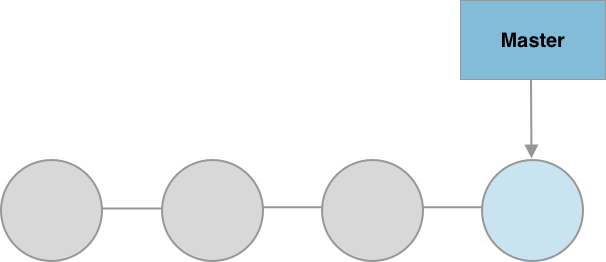
**Главная шпаргалка**

**https://training.github.com/downloads/ru/github-git-cheat-sheet/**

**ветка** — это всего лишь указатель на коммит, не меняющий репозиторий. Когда вы делаете коммиты, не создавая и не меняя ветки, то работаете в ветке *master*. Она указывает на последний сделанный коммит и при каждом новом коммите сдвигается вперёд автоматически. Выглядит это примерно так:

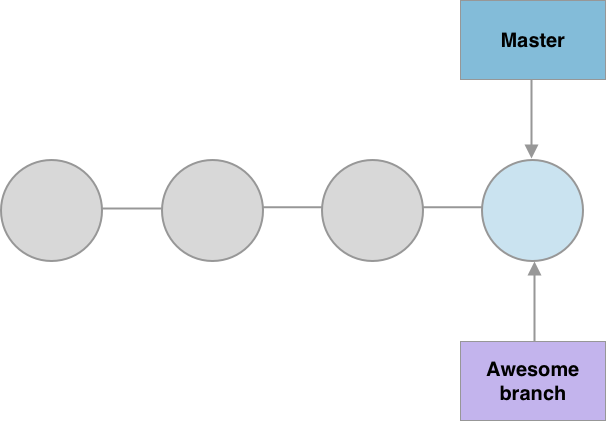


Давайте создадим новую ветку. Для этого выполним в*Git*-репозитории следующую команду:

git branch awesome-branch

В приложениях для *Git* с графическим интерфейсом это тоже можно сделать. Например, если вы используете *GitHub Desktop*, то можете воспользоваться этой [>инструкцией](https://help.github.com/en/desktop/contributing-to-projects/creating-a-branch-for-your-work).

После того как вы создали новую ветку, состояние репозитория выглядит следующим образом:

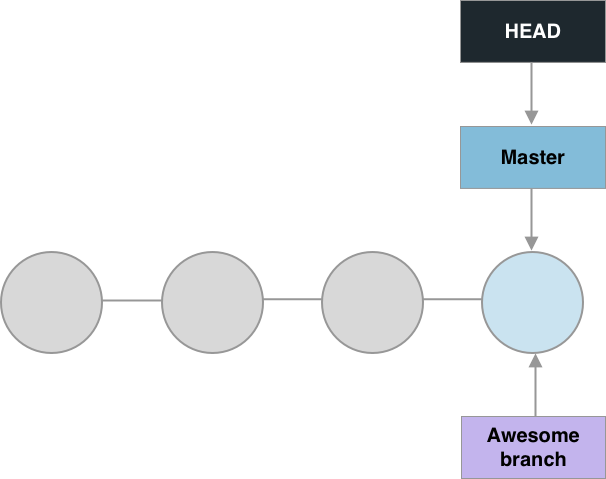


При этом история репозитория никак не изменилась. Произошло только создание нового указателя на уже существующий коммит.

Но если веток несколько и можно с ними по-разному взаимодействовать, то как *Git* узнаёт, в какую ветку необходимо вносить изменения? Есть какая-то «текущая»?

Да, действительно есть. *Git* хранит специальный указатель, который называется **HEAD** (верхушка). *HEAD* указывает на локальную ветку, на которой вы находитесь.

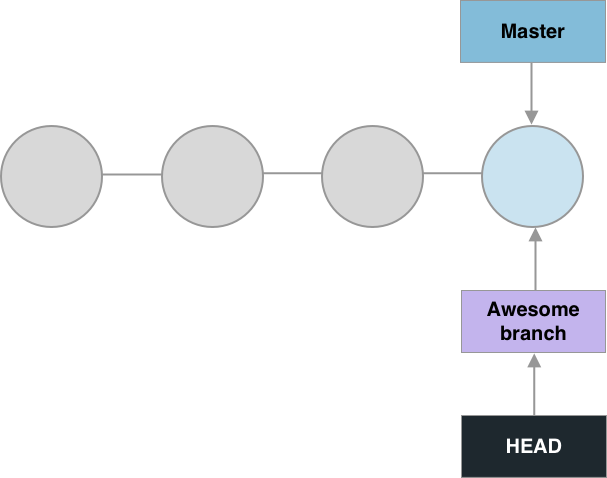
В данный момент вы всё ещё находитесь на ветке *master*. А как же *awesome-branch*? Команда git branch только создала новую ветку, она не переключила вас на неё. Так что, учитывая новое знание о *HEAD*, состояние системы на данный момент, на самом деле, выглядит так:



Чтобы перейти на существующую ветку, необходимо выполнить следующую команду:

git checkout awesome-branch

Как вы могли догадаться, после переключения *HEAD* указывает на другую ветку:



Команда git checkout позволяет вам перемещаться между ветками, которые уже были созданы. Когда вы переключаетесь на ветку, то файлы в вашей рабочей директории меняются в соответствии с той версией, на которую указывает выбранная ветка. Все новые коммиты будут теперь добавляться в текущую ветку, пока вы снова не переключитесь на другую.

\*Если вы ошиблись в названии, например, допустили опечатку, вы можете изменить название ветки с помощью команды: git branch -m старое-имя-ветки новое-имя-ветки.

Чтобы внести изменения из одной ветки в другую, нужно выполнить команду **git merge**. По-русски эти команды часто называют «вмержить», «смержить» или «сделать мерж», хотя, конечно, это всё производные от английских терминов. Правильно на русский язык merge можно перевести как «слияние», поэтому говорят также «слияние веток».

**Слияние**— это способ соединить обратно раздвоенную историю. Команда git merge позволяет взять независимые части разработки и интегрировать их в одну ветку.

В нашем же случае нам надо наоборот в ветку master влить black-friday. Как это можно сделать?

Для этого нам надо сначала переключиться на ту ветку, в которую мы хотим вливать, а затем уже «мержить».

git checkout master

git merge black-friday

Если все изменения уже находятся в главной ветке и Чёрная пятница прошла, нужна ли отдельная ветка, которая указывает на последний коммит этих изменений?

Чаще всего нет: если в ветке ничего не разрабатывается, то она продолжает указывать на один и тот же коммит, который с течением разработки остаётся где-то далеко в истории коммитов. Если вам когда-нибудь понадобится вернуться к этой версии или посмотреть, какие изменения были тогда внесены, для этого будет достаточно воспользоваться механизмом коммитов.

Значит, её нужно удалять, потому что незачем хранить лишнее. Чтобы удалить ветку, достаточно выполнить команду:

git branch -d black-friday

Как только вы перестали работать с веткой и слили все изменения с основной, можете удалять ветку без потери истории изменений. Обычно ветку оставляют до тех пор, пока не удостоверятся, что новый функционал работает как ожидалось, а затем сразу удаляют.

1. Для проверки состояния выполните команды git status и git log. Последняя показывает историю изменений: перечисляет коммиты, сделанные в репозитории, в обратном хронологическом порядке. Последние коммиты находятся вверху.
2. Команду git branch --list. Она выведет список веток, которые существуют в вашем локальном репозитории. Текущая ветка *master* будет выделена зелёным цветом.
3. Команду git branch --list -v . Вы увидите информацию о последних коммитах каждой из веток. Так как коммитов в ветку *testbranch00* ещё не было, то она указывает на коммит в *master*, от которого была создана.
4. Команда git checkout переключит нас на созданную ветку. Выполняя команду git checkout, укажите после неё имя ветки, на которую вы хотите переключиться. Например, git checkout testbranch00. Вы получите сообщение о переключении на ветку *testbranch00*.
5. Выполните команду git status. Вы увидите сообщение о том, что вы находитесь на ветке *testbranch00*.

git checkout -b [yourbranchname] - создать новую ветку и переключиться на неё с помощью одной команды.

Перемещение на один коммит назад ^

Перемещение на несколько коммитов назад ~<num>

git checkout HEAD^

Оператор "~"

Предположим, нужно переместиться на много шагов назад по дереву. Было бы неудобно печатать ^ несколько раз (или несколько десятков раз), так что Git поддерживает также оператор тильда (~).

К тильде (опционально) можно добавить количество родительских коммитов, через которые нужно пройти.

git checkout HEAD~4

Одна из наиболее распространённых целей, для которых используются относительные ссылки - это перемещение веток. Можно напрямую прикрепить ветку к коммиту при помощи опции -f. Например, команда:

git branch -f main HEAD~3

Переместит (принудительно) ветку main на три родителя назад от HEAD

**Добавление новых файлов в GitHub**

git remote add origin https://github.com/EvgeniyArckhipov/Perviy\_probniy.git

ИЛИ

git remote set-url origin https://github.com/EvgeniyArckhipov/Perviy.git