилось несколько новых фалов. Остановимся на файлах, размещенных в директории `.git/objects`. Как можно заметить, первые символы одного из них совпадают с набором символов в выводе команды `git commit` ↗, для просмотра его содержимого используем команду `git cat-file` ↗

```
$ git cat-file -t 60492349e637ccee64ceded6894b88f8e7e8bd72
commit
```

```
$ git cat-file commit 60492349e637ccee64ceded6894b88f8e7e8bd72
tree e74a8bdc9580ebed872e6b64a16c075f8194b1db
author Aleksei Sokolov <Aleksei_Sokolov2@epam.com> 1626338024 +0300
committer Aleksei Sokolov <Aleksei_Sokolov2@epam.com> 1626338024 +0300

Add file1 and file2
```

В тексте указан автор изменений, создатель коммита и соответствующие отметки времени. Также в первой строке отображается информация о новом объекте под названием `tree`. Его можно просмотреть с помощью команды `git cat-file` ↗. Объект `tree`, как и индекс, из которого он формируется, является бинарным. Для просмотра его содержимого необходимо конвертировать его в текстовый вид.

Tip

Команда `git cat-file -p` ↗ приводит любой объект к текстовому виду.

```
$ git cat-file -t e74a8bdc9580ebed872e6b64a16c075f8194b1db
tree
```

```
$ git cat-file -p e74a8bdc9580ebed872e6b64a16c075f8194b1db
100644 blob 8e4b04b468c5350cad218004489cc896e80df946        file1.txt
100644 blob 8e4b04b468c5350cad218004489cc896e80df946        file2.txt
```

В дереве содержится перечень файлов с их типами, правами и идентификаторами объектов. Этот список связывает коммит с файлами, которые были добавлены в текущем коммите или существовали в репозитории до его появления, а также хранит метаданные всех перечисленных файлов.

Tip

Git отслеживает смену имени, типа и прав доступа объектов файловой системы.

2.7. Итоги раздела

- Сущности репозитория хранятся внутри его собственной директории
- Git не производит автоматических операций над файловой системой
- Данные в Git разделены на три пространства
  - **рабочая директория** — фактическое состояние файлов в директории проекта
  - **индекс** — файлы, подготовленные к сохранению в репозиторий
  - **репозиторий** — файлы, сохраненные в виде наборов изменений
- Основные типы создаваемых объектов
  - **blob** — архивированное содержимое исходного файла
  - **tree** — дерево изменений с перечнем файлов и метаданных
  - **commit** — аннотированный набор изменений с указанием времени и авторства
- Команды для управления изменениями:
  - `git add` ↗ — добавление файлов в индекс
  - `git ls-files` ↗ — отображение списка файлов в пространствах
  - `git status` ↗ — вывод состояния изменений в репозитории

- `git cat-file` ↗ — отображение содержимого объектов
- `git commit` ↗ — сохранение изменений в историю репозитория

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 отсылает к предыдущему и далее до первого коммита в истории.

Attention

С какой целью каждый коммит отсылает к своему родителю?

<details>
<summary>Причины наличия указателя на предыдущий коммит</summary>

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

Attention

Каково содержимое дерева в этом коммите?

<details>
<summary>Содержимое дерева</summary>

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.
</details>

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 директирия
- Команды для работы с историей:
  - `git log` ↗ — просмотр истории коммитов
  - `git blame` ↗ — поиск источника изменений

4. Указатели, ветки и теги

Раздел познакомит с концепцией веток, перемещением по истории и тегами.

4.1. Использование нескольких контекстов

Представим, что есть необходимость работать с одним и тем же кодом в рамках двух задач, и каждую задачу требуется проверять отдельно. Подобная ситуация неразрешима при наличии линейной истории. В качестве решения Git предоставляет возможность разветвлять историю, создавая отдельные контексты для работы. Они называются ветками и управляются командой `git branch` ↗

```
$ git branch
* master
```

При запуске без параметров команда показывает список локальных веток и выделяет текущую ветку. Рассмотрим листинг файлов из предыдущего раздела.

```
$ 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
```

В директории `.git/refs/heads/` находится файл с названием, совпадающим с названием ветки. Эта директория предназначена для хранения указателей на верхушки (`head`) веток, т.е. последние коммиты в них.

Attention

Если ветка является указателем на коммит, как осуществляется изоляция контекстов?

Правила формирования контекстов

На самом деле ветвится не хранилище файлов, а история. У одного и того же родительского коммита может быть несколько дочерних. Каждый из дочерних коммитов будет в дальнейшем формировать отдельную линейную историю, сводящуюся к единому источнику - их родительскому коммиту.

При создании коммита Git анализирует содержимое файла `HEAD`. Если оно ссылается на указатель ветки, он будет переставлен на новый коммит.

4.2. Переключение состояния рабочей директории

Для того, чтобы продемонстрировать механизм ветвления в действии, нам потребуется команда, позволяющая двигаться по истории и указателям — `git checkout` ↗ Данная команда переставляет указатель `HEAD` на заданную точку в истории и формирует содержимое рабочей директории из ее дерева.

```
$ 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
```

```
$ git checkout 60492349e637ccee64ceded6894b88f8e7e8bd72
Note: switching to '60492349e637ccee64ceded6894b88f8e7e8bd72'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 6049234 Add file1 and file2
```

Вывод предупреждает, что рабочая директория находится в состоянии `detached HEAD` Рассмотрим содержимое `.git/HEAD`, чтобы понять смысл этого сообщения.

```
$ cat .git/HEAD
60492349e637ccee64ceded6894b88f8e7e8bd72
```

На этот раз в файле `HEAD` указан не путь к указателю ветки, а конкретный коммит. Таким образом следующий коммит не будет привязан к какой-либо ветке и будет создан без указателей на него. После переключения на другой коммит или ветку вернуться к созданному коммиту можно будет только по идентификатору.

