? ????????? ? ???? ??????

Почти каждая система контроля версий в той или иной форме поддерживает ветвление. Используя ветвление, Вы отклоняетесь от основной линии разработки и продолжаете работу независимо от неё, не вмешиваясь в основную линию. Во многих системах контроля версий создание веток — это очень затратный процесс, часто требующий создания новой копии каталога с исходным кодом, что может занять много времени для большого проекта.

Некоторые люди, говоря о модели ветвления Git, называют её «киллер-фича», что выгодно выделяет Git на фоне остальных систем контроля версий. Что в ней такого особенного? Ветвление Git очень легковесно: операция создания ветки выполняется почти мгновенно, переключение между ветками туда-сюда, обычно, также быстро. В отличие от многих других систем контроля версий, 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.

? ????????? ? ???? ??????

Для точного понимания механизма ветвлений, необходимо вернуться назад и изучить то, как Git хранит данные.

Как вы можете помнить из Что такое Git?, Git не хранит данные в виде последовательности изменений, он использует набор снимков (snapshot).

Когда вы делаете коммит, Git сохраняет его в виде объекта, который содержит указатель на снимок (snapshot) подготовленных данных. Этот объект так же содержит имя автора и email, сообщение и указатель на коммит или коммиты непосредственно предшествующие данному (его родителей): отсутствие родителя для первоначального коммита, один родитель для обычного коммита, и несколько родителей для результатов слияния двух и более веток.

Предположим, у вас есть каталог с тремя файлами и вы добавляете их все в индекс и создаёте коммит. Во время индексации вычисляется контрольная сумма каждого файла (SHA-1 как мы узнали из Что такое Git?), затем каждый файл сохраняется в репозиторий (Git называет такой файл *блоб* — большой бинарный объект), а контрольная сумма попадёт в индекс:

```
$ git add README test.rb LICENSE
$ git commit -m 'Initial commit'
```

Когда вы создаёте коммит командой `git commit`, Git вычисляет контрольные суммы каждого подкаталога (в нашем случае, только основной каталог проекта) и сохраняет его в репозитории как объект дерева каталогов. Затем Git создаёт объект коммита с метаданными и указателем на основное дерево проекта для возможности воссоздать этот снимок в случае необходимости.

Ваш репозиторий Git теперь хранит пять объектов: три блоб объекта (по одному на каждый файл), объект *дерева* каталогов, содержащий список файлов и соответствующих им блобов, а так же объект *коммита*, содержащий метаданные и указатель на объект дерева каталогов.

Коммит и его дерево
Рисунок 9. Коммит и его дерево

Если вы сделаете изменения и создадите ещё один коммит, то он будет содержать указатель на предыдущий коммит.

Коммит и его родители
Рисунок 10. Коммит и его родители

Ветка в Git — это простой перемещаемый указатель на один из таких 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.

Примечание	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.

Ветка и история коммитов
Рисунок 11. Ветка и история коммитов

???????? ????? ?????

Что же на самом деле происходит при создании ветки? Всего лишь 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.

Две ветки указывают на одну и ту же последовательность коммитов

Рисунок 12. Две ветки указывают на одну и ту же последовательность коммитов

Как Git определяет, в какой ветке вы находитесь? Он хранит специальный указатель `HEAD`. Имейте ввиду, что в Git концепция `HEAD` значительно отличается от других систем контроля версий, которые вы могли использовать раньше (Subversion или CVS). В Git — это указатель на текущую локальную ветку. В нашем случае мы всё ещё находимся в ветке `master`. Команда `git branch` только *создаёт* новую ветку, но не переключает на неё.

HEAD указывает на ветку
Рисунок 13. HEAD указывает на ветку

Вы можете легко это увидеть при помощи простой команды `git log`, которая покажет вам куда указывают указатели веток. Эта опция называется `--decorate`.

```
$ git log --oneline --decorate
f30ab (HEAD -> master, testing) Add feature #32 - ability to add new formats to the central
interface
34ac2 Fix bug #1328 - stack overflow under certain conditions
98ca9 Initial commit
```

Здесь можно увидеть указывающие на коммит `f30ab` ветки: `master` и `testing`.

