**Что такое ветка**

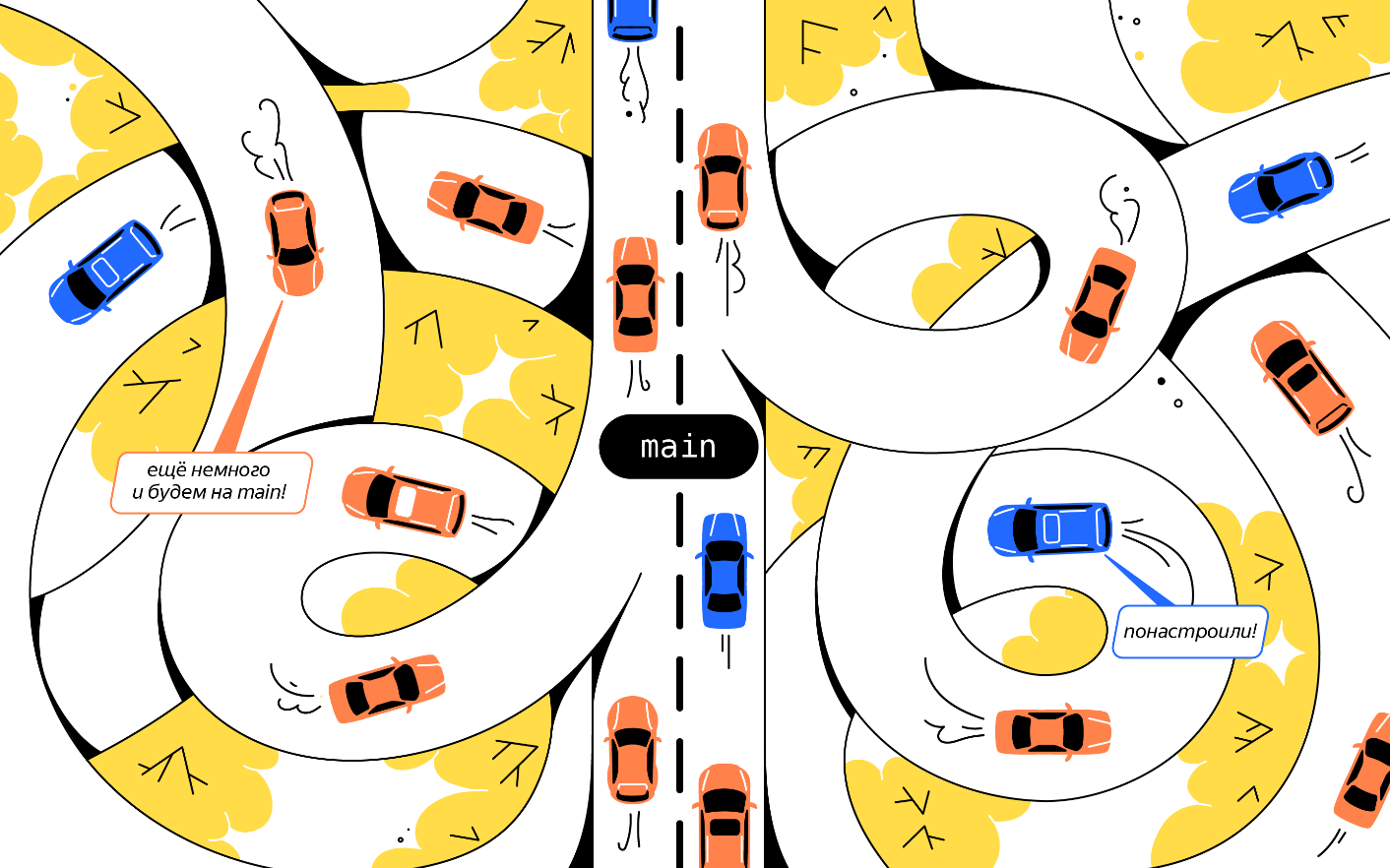
Представьте: ваш рабочий проект запущен и у него уже появились пользователи. Вам пришла идея, как его ускорить. Но нужно провести эксперимент — изменить код и посмотреть, будет ли программа работать быстрее. В процессе важно скооперироваться с коллегами и ничего не сломать — в этом помогут **ветки**.

**Зачем нужны ветки**

**Ветка** (англ. *branch*) — это изолированный поток разработки проекта. В таком потоке можно проверять разные идеи, тестировать новую функциональность и так далее.