Рассмотрим листинг файлов в рабочей директории, скрыв директорию `.git`

```
$ tree -aF -I '.git'
.
├── file1.txt
└── file2.txt
```

Файл `file3.txt` исчез из рабочей директории, так как она была приведена в состояние, соответствующее коммиту, в дереве которого этого файла нет. Создадим новый файл и сохраним его в репозиторий.

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

```
$ git add .
```

```
$ git commit -m 'Add file3 with new data'
[detached HEAD af57806] Add file3 with new data
1 file changed, 1 insertion(+)
create mode 100644 file3.txt
```

4.3. Создание ветки

На предыдущем этапе был создан коммит, не относящийся ни к одной ветке. Для удобства доступа можно создать ветку, которая будет указывать на этот коммит. Существует две команды, позволяющие выполнить это операцию: более новая `git switch` ↗ и исходная `git checkout` ↗. В примерах будет использоваться старая команда для совместимости.

```
$ git checkout -b 'branch_1'
```

Attention

Какие объекты будут созданы в результате выполения команды?

<details>
<summary>Объекты, появляющиеся при создании ветки</summary>

- указатель на ветку
- журнал перемещения указателя

Tip

Git хранит историю всех переключений указателей. Этот механизм позволяет восстанавливать состояние ветки после ошибочно выполненных операций. Журналы хранятся в файлах в директории `.git/logs/`. Оперативно посмотреть их можно с помощью команды `git reflog` ↗
</details>

Убедимся, что мы находимся в новой ветке, с помощью команды `git branch` ↗

```
$ git branch
* branch_1
  master
```

Git показывает, что в локальном репозитории существуют две ветки, и на данный момент `HEAD` рабочей директории указывает на ветку `branch_1`. Для перехода на другую ветку можно снова воспользоваться командой `git checkout` ↗

```
$ git checkout master
Switched to branch 'master'
```

В результате выполнения команды указатель переключается на другую ветку, а состояние рабочей директории приводится к состоянию дерева последнего коммита этой ветки.

```
$ cat file3.txt
Less arbitrary data
```

Не смотря на то, что в другой ветке содержимое данного файла иное, в текущей ветке сохранено его исходное содержимое.

4.4. Теги

В определенных ситуациях необходимо оставить в истории указатель на конкретный коммит. Он может служить отметкой для релиза или коммита, привнесшего баг. Ветки не подходят для подобных задач, так как их идентификатор переносится при добавлении коммитов. Вместо них используются теги, устанавливаемые командой `git tag` ↗

```
$ git tag original_file3
```

Команда выполнит установку тега с заданным именем на текущий коммит. Рассмотрим вывод команды `git log` ↗

```
$ git log
commit ad9a2121363fb677b8b32c843616f019497f2b3e (HEAD -> master, tag: original_file3)
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
```

В выводе команды отображаются все указатели, установленные на каждый коммит. На текущий коммит указывает `HEAD`, ссылающийся на ветку `master`, и тег `file3`. В листинге файлов в директории `.git` можно увидеть новый указатель `.git/refs/tags/file3`. Он содержит идентификатор коммита.

```
$ tree -aF -I '*sample|objects'
.
├── .git/
│   ├── COMMIT_EDITMSG
│   ├── HEAD
│   ├── ORIG_HEAD
│   ├── config
│   ├── description
│   ├── hooks/
│   ├── index
│   ├── info/
│   │   └── exclude
│   ├── logs/
│   │   ├── HEAD
│   │   └── refs/
│   │       └── heads/
│   │           ├── branch_1
│   │           └── master
│   └── refs/
│       ├── heads/
│       │   ├── branch_1
│       │   └── master
│       └── tags/
│           └── original_file3
├── file1.txt
├── file2.txt
└── file3.txt

9 directories, 16 files
```

```
$ cat .git/refs/tags/original_file3
ad9a2121363fb677b8b32c843616f019497f2b3e
```

4.5. Итоги раздела

- Ветки и теги — указатели на произвольную точку в истории
- Указатель ветки передвигается по мере создания коммитов
- Указатель тега привязан к конкретному коммиту
- Историю переключения каждого указателя можно отследить
- Команды для работы с указателями и перемещения по истории:
  - `git branch` ↗ — отображение имени ветки, на которую указывает `HEAD`
  - `git checkout` ↗ — переход на коммит или указатель
  - `git switch` ↗ — альтернативная команда для навигации по истории
  - `git reflog` ↗ — просмотр истории перестановки указателя
  - `git tag` ↗ — произведение операций с тегами

5. Объединение изменений

Раздел познакомит с различными механизмами объединения изменений.

5.1. Введение в объединение веток

Зачастую структура репозитория подразумевает наличие одной или нескольких веток, в которые будет вносится содержимое остальных. Для объединения изменений из двух веток существует несколько способов. Демонстрация операций будет производится на специально подготовленном репозитории.