?????????????? ?????

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`.

HEAD указывает на текущую ветку
Рисунок 14. HEAD указывает на текущую ветку

Какой в этом смысл? Давайте сделаем ещё один коммит:

```
$ vim test.rb
$ git commit -a -m 'made a change'
```

Указатель на ветку HEAD переместился вперёд после коммита
Рисунок 15. Указатель на ветку HEAD переместился вперёд после коммита

Интересная 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
```

Примечание	`git log` не показывает все ветки по умолчанию Если выполнить команду `git log` прямо сейчас, то в её выводе только что созданная ветка «testing» фигурировать не будет. Ветка никуда не исчезла; просто Git не знает, что именно она вас интересует, и выводит наиболее полезную по его мнению информацию. Другими словами, по умолчанию `git log` отобразит историю коммитов только для текущей ветки. Для просмотра истории коммитов другой ветки необходимо явно указать её имя: `git log testing` Чтобы посмотреть историю по всем веткам — выполните команду с дополнительным флагом: `git log --all`.

HEAD перемещается когда вы делаете checkout
Рисунок 16. HEAD перемещается когда вы делаете checkout

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.

Примечание	Переключение веток меняет файлы в рабочем каталоге Важно запомнить, что при переключении веток в Git происходит изменение файлов в рабочем каталоге. Если вы переключаетесь на старую ветку, то рабочий каталог будет выглядеть так же, как выглядел на момент последнего коммита в ту ветку. Если Git по каким-то причинам не может этого сделать — он не позволит вам переключиться вообще.

Давайте сделаем ещё несколько изменений и создадим очередной коммит:

```
$ vim test.rb
$ git commit -a -m 'made other changes'
```

Теперь история вашего проекта разошлась (см Разветвлённая история). Вы создали ветку и переключились на неё, 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. И всё это делается простыми командами: `branch`, `checkout` и `commit`.

Разветвлённая история
Рисунок 17. Разветвлённая история

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`.

```
$ git log --oneline --decorate --graph --all
* c2b9e (HEAD, master) Made other changes
| * 87ab2 (testing) Made a change
|/
* f30ab Add feature #32 - ability to add new formats to the central interface
* 34ac2 Fix bug #1328 - stack overflow under certain conditions
* 98ca9 initial commit of my project
```

Ветка в Git — это простой файл, содержащий 40 символов контрольной суммы SHA-1 коммита, на который она указывает; поэтому операции с ветками являются дешёвыми с точки зрения потребления ресурсов или времени. Создание новой ветки в Git происходит так же быстро и просто как запись 41 байта в файл (40 знаков и перевод строки).

Это принципиально отличает процесс ветвления в Git от более старых систем контроля версий, где все файлы проекта копируются в другой подкаталог. В зависимости от размера проекта, операции ветвления в таких системах могут занимать секунды или даже минуты, когда в Git эти операции мгновенны. Поскольку при коммите мы сохраняем указатель на родительский коммит, то поиск подходящей базы для слияния веток делается автоматически и, в большинстве случаев, очень прост. Эти возможности побуждают разработчиков чаще создавать и использовать ветки.

Давайте посмотрим, почему и вам имеет смысл делать так же.

Примечание	Одновременное создание новой ветки и переключение на неё Как правило, при создании новой ветки вы хотите сразу на неё переключиться — это можно сделать используя команду `git checkout -b <newbranchname>`.

Примечание	Начиная с Git версии 2.23, вы можете использовать `git switch` вместо `git checkout`, чтобы: • Переключиться на существующую ветку: `git switch testing-branch`. • Создать новую ветку и переключиться на неё: `git switch -c new-branch`. Флаг `-c` означает создание, но также можно использовать полный формат:` --create`. • Вернуться к предыдущей извлечённой ветке: `git switch -`.

Revision #1
Created 2 May 2024 07:19:32 by Антон Сергеевич Абраменко
Updated 2 May 2024 07:20:32 by Антон Сергеевич Абраменко