Ветки позволяют экспериментировать с проектом в Git, но при этом сохранять репозиторий в стабильном состоянии. Каждый член команды может работать в своей ветке и не мешать другим: коммиты, которые он сделает, не будут видны из других веток. А когда работа будет доделана, ветки можно соединить.

Ветки полезны, даже если вы работаете в одиночку — например, над сайтом. Прежде чем писать новую функциональность, для неё следует создать отдельную ветку. Также ветки позволяют одному человеку переключаться между несколькими задачами сразу.



Основная, стабильная версия проекта хранится в главной ветке main или master. Она появляется автоматически при создании репозитория. Часто все новые ветки в репозитории отходят от main, хотя это не является правилом.

⚠️ Далее в наших примерах мы будем обозначать основную ветку как main. Обращайте на это внимание и не забывайте менять команды, если ваша ветка называется master.

**Просмотреть ветки проекта — git branch**

Теперь покажем, как получить список веток проекта. Создайте тестовый проект learn\_branches. Добавьте в него файл README.md, проиндексируйте изменения и сделайте коммит: git commit -m "Выполнить первый коммит".

Вспомните, что несколько команд можно объединять в одной строке терминала символом логического «и» — &&.

Скопировать код

BASH

$ mkdir learn\_branches && cd learn\_branches && git init # создали новый репозиторий

$ touch README.md # создали файл

$ git add . # команда git add с флагом-точкой подготовит к сохранению текущую папку; вместо этого можно вызвать git add --all

$ git commit -m "Выполнить первый коммит"

Репозиторий создан, файлы добавлены и закоммичены. В репозитории появилась главная ветка. Это можно проверить командой для просмотра веток git branch.

Скопировать код

BASH

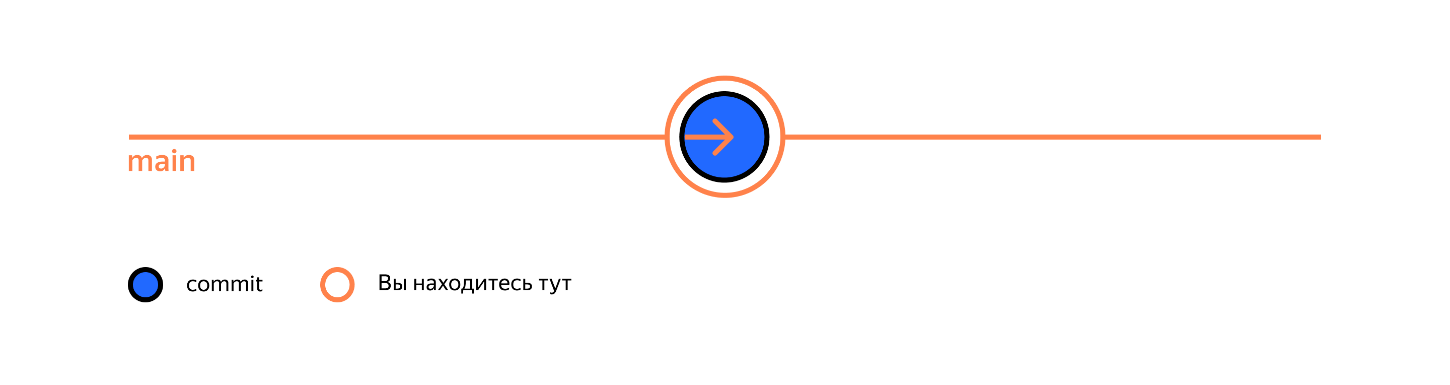
$ git branch

\* main # мы в основной ветке

# чтобы выйти из просмотра веток, может понадобиться Q!

При вызове git branch выводятся ветки, которые есть в проекте. Звёздочкой (\*) отмечено, в какой ветке вы находитесь в текущий момент.

Сейчас проект learn\_branches выглядит так.



**Дополняем ветку**

Добавьте в README.md следующий текст.