Ссылка на демонстрационный репозиторий

https://github.com/aleksei-sokolov/devops-school-git

В демонстрационном репозитории результат работ хранится в ветке `master`, а история коммитов обусловлена последовательностью событий:

- Команда разработки написала код в файле `code.txt`
- Результаты тестирования кода были сохранены в `test_results.txt`
- Не смотря на ошибки было решено выпустить релиз `v0.1`
- Команда начала исправление бага в ветке `bugfix`
- Результаты тестов исправленного кода обновили `test_results.txt`
- Обратная связь по результатам релиза была собрана в `feedback.txt`
- Было решено разработать новый функционал в ветке `new_feature`
- Результаты тестов нового функционала обновили `test_results.txt`
- Планы по разработке были подговтовлены в ветке `roadmap`

Древовидную историю можно визуализировать командой `git log --all --graph` ↗

```
* commit a59d38377a09be7cc4ee1a34063c23366d7d94df (HEAD -> roadmap, origin/roadmap)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Thu Jul 15 11:57:26 2021 +0300
|
|     Add roadmap
|
| * commit 13ce06113b312473249c16de9e5606e4f7b44afd (origin/new_feature, new_feature)
| | Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| | Date:   Wed Jul 14 12:57:57 2021 +0300
| |
| |     Add test results for the new feature
| |
| * commit b7ab163c6c1aa710b06d1a9877126abe212783c6
|/  Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
|   Date:   Wed Jul 14 12:57:05 2021 +0300
|
|       Add new feature
|
* commit 7dde0effd0ff79b96f7973123e16738aed9734c1 (origin/master, master)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:01:33 2021 +0300
|
|     Add user feedback
|
| * commit 9546beea2635b1b54edf384efa038ff06042fd02 (origin/bugfix, bugfix)
| | Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| | Date:   Wed Jul 14 12:55:19 2021 +0300
| |
| |     Add bugfix test results
| |
| * commit 1033d95aa08809c37c15a43fc33d9fd7b9c42d25
|/  Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
|   Date:   Wed Jul 14 12:00:14 2021 +0300
|
|       Fix bug in code
|
* commit 7bc0cb6696c337ab502b5666c694a233eca90f0b (tag: v0.1)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 11:57:16 2021 +0300
|
|     Add test results
|
* commit a15efae1f812b5378c12fedb45de1d722f0949df
  Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
  Date:   Wed Jul 14 11:55:22 2021 +0300

      Add initial code
```

Tip

Сократить вывод `git log --all --graph` ↗ можно с помощью ключа `--oneline`

Использование ключа `--oneline`

Tip

Указатели, начинающиеся с `origin/` будут рассмотрены позже.

В зависимости от ситуации объединение может происходить несколькими способами:

- Слияние через перестановку указателей
- Слияние с нелинейной историей
- Перенос коммитов в конец линейной истории

5.2. Слияние в рамках линейной истории

Первой задачей стоит внесение планов разработки в ветку `master`. Рассмотрим историю ветки `roadmap` с помощью команды `git log` ↗

```
$ git log roadmap
```

```
commit a59d38377a09be7cc4ee1a34063c23366d7d94df (origin/roadmap, roadmap)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Thu Jul 15 11:57:26 2021 +0300

    Add roadmap

commit 7dde0effd0ff79b96f7973123e16738aed9734c1 (HEAD -> master, origin/master)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 12:01:33 2021 +0300

    Add user feedback

commit 7bc0cb6696c337ab502b5666c694a233eca90f0b (tag: v0.1)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:57:16 2021 +0300

    Add test results

commit a15efae1f812b5378c12fedb45de1d722f0949df
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:55:22 2021 +0300

    Add initial code
```

Tip

Команда `git log` ↗ позволяет отобразить историю ветки, не переключаясь на нее.

Attention

Какой из способов подойдет для переноса изменений в `master`?

Обновление ветки `master`

Путь от последнего коммита в ветке `master` до последнего коммита в ветке `roadmap` линеен. Достаточно переставить указатель ветки `master` на соответсвующий коммит.

Переключимся на ветку, в которую хотим внести изменения.

```
$ git checkout master
```

Для слияния веток в одной линейной истории используется команда `git merge` ↗

```
$ git merge roadmap
Updating 7dde0ef..a59d383
Fast-forward
 roadmap.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 roadmap.txt
```

Упоминание режима `fast-forward` означает, что обновление произошло путем перемещения указателя, а новые объекты не создавались. Повторный запуск `git log` ↗ подтверждает, что указатели обеих веток ссылаются на один и тот же коммит.

```
$ git log
    commit a59d38377a09be7cc4ee1a34063c23366d7d94df (HEAD -> master, origin/roadmap,
roadmap)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Thu Jul 15 11:57:26 2021 +0300

    Add roadmap

commit 7dde0effd0ff79b96f7973123e16738aed9734c1 (origin/master)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 12:01:33 2021 +0300

    Add user feedback

commit 7bc0cb6696c337ab502b5666c694a233eca90f0b (tag: v0.1)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:57:16 2021 +0300

    Add test results

commit a15efae1f812b5378c12fedb45de1d722f0949df
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:55:22 2021 +0300

    Add initial code
```

5.3. Слияние с нелинейной историей

Следующей задачей станет внесение в ветку `master` изменений из ветки `new_feature` Рассмотрим историю обеих.

```
$ git log master
    commit a59d38377a09be7cc4ee1a34063c23366d7d94df (HEAD -> master, origin/roadmap,
roadmap)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Thu Jul 15 11:57:26 2021 +0300

    Add roadmap

commit 7dde0effd0ff79b96f7973123e16738aed9734c1 (origin/master)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 12:01:33 2021 +0300

    Add user feedback

commit 7bc0cb6696c337ab502b5666c694a233eca90f0b (tag: v0.1)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:57:16 2021 +0300

    Add test results

commit a15efae1f812b5378c12fedb45de1d722f0949df
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:55:22 2021 +0300

    Add initial code
```

```
$ git log new_feature
    commit 13ce06113b312473249c16de9e5606e4f7b44afd (origin/new_feature, new_feature)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 12:57:57 2021 +0300

    Add test results for the new feature

commit b7ab163c6c1aa710b06d1a9877126abe212783c6
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 12:57:05 2021 +0300

    Add new feature

commit 7dde0effd0ff79b96f7973123e16738aed9734c1 (origin/master)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 12:01:33 2021 +0300

    Add user feedback

commit 7bc0cb6696c337ab502b5666c694a233eca90f0b (tag: v0.1)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:57:16 2021 +0300

    Add test results

commit a15efae1f812b5378c12fedb45de1d722f0949df
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:55:22 2021 +0300

    Add initial code
```

Attention

Указатели `origin/` следует игнорировать, они будут рассмотрены позднее.

