

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 вершины (`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 эту 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 выполнения команды?

Объекты, появляющиеся при создании ветки ^


- 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` [↗](#)

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