Скопировать код

MARKDOWN

# Описание

Это проект по изучению работы с ветками

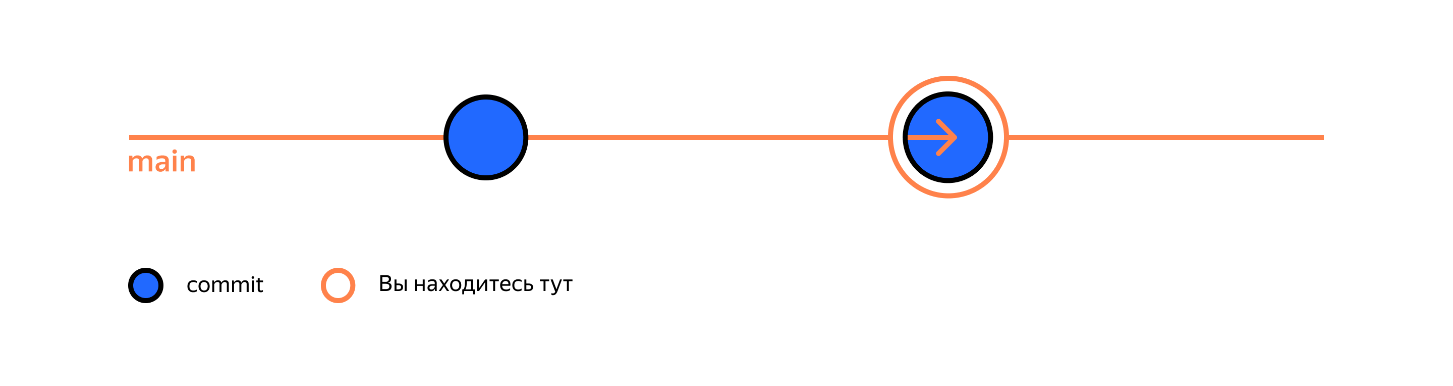
Сделайте ещё один коммит.

Скопировать код

BASH

$ git add . && git commit -m "Обновить README"

Как и предыдущий, этот коммит сохранится внутри главной ветки. Схематически это выглядит так.



Теперь у вас есть два коммита — последовательность изменений. А это уже целая ветка!

Начало формы

Выберите верные утверждения о ветках в Git.

Правильный ответ

Ветки позволяют нескольким людям вносить изменения в проект независимо друг от друга.

Благодаря веткам вы даже можете менять один и тот же файл одновременно!



Просмотреть все ветки проекта можно командой git branching.

Команда, о которой идёт речь, называется git branch.

Правильный ответ

Ветки — независимые потоки разработки.

Ветки могут развиваться совершенно независимо друг от друга.



При работе с Git-репозиторием нужно обязательно вносить каждое изменение в новую ветку.

Если вы работаете в одиночку, в этом нет необходимости. При работе в команде такой подход может быть полезен, но также не является обязательным.

Неправильный ответ

Все ветки в Git по умолчанию создаются от основной ветки.

Так можно делать, но это не является обязательным правилом.

Правильный ответ

Команда git branch позволяет получить список существующих веток проекта.

А также узнать, в какой из них вы находитесь прямо сейчас: такая ветка будет отмечена звёздочкой.

Конец формы

Замечательно! Ваше знакомство с ветками состоялось, и постепенно вы узнаете о них ещё больше. Коротко подытожим:

* Ветка — это последовательность независимых изменений.
* Благодаря веткам несколько человек могут работать над одним репозиторием и не мешать друг другу. А ещё ветки помогают декомпозировать большую и страшную задачу на маленькие и понятные.
* Основная версия проекта хранится в главной ветке main (или master).
* С помощью команды git branch можно посмотреть, какие в проекте есть ветки и в какой из них вы сейчас находитесь.

# Создаём ветку

Вы узнали, что такое ветки и зачем они нужны. В ветках можно экспериментировать с проектом, не боясь что-либо поломать или помешать другим членам команды. А можно разделить крупную задачу на несколько небольших и работать независимо.

Но чтобы начать работу в отдельной ветке, её нужно создать. Самое время научиться это делать!

### Создать ветку — git branch <название\_ветки>

Инициализируйте новый тестовый проект git-branches и создайте в нём файл README.md. Откройте файл и добавьте туда следующий текст.

Скопировать код

MARKDOWN

# Ветки в Git

Добавьте изменения в staging area и сделайте коммит с сообщением: git add . && git commit -m "Добавить файл README".

Для создания веток в Git есть команда git branch с параметром в виде названия ветки: git branch <название\_ветки>. Например, создадим ветку с названием feature/add-branch-info.

💡 **В названии ветки есть слеш — что это значит?**

Название ветки в Git может состоять из букв, цифр, а также включать любой из четырёх символов: ., -, \_, /. Эти символы не несут особого смысла. Например, ветка feature/add-branch-info могла бы называться feature\_add-branch-info или feature-add-branch. Обратите внимание, что ветки не образуют иерархии, как директории, разделённые символом /.

Чтобы создать ветку feature/add-branch-info, выполните команду git branch feature/add-branch-info. Затем выполните git branch, чтобы посмотреть список всех веток проекта.