Вывод обеих команд указывает на то, что последние коммиты в ветках `master` и `new_feature` не объединены линейной историей. Следовательно, слияние путем перестановки указателя невозможно. В таких случаях можно использовать как слияние с нелинейной историей, так и копирование коммитов в линейную. Рассмотрим первую операцию, для которой также используется команда `git merge` ↗

```
$ git branch
  bugfix
* master
  new_feature
  roadmap
```

```
$ git merge new_feature
```

Для создания нелинейной истории требуется создать коммит, который объединит хронологию двух ее ветвей. После запуска команды `git merge` ↗ откроется текстовый редактор для внесения сообщения к коммиту. Первая строка содержит типовое сообщение.

```
Merge branch 'new_feature'
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

```
Merge made by the 'recursive' strategy.
 code.txt         | 1 +
 test_results.txt | 1 +
 2 files changed, 2 insertions(+)
```

История ветки стала нелинейной. Рассмотрим ее структуру и последний коммит.

```
$ git log --graph
*   commit 49cee3b567752209d0b1447ff6b78511aa3816f0 (HEAD -> master)
|\  Merge: a59d383 13ce061
| | Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| | Date:   Thu Jul 15 13:03:19 2021 +0300
| |
| |     Merge branch 'new_feature'
| |
| * commit 13ce06113b312473249c16de9e5606e4f7b44afd (origin/new_feature, new_feature)
| | Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| | Date:   Wed Jul 14 12:57:57 2021 +0300
| |
| |     Add test results for the new feature
| |
| * commit b7ab163c6c1aa710b06d1a9877126abe212783c6
| | Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| | Date:   Wed Jul 14 12:57:05 2021 +0300
| |
| |     Add new feature
| |
* | commit a59d38377a09be7cc4ee1a34063c23366d7d94df (origin/roadmap, roadmap)
|/  Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
|   Date:   Thu Jul 15 11:57:26 2021 +0300
|
|       Add roadmap
|
* commit 7dde0effd0ff79b96f7973123e16738aed9734c1 (origin/master)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:01:33 2021 +0300
|
|     Add user feedback
|
* commit 7bc0cb6696c337ab502b5666c694a233eca90f0b (tag: v0.1)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 11:57:16 2021 +0300
|
|     Add test results
|
* commit a15efae1f812b5378c12fedb45de1d722f0949df
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:55:22 2021 +0300

    Add initial code
```

```
$ git cat-file -p 49cee3b567752209d0b1447ff6b78511aa3816f0
tree e9b20741f9c158f24dba80b4d071a8e7e9ecaf0d
parent a59d38377a09be7cc4ee1a34063c23366d7d94df
parent 13ce06113b312473249c16de9e5606e4f7b44afd
author Aleksei Sokolov <Aleksei_Sokolov2@epam.com> 1626343399 +0300
committer Aleksei Sokolov <Aleksei_Sokolov2@epam.com> 1626343399 +0300

Merge branch 'new_feature'
```

У коммита есть дерево файлов, а также ссылки на два родительских объекта.

5.4. Ручное перемещение указателей

Переставлять указатели веток можно вручную. В данном примере это будет сделано с целью отмены предыдущей операции. Перестановка указателя не удалит созданный предыдущей операцией коммит, однако он перестанет быть частью ветки и будет доступен только по идентификатору. Перестановка указателей выполняется командой `git reset` ↗

Tip

Все операции применяются к текущей ветке, если не указано иное.

```
$ git branch
  bugfix
* master
  new_feature
  roadmap
```

```
$ git reset --hard a59d38377a09be7cc4ee1a34063c23366d7d94df
HEAD is now at a59d383 Add roadmap
```

Указатель `HEAD` вслед за указателем ветки был переставлен на другой коммит, а ключ `--hard` привел рабочую директорию в соответствие с его деревом.

```
$ git branch
  bugfix
* master
  new_feature
  roadmap
```

```
$ git log
    commit a59d38377a09be7cc4ee1a34063c23366d7d94df (HEAD -> master, origin/roadmap,
roadmap)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Thu Jul 15 11:57:26 2021 +0300

    Add roadmap

commit 7dde0effd0ff79b96f7973123e16738aed9734c1 (origin/master)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 12:01:33 2021 +0300

    Add user feedback

commit 7bc0cb6696c337ab502b5666c694a233eca90f0b (tag: v0.1)
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:57:16 2021 +0300

    Add test results

commit a15efae1f812b5378c12fedb45de1d722f0949df
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:55:22 2021 +0300

    Add initial code
```

5.5. Перенос коммитов в конец линейной истории

Для превращения разветвленной историю в линейную существует механизм переноса коммитов в конец линейной истории. Процесс состоит из следующих этапов:

- Выбирается ветка, которая станет основой для текущей
- У двух веток ищется последний общий коммит
- Коммиты из текущей ветки пересоздаются поверх последнего коммита основной
- Указатель текущей ветки переставляется на последний созданный коммит

Операция производится командой `git rebase` ↗ из ветки, содержащей обновления.

