Лабораторная работа 2
Ветвление с GIT

Ветвление в Git

Почти каждая система контроля версий (СКВ) в какой-то форме поддерживает ветвление. Используя ветвление, Вы отклоняетесь от основной линии разработки и продолжаете работу независимо от неё, не вмешиваясь в основную линию. Во многих СКВ создание веток — это очень затратный процесс, часто требующий создания новой копии директории, что может занять много времени для большого проекта.

Ветвление Git очень легковесно: операция создания ветки выполняется почти мгновенно, переключение между ветками туда-сюда, обычно, также быстро. В отличие от многих других СКВ, Git поощряет процесс работы, при котором ветвление и слияние выполняется часто, даже по несколько раз в день. Понимание и владение этой функциональностью дает вам уникальный и мощный инструмент, который может полностью изменить привычный процесс разработки.

Ветка в Git — это простой перемещаемый указатель на один из коммитов. По умолчанию, имя основной ветки в Git — `master`. Как только вы начнёте создавать коммиты, ветка `master` будет всегда указывать на последний коммит. Каждый раз при создании коммита указатель ветки `master` будет передвигаться на следующий коммит автоматически.

> Ветка `master` в Git — это не какая-то особенная ветка. Она точно такая же, как и все остальные ветки. Она существует почти во всех репозиториях только лишь потому, что её создаёт команда `git init`, а большинство людей не меняют её название.

Создание новой ветки

При создании ветки создаётся новый указатель для дальнейшего перемещения. Допустим вы хотите создать новую ветку с именем `testing`. Вы можете это сделать командой `git branch`:

```
$ git branch testing
```

В результате создаётся новый указатель на текущий коммит.

Git использует специальный указатель `HEAD` - это указатель на текущую локальную ветку. Увидеть куда указывают указатели веток можно при помощи команды `git log` с опцией `--decorate`.

> Команда `git branch` только создаёт новую ветку, но не переключает на неё.

Переключение веток

Для переключения на существующую ветку выполните команду `git checkout`. Давайте переключимся на ветку `testing`:

```
$ git checkout testing
```

В результате указатель `HEAD` переместится на ветку `testing`.

Давайте сделаем ещё один коммит:

```
$ vim test.txt
$ git commit -a -m 'Изменен файл test.txt'
```

Возникнет следующая ситуация: указатель на ветку `testing` переместился вперёд, а `master` указывает на тот же коммит, где вы были до переключения веток командой `git checkout`. Давайте переключимся назад на ветку master:

```
$ git checkout master
```

Эта команда сделала две вещи: переместила указатель `HEAD` назад на ветку `master` и вернула файлы в рабочем каталоге в то состояние, на снимок которого указывает `master`. Это также означает, что все вносимые с этого момента изменения будут относиться к старой версии проекта. Другими словами, вы откатили все изменения ветки `testing` и можете продолжать в другом направлении.

> ***Переключение веток меняет файлы в рабочем каталоге***
> Важно запомнить, что при переключении веток в Git происходит изменение файлов в рабочей директории. Если вы переключаетесь на старую ветку, то рабочий каталог будет выглядеть так же, как выглядел на момент последнего коммита в ту ветку.

Давайте сделаем еще несколько изменений и создадим очередной коммит:

```
$ vim test.txt
$ git commit -a -m 'Другие изменения файла test.txt'
```

Теперь история проекта разошлась. Вы создали ветку и переключились на нее, поработали, а затем вернулись в основную ветку и поработали в ней. Эти изменения изолированы друг от друга: вы можете свободно переключаться туда и обратно, а когда понадобится — объединить их.

Все описанные действия можно визуализировать с помощью команды `git log`. Для отображения истории коммитов, текущего положения указателей веток и истории ветвления выполните команду:

```
$ git log --oneline --decorate --graph --all
```

> При переключении веток можно использовать опцию `-b`, так вы одновременно создадите новую ветку, если её не было: `git checkout -b <newbranchname>`.

Слияние веток

Простое слияние fast-forward

Предположим, вы работаете над проектом и уже имеете несколько коммитов в ветке `master`. Вам предстоит написать новую фичу для проекта и вы решаете создать для этого новую ветку:

```
$ git checkout -b feature
```

Поработав над фичей и сделав несколько коммитов в этой ветке вы решаете слить эти изменения в ветку `master`. Для этого необходимо переключится на ветку `master` и выполнить слияние командой `git merge`:

```
$ git checkout master
$ git merge feature
```

Это достаточно простая ситуация, так как в ветке `master` не было никаких изменений. Git просто переместил указатель ветки вперед, потому что коммит, на который указывает слитая ветка `feature`, был прямым потомком коммита, на котором вы находились до этого. Другими словами, если коммит сливается с тем, до которого можно добраться двигаясь по истории прямо, Git упрощает слияние просто перенося указатель ветки вперед, так как нет расхождений в изменениях. Это называется `fast-forward`, что подтверждается сообщением в консоли.