```
$ git checkout new_feature
Switched to branch 'new_feature'
```

```
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Add new feature
Applying: Add test results for the new feature
```

Вывод показывает, что указатель `HEAD` был перемещен на последний коммит в основной ветке, после чего были заново созданы оба коммита, отличавшие текущую ветку от общего с основной веткой состояния. Рассмотрим историю с помощью

команды `git log` ↗

```
* commit edb3a693dcc61305683da84390ffb4ca98bff606 (HEAD -> new_feature)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:57:57 2021 +0300
|
|     Add test results for the new feature
|
* commit 11281ba935ee627cf78678a503015770f5d2fec1
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:57:05 2021 +0300
|
|     Add new feature
|
* commit a59d38377a09be7cc4ee1a34063c23366d7d94df (origin/roadmap, roadmap, master)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Thu Jul 15 11:57:26 2021 +0300
|
|     Add roadmap
|
| * commit 13ce06113b312473249c16de9e5606e4f7b44afd (origin/new_feature)
| | Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| | Date:   Wed Jul 14 12:57:57 2021 +0300
| |
| |     Add test results for the new feature
| |
| * commit b7ab163c6c1aa710b06d1a9877126abe212783c6
|/  Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
|   Date:   Wed Jul 14 12:57:05 2021 +0300
|
|       Add new feature
|
* commit 7dde0effd0ff79b96f7973123e16738aed9734c1 (origin/master)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:01:33 2021 +0300
|
|     Add user feedback
|
| * commit 9546beea2635b1b54edf384efa038ff06042fd02 (origin/bugfix, bugfix)
| | Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| | Date:   Wed Jul 14 12:55:19 2021 +0300
| |
| |     Add bugfix test results
| |
| * commit 1033d95aa08809c37c15a43fc33d9fd7b9c42d25
|/  Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
|   Date:   Wed Jul 14 12:00:14 2021 +0300
|
|       Fix bug in code
|
* commit 7bc0cb6696c337ab502b5666c694a233eca90f0b (tag: v0.1)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 11:57:16 2021 +0300
|
|     Add test results
|
* commit a15efae1f812b5378c12fedb45de1d722f0949df
  Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
  Date:   Wed Jul 14 11:55:22 2021 +0300

      Add initial code
```

Tip

Благодаря указателю `origin/` можно увидеть исходную историю ветки.

Attention

Почему идентификаторы коммитов изменились при переносе?

<details>
<summary>Причина изменения идентификторов</summary>

Коммиты содержат указатель на родительский объект, который изменился.
</details>

После выполения операции ветки `new_feature` и `master` стали частью линейной истории. Для внесения изменений в ветку `master` достаточно переставить ее указатель.

```
git merge new_feature
Updating a59d383..edb3a69
Fast-forward
 code.txt         | 1 +
 test_results.txt | 1 +
 2 files changed, 2 insertions(+)
```

5.6. Итоги раздела

- История может быть линейной и нелинейной
- Для слияния веток с общей историей используется перестановка указателя
- Слияние веток возможно через создание нелинейной истории
- Для слияния в линейную историю используется перенос коммитов
- Указатели веток можно переставлять вручную
- Команды для слияния веток и перемещения указателей:
    - `git merge` ↗ — слияние линейной истории и формирование нелинейной
    - `git rebase` ↗ — слияние в линейную историю копированием коммитов
    - `git reset` ↗ — перемещение указателя ветки

6. Конфликты при объединении изменений

Раздел познакомит с причинами возникновения и принципами разрешения конфликтов.

6.1. Выбор стратегии объединения изменений

Продолжим работу с демонстрационным репозиторием. Следующей задачей станет внесение исправлений из ветки `bugfix` в основную ветку `master`. Перед выполнением операций рассмотрим состояние истории с помощью команды `git log` ↗

```
$ git log --graph --all
* commit edb3a693dcc61305683da84390ffb4ca98bff606 (HEAD -> master, new_feature)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:57:57 2021 +0300
|
|     Add test results for the new feature
|
* commit 11281ba935ee627cf78678a503015770f5d2fec1
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:57:05 2021 +0300
|
|     Add new feature
|
* commit a59d38377a09be7cc4ee1a34063c23366d7d94df (origin/roadmap, roadmap)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Thu Jul 15 11:57:26 2021 +0300
|
|     Add roadmap
|
| * commit 13ce06113b312473249c16de9e5606e4f7b44afd (origin/new_feature)
| | Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| | Date:   Wed Jul 14 12:57:57 2021 +0300
| |
| |     Add test results for the new feature
| |
| * commit b7ab163c6c1aa710b06d1a9877126abe212783c6
|/  Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
|   Date:   Wed Jul 14 12:57:05 2021 +0300
|
|       Add new feature
|
* commit 7dde0effd0ff79b96f7973123e16738aed9734c1 (origin/master)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:01:33 2021 +0300
|
|     Add user feedback
|
| * commit 9546beea2635b1b54edf384efa038ff06042fd02 (origin/bugfix, bugfix)
| | Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| | Date:   Wed Jul 14 12:55:19 2021 +0300
| |
| |     Add bugfix test results
| |
| * commit 1033d95aa08809c37c15a43fc33d9fd7b9c42d25
|/  Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
|   Date:   Wed Jul 14 12:00:14 2021 +0300
|
|       Fix bug in code
|
* commit 7bc0cb6696c337ab502b5666c694a233eca90f0b (tag: v0.1)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 11:57:16 2021 +0300
|
|     Add test results
|
* commit a15efae1f812b5378c12fedb45de1d722f0949df
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:55:22 2021 +0300

    Add initial code
```

Attention

Какой из механизмов объединения изменений подойдет в данном случае?

- Слияние через перестановку указателей
- Слияние с нелинейной историей
- Перенос коммитов в конец линейной истории

Подходящие механизмы объединения изменений

- Слияние с нелинейной историей
- Перенос коммитов в конец линейной истории

Tip

Избегайте слияния с нелинейной историей, когда ветки сильно отличаются. Рекомендуется перенести коммиты поверх текущей версии ветки, в которую планируется вносить изменения.

Перенос изменений будет проведен в два этапа:

- Копирование коммитов из ветки `bugfix` поверх ветки `master`
- Перенос указателя ветки `master` на последний созданный коммит

6.2. Копирование коммитов и анализ конфликтов

Выполним команду `git rebase` ↗ в ветке `bugfix`, чтобы внести изменения из ветки `master`

```
$ git checkout bugfix
Switched to branch 'bugfix'
```

```
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Fix bug in code
Using index info to reconstruct a base tree...
M   code.txt
Falling back to patching base and 3-way merge...
Auto-merging code.txt
CONFLICT (content): Merge conflict in code.txt
error: Failed to merge in the changes.
Patch failed at 0001 Fix bug in code
hint: Use 'git am --show-current-patch' to see the failed patch
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
```

Attention

Почему возник конфликт при копировании коммитов?

<details>
<summary>Причина возникновения конфликта</summary>

Изменения, внесенные в файлы `code.txt` и `test_results.txt` в ветках `master` и `bugfix` отличаются между собой. Git не может однозначно определить, какие из изменений следует сохранить и указывает на необходимость ручного разрешения конфликтов.
</details>

Рассмотрим содержимое файла `code.txt`, на который ссылается вывод команды.

```
$ cat code.txt
This line is correct
<<<<<<< HEAD
This line contains a bug
=======
>>>>>>> Fix bug in code
This line is correct
```

Git демонстрирует разницу изменений, внесенных в рамках двух веток. В процессе операции `rebase` указатель `HEAD` перемещается на верхушку ветки, взятой за основу. В последнем коммите ветки `master` присутствует ошибка, а коммит из ветки `bugfix`, который применяется поверх него, содержит исправление.

Attention

Почему в файле `code.txt` оказалось две строчки кода, хотя было три?

Причина исчезновения строки кода

Обратимся к изменению, которое Git пытается применить из коммита с комментарием `Fix bug in code`, отобразив его содержимое командой `git am --show-current-patch` ↗

```
$ git am --show-current-patch
commit 1033d95aa08809c37c15a43fc33d9fd7b9c42d25
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 12:00:14 2021 +0300

    Fix bug in code

diff --git a/code.txt b/code.txt
index d66594c..4672aef 100644
--- a/code.txt
+++ b/code.txt
@@ -1,2 +1,2 @@
This line is correct
-This line contains a bug
+This line is correct
```

Команда пытается удалить одну строчку и добавить другую. Однако, строка с содержимым `This line is correct` уже привнесена другим коммитом. Полное совпадение строк из двух коммитов вызывает некорректное поведение механизмов проверки изменений. Подробнее об этих механизмах — в приложенной статье.

Springer — How different are different diff algorithms in Git? ↗

6.3. Ручное разрешение конфликтов

Для разрешения конфликта в файле `code.txt` необходимо привести его в желаемое состояние, после чего добавить файл в индекс и продолжить процедуру переноса коммитов. Таким образом в дерево перенесенного коммита попадет новая версия файла. Отредактируем файл и добавим его в индекс.

```
$ cat code.txt
This line is correct
This line is correct
This line is correct
```

```
$ git add code.txt
```

После этого продолжим операцию, выполнив команду `git rebase --continue` ↗

```
$ git rebase --continue
Applying: Fix bug in code
Applying: Add bugfix test results
Using index info to reconstruct a base tree...
M   test_results.txt
Falling back to patching base and 3-way merge...
Auto-merging test_results.txt
CONFLICT (content): Merge conflict in test_results.txt
error: Failed to merge in the changes.
Patch failed at 0002 Add bugfix test results
hint: Use 'git am --show-current-patch' to see the failed patch
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
```

Аналогичным способом разрешим конфликт в файле `test_results.txt`

```
$ cat test_results.txt
First line of code is correct
<<<<<<< HEAD
Second line of code contains bugs
Third line of code is correct
=======
Second line of code is correct
>>>>>>> Add bugfix test results
```

Файл `test_results.txt` не содержит одинаковых строк и отображает разницу корректно. Однако, желаемый результат все еще необходимо сформировать вручную.

Tip

Среды разработки позволяют выбрать желаемое изменение в один клик.

```
$ cat test_results.txt
First line of code is correct
Second line of code is correct
Third line of code is correct
```

```
$ git add test_results.txt
```

```
$ git rebase --continue
Applying: Add bugfix test results
```

На этом процесс копирования коммитов завершен. Рассмотрим историю коммитов.