Если ветка больше не нужна, её можно удалить командой:

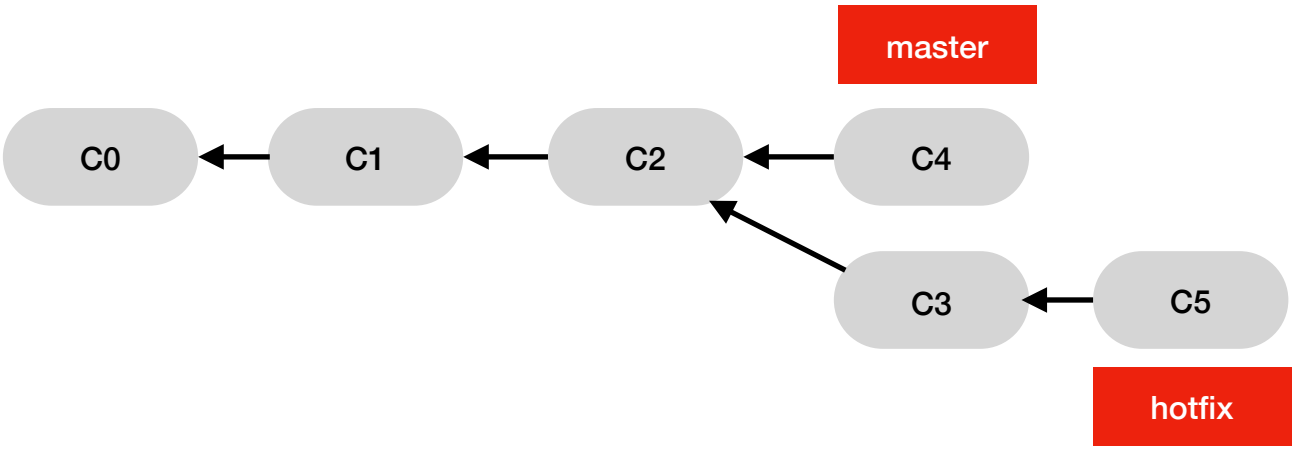
```
$ git branch -d feature
```

Трёхстороннее слияние

Предположим, вы работаете над проектом и уже имеете несколько коммитов в ветке `master`. От руководства вы получаете информацию о том, что в коде вашего проекта найден баг, для его устранения вы создаёте ветку с названием `hotfix`:

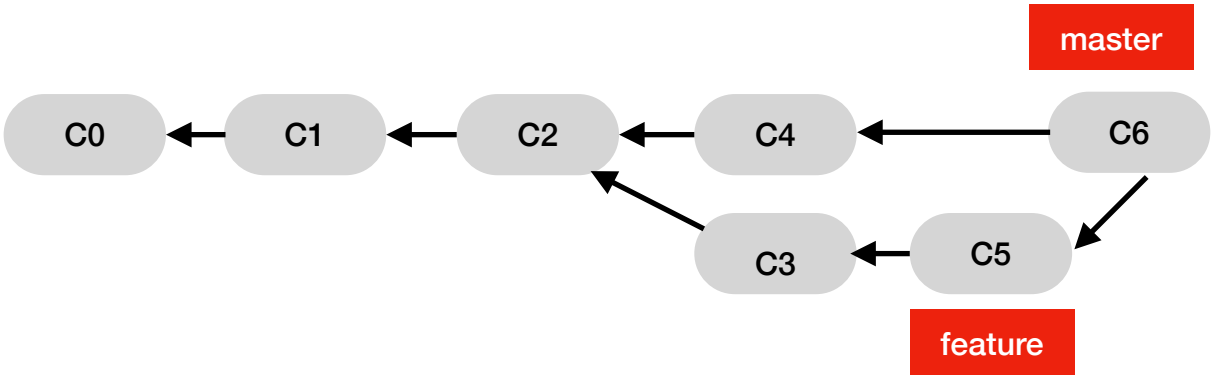
```
$ git checkout -b hotfix
```

В процессе устранения бага вы или ваши коллеги сделали несколько коммитов в ветке `master`, таким образом что ваша ветка `hotfix` унаследована от более раннего коммита:



Использование трех снимков слияния

Вместо того, чтобы просто передвинуть указатель ветки вперёд, Git создаёт новый результирующий снимок трёхстороннего слияния, а затем автоматически делает коммит. Этот особый коммит называют коммитом слияния, так как у него более одного предка:



Коммит слияния

Конфликты слияния

Иногда процесс не проходит гладко. Если вы изменили одну и ту же часть одного и того же файла по-разному в двух объединяемых ветках, Git не сможет их чисто объединить. Если ваше исправление ошибки в ветке `hotfix` потребовало изменить ту же часть файла что и `feature`, вы получите примерно такое сообщение о конфликте слияния:

```
Automatic merge failed; fix conflicts and then commit the result.
```

Git не создал коммит слияния автоматически. Он остановил процесс до тех пор, пока вы не разрешите конфликт. Чтобы в любой момент после появления конфликта увидеть, какие файлы не объединены, вы можете запустить `git status`.

Всё, где есть неразрешённые конфликты слияния, перечисляется как неслитое. В конфликтующие файлы Git добавляет специальные маркеры конфликтов, чтобы вы могли исправить их вручную.

Разрешив каждый конфликт во всех файлах, запустите `git add` для каждого файла, чтобы отметить конфликт как решённый. Добавление файла в индекс означает для Git, что все конфликты в нём исправлены.

Если вы хотите использовать графический инструмент для разрешения конфликтов, можно запустить `git mergetool`, которое проведет вас по всем конфликтам.

Если вы убедились, что все файлы, где были конфликты, добавлены в индекс — выполните команду `git commit` для создания коммита слияния.

Задание

Задание 1 - Простое слияние

Создайте проект, настройте git-репозиторий, добавьте файлы в основную ветку, добавьте ветку `new-feature`, измените или добавьте файлы, выполните слияние.

Создаем рабочую директорию проекта:

```
$ mkdir git-branch-demo
```

Переходим в рабочую директорию:

```
$ cd git-branch-demo
```

Создаем репозиторий в директории:

```
$ git init
```

Создайте файл `readme.md` и внесите в него любую информацию:

```
$ vim readme.md
```

Индексируем все существующие файлы проекта (добавляем в репозиторий):

```
$ git add .
```

Создаем инициализирующий коммит:

```
$ git commit -m «initial commit»
```

> *Последние два действия можно сделать в один шаг командой:* `git commit -a -m «initial commit»`

Создаем новую ветку:

```
$ git branch new-feature
```

Переключаемся в новую ветку:

```
$ git checkout new-feature
```

> *Последние два действия можно сделать в один шаг командой:* `git checkout -b new-feature`

После непосредственной работы с кодом или текстом индексируем внесенные изменения:

```
$ git add .
```

Совершаем коммит:

```
$ git commit -m «Done with the new feature»
```

Переключаемся в основную ветку:

```
$ git checkout master
```

Смотрим отличия между последним коммитом активной ветки и последним коммитом экспериментальной:

```
$ git diff HEAD new-feature
```

Проводим слияние:

```
$ git merge new-feature
```

Если не было никаких конфликтов, удаляем ненужную больше ветку:

```
$ git branch -d new-feature
```

Оценим проведенную за последний день работу:

```
$ git log --since=«1 day»
```

Задание 2 - трёхстороннее слияние

Сделайте изменения в дополнительной и основной ветке, затем произведите слияние.

Создаём новую ветку `hotfix` и переключаемся на неё:

```
$ git branch -b hotfix
```

Создаём новый файл или редактируем существующий, например `readme.md`:

```
$ vim readme.md
```

Выполняем коммит изменений:

```
$ git commit -a -m «changes in readme»
```

Возвращаемся в ветку `master` и создаем новый файл:

```
$ git checkout master
$ vim license.md
```

Выполняем коммит изменений:

```
$ git commit -a -m «created license»
```

Производим слияние `master` и `hotfix`:

```
$ git merge hotfix
```

Оценим проведенную за последний день работу:

```
$ git log --since=«1 day»
```

Задание 3 - конфликт слияния

Самостоятельно сделайте две новые дополнительные ветки, в обоих ветках внесите разные изменения в один и тот же файл (например `readme.md`), произведя коммит в каждой, выполните слияние каждой из веток в ветку `master`, обработайте различия и сделайте коммит слияния.