```
$ git log --graph
* commit 16ef4b7f4756dacdb638f20afefa2679f3489634 (HEAD -> bugfix)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:55:19 2021 +0300
|
|     Add bugfix test results
|
* commit 0c867f5c1a04da3fc03e6e5fd065ff7c163945b6
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:00:14 2021 +0300
|
|     Fix bug in code
|
* commit edb3a693dcc61305683da84390ffb4ca98bff606 (new_feature, master)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:57:57 2021 +0300
|
|     Add test results for the new feature
|
* commit 11281ba935ee627cf78678a503015770f5d2fec1
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:57:05 2021 +0300
|
|     Add new feature
|
* commit a59d38377a09be7cc4ee1a34063c23366d7d94df (origin/roadmap, roadmap)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Thu Jul 15 11:57:26 2021 +0300
|
|     Add roadmap
|
* commit 7dde0effd0ff79b96f7973123e16738aed9734c1 (origin/master)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 12:01:33 2021 +0300
|
|     Add user feedback
|
* commit 7bc0cb6696c337ab502b5666c694a233eca90f0b (tag: v0.1)
| Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
| Date:   Wed Jul 14 11:57:16 2021 +0300
|
|     Add test results
|
* commit a15efae1f812b5378c12fedb45de1d722f0949df
Author: Aleksei Sokolov <Aleksei_Sokolov2@epam.com>
Date:   Wed Jul 14 11:55:22 2021 +0300

    Add initial code
```

История коммитов в ветке `bugfix` теперь линейна и содержит все изменения из ветки `master`. Для переноса изменений в ветку `master` достаточно переставить ее указатель.

```
$ git checkout master
Switched to branch 'master'
```

```
$ git merge
Updating edb3a69..16ef4b7
Fast-forward
 code.txt         | 2 +-
 test_results.txt | 2 +-
 2 files changed, 2 insertions(+), 2 deletions(-)
```

6.4. Итоги раздела

- Конфликты возникают при объединении изменений в одних и тех же сегментах файлов
- Разрешение конфликтов всегда производится в ручном режиме
- В режиме нелинейной истории все конфликты будут разрешаться в одном коммите
- Команды для работы с патчами:
    - `git am` ↗ — просмотр и применение патчей

7. Стратегии ветвления

Раздел познакомит с различными стратегиями ветвления и сферами их применения.

7.1. Введение в стратегии ветвления

Выбор стратегии ветвления для репозитория обуславливается следующими факторами:

- Линейность истории и дублируемость коммитов
- Разделение кода в разработке и развертываемого кода
- Удобство построения процессов непрерывной доставки

В зависимости от предпочтений и требований команд разработки или эксплуатации может быть выбрана та или иная стратегия ветвления. В данном разделе приведены три наиболее распространенных. Любая из них является ориентиром и может быть модифицирована.

7.2. Git flow

Git flow является одной из первых документированных стратегий ветвления. Она подразумевает разделение веток на 5 основных типов:

- `main` — готовый к поставке код
- `develop` — актуальное состояния процесса разработки
- `feature` — код отдельно разрабатываемого функционала
- `release` — слепок кода для каждой поставленно версии
- `hotfix` — временное хранилище оперативных исправлений

Ветки с новым функционалом попадают в `develop`, где происходит интеграционное тестирование. В определенный момент на основе актуального состояния ветки `develop` создается ветка `release`, содержащая готовый к поставке код, который после внесения необходимых корректировок попадает в ветку `main`. Оперативные исправления основываются на ветке `main` и по готовности применяются к ней самой и к ветке `develop`

Vincent Driessen - A successful Git branching model ↗

7.3. Github flow

Github flow упрощает предыдущую схему до двух типов веток:

- `main` — готовый к поставке код
- `feature` — отдельно разрабатываемый набор изменений

Основная идея этой стратегии заключается в том, что любой новый код основывается на наиболее актуальной версии существующего, содержимое ветки `main` всегда готово к поставке, а содержимое веток `feature` постоянно тестируется, переносится в `master` по готовности и сразу развертывается. Подобный механизм удобен для небольших команд и проектов с низкой вероятностью возникновения проблем на этапе интеграции нового кода и отсутствием необходимости поддерживать несколько версий кода одновременно.

Scott Chacon — GitHub Flow: The best way to use Git and GitHub ↗
Github Guides — Understanding the GitHub flow ↗

7.4. Gitlab flow

Основное отличие Gitlab flow от Github flow заключается в наличии отдельных веток для различных окружений и релизов. Эта особенность позволяет упростить процесс доставки на несколько окружений и поддержки различных версий кода в рамках цикла поставок.

GitLab Docs — Introduction to GitLab Flow ↗

7.5. Итоги раздела

- Стратегии ветвления предназначены для формализации принципов работы
- Выбор стратегии обуславливается спецификой команды, проекта, инструментов
- Стратегии могут меняться и подстраиваться под различные требования

8. Работа с удаленными репозиториями

Раздел познакомит с операциями синхронизации данных с удаленными репозиториями и основными принципами обмена изменениями.

8.1. Клонирование удаленного репозитория

Наиболее часто работа происходит с уже созданными репозиториями. Для работы с кодом из удаленного репозитория необходимо скопировать его на свой компьютер с помощью команды `git clone` ↗. Данная команда загружает из удаленного репозитория все объекты и указатели. Рассмотрим процесс клонирования демонстрационного репозитория.

Ссылка на демонстрационный репозиторий

https://github.com/aleksei-sokolov/devops-school-git

```
$ mkdir /tmp/git-clone
$ cd /tmp/git-clone
```

```
$ git clone https://github.com/aleksei-sokolov/devops-school-git .
Cloning into '.'...
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 24 (delta 6), reused 20 (delta 2), pack-reused 0
Unpacking objects: 100% (24/24), done.
```

Рассмотрим перечень объектов в директории.

```
$ tree -aF -I '*sample|objects'
.
├── .git/
│   ├── HEAD
│   ├── config
│   ├── description
│   ├── hooks/
│   ├── index
│   ├── info/
│   │   └── exclude
│   ├── logs/
│   │   ├── HEAD
│   │   └── refs/
│   │       ├── heads/
│   │       │   └── master
│   │       └── remotes/
│   │           └── origin/
│   │               └── HEAD
│   ├── packed-refs
│   └── refs/
│       ├── heads/
│       │   └── master
│       ├── remotes/
│       │   └── origin/
│       │       └── HEAD
│       └── tags/
├── code.txt
├── test_results.txt
└── user_feedback.txt
```

Помимо указателя на ветку `master` в директории `.git` также появилась поддиректория `.git/refs/remotes/origin/`, где `origin` — наименование удаленного репозитория по умолчанию. В этой директории содержится указатель на ветку по умолчанию.

```
$ cat .git/refs/remotes/origin/HEAD
ref: refs/remotes/origin/master
```

Указатель `refs/remotes/origin/master` отсутствует на файловой системе. Он расположен в файле `packed-refs`, который объединяет в себе скачанные из удаленного репозитория указатели. Просмотреть перечень хранимых указателей позволяет команда `git show-ref` ↗

```
$ git show-ref
7dde0effd0ff79b96f7973123e16738aed9734c1 refs/heads/master
7dde0effd0ff79b96f7973123e16738aed9734c1 refs/remotes/origin/HEAD
9546beea2635b1b54edf384efa038ff06042fd02 refs/remotes/origin/bugfix
7dde0effd0ff79b96f7973123e16738aed9734c1 refs/remotes/origin/master
13ce06113b312473249c16de9e5606e4f7b44afd refs/remotes/origin/new_feature
a59d38377a09be7cc4ee1a34063c23366d7d94df refs/remotes/origin/roadmap
```

Tip

Автоматически создается только копия указателя на ветку по умолчанию.

С момента клонирования репозитория информация о нем хранится в `.git/config`

```
$ grep 'remote "origin"' .git/config -A2
[remote "origin"]
        url = https://github.com/aleksei-sokolov/devops-school-git.git
        fetch = +refs/heads/*:refs/remotes/origin/*
```

Просмотреть список удаленных репозиториев можно с помощью команды `git remote -v` ↗

```
$ git remote -v
origin      https://github.com/aleksei-sokolov/devops-school-git.git (fetch)
origin      https://github.com/aleksei-sokolov/devops-school-git.git (push)
```

Tip

Получение и отправка могут осуществляться с использованием разных репозиториев.

8.2. Получение данных из удаленного репозитория

Для обновления локального репозитория необходимо получить из удаленного изменения указателей и добавленные объекты. Для этого используется команда `git fetch` ↗

```
$ git fetch
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 12 (delta 2), reused 12 (delta 2), pack-reused 0
Unpacking objects: 100% (12/12), done.
From https://github.com/aleksei-sokolov/devops-school-git
    7dde0ef..16ef4b7  master     -> origin/master
```

Вывод команды предоставляет справочную информацию о полученных обновлениях.

8.3. Применение правок из удаленного репозитория

После получения обновлений применение их к локальной ветке сводится к одной из операций объединения изменений. При отсутствии локальных изменений в ветке актуализировать ее можно путем перенаправления указателя с помощью команды `git merge` ↗, используя в качестве источника указатель из удаленного репозитория.

```
$ git merge origin/master
Updating 7dde0ef..16ef4b7
Fast-forward
 code.txt         | 3 ++-
 roadmap.txt      | 1 +
 test_results.txt | 3 ++-
 3 files changed, 5 insertions(+), 2 deletions(-)
 create mode 100644 roadmap.txt
```

Tip

При отсутствии локальных изменений операции получения и применения изменений из удаленного репозитория можно объединить, использовав команду `git pull` ↗

Attention

Что будет, если выполнить команду `git pull` ↗ при наличии локальных изменений?

Последствия слияния локальных и удаленных изменений

Как и в случае обычного слияния, будет создан коммит, объединяющий две ветви истории. Это может стать проблемой, если в репозитории принято линейное формирование истории. Для объединения локальных и удаленных изменений в линейную историю следует воспользоваться командой `git rebase` ↗

8.4. Отправка данных в удаленный репозиторий

Для отправки изменений загруженной ветки используется команда `git push` ↗

```
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 585 bytes | 585.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/aleksei-sokolov/devops-school-git.git
    16ef4b7..89ea13f  master -> master
```

Tip

Основная ветка репозитория зачастую запрещает прямые обновления.

Для загрузки в удаленный репозиторий локально созданной ветки необходимо добавить к запуску команды ключ, указывающий имя удаленного репозитория и целевой ветки в нем.

```
$ git push --set-upstream origin master-patched
```

Tip

Данную команду можно использовать для загрузки ветки под другим именем.

8.5. Ручное добавление удаленного репозитория

При необходимости загрузить содержимое локального репозитория в пустой удаленный, вместо клонирования выполняется добавление ссылки на удаленный репозиторий в локальном. Для этого используется команда `git remote add` ↗

```
$ git remote add origin https://github.com/aleksei-sokolov/devops-school-git-
homework.git
```

Первый ключ задает имя ссылки на удаленный репозиторий, а второй — его адрес.

Tip

Локальный репозиторий может иметь несколько ссылок на различные удаленные.

В пустом репозитории отсутствуют указатели веток, поэтому загрузка данных выполняется с указанием имени ссылки на удаленный репозиторий и желаемого имени ветки в нем.

8.6. Итоги раздела

- Локальный репозиторий может содержать ссылки на несколько удаленных
- Загрузка репозитория подразумевает получение всех объектов и ссылок
- При отправке данных передаются только создаваемые объекты и обновление указателя
- Команды для работы с удаленными репозиториями:
    - `git remote` ↗ — просмотр и редактирование списка удаленных репозиториев
    - `git fetch` ↗ — получение обновлений из удаленного репозитория
    - `git pull` ↗ — получение и применение обновлений из удаленного репозитория
    - `git push` ↗ — отправка изменений в удаленный репозиторий