Скопировать код

BASH

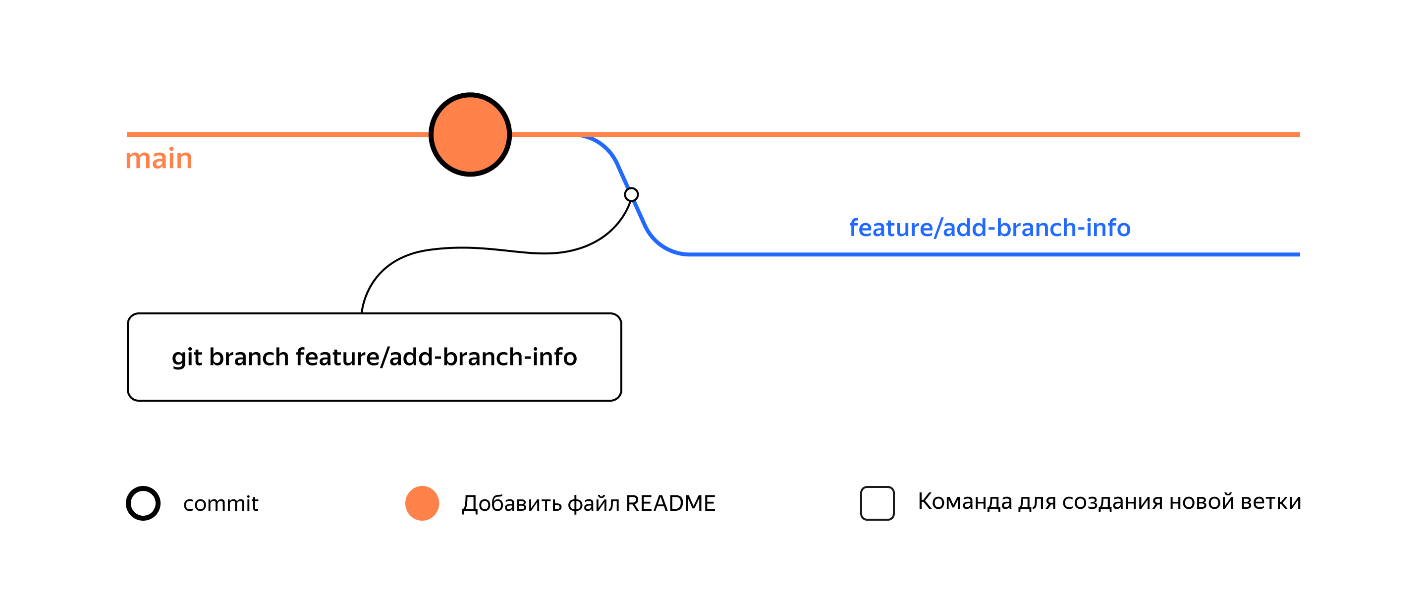
$ git branch feature/add-branch-info # создали ветку feature/add-branch-info

$ git branch # посмотрели ветки

feature/add-branch-info # появилась новая

\* main # \* значит, что мы находимся в основной ветке

Готово! Сейчас в вашем репозитории две ветки — основная и feature/add-branch-info.



### ****Как назвать новую ветку****

Есть разные подходы к наименованию веток. Каждая команда разработки выбирает свой. Но независимо от подхода ветки нужно называть так, чтобы другим участникам было понятно, что в них происходит.

Мы будем использовать указатели feature (англ. «особенность», «деталь») для веток, где прорабатывается новая функциональность, и bugfix (от англ. bug — «жук», «ошибка» и fix — «исправить») для веток, где ведётся работа по исправлению ошибок.

После ключевого слова идёт слеш и описание проблемы или задачи (например, /add-branch-info). Это описание не должно содержать пробелов — следует использовать нижнее подчёркивание или дефис. В наших примерах мы будем использовать дефис.

Хорошо! Теперь вы умеете создавать ветки с помощью команды git branch <название\_ветки>. Ветки в Git — удобный инструмент. В каждой может быть сколько угодно коммитов, а когда работа будет сделана, останется только объединить их.

# Шагаем с ветки на ветку

Допустим, вы создали ветку для экспериментов с вашим рабочим проектом. Чтобы поработать в ней, сначала вам нужно на неё переключиться.

Например, в прошлом уроке вы создали ветку feature/add-branch-info, но пока по-прежнему находитесь в main (или master). Пора это исправить и научиться перемещаться между ветками!

### Переключиться на другую ветку — git checkout <название\_ветки>

Чтобы начать работу в feature/add-branch-info, перейдите в неё с помощью команды git checkout с флагом — названием ветки: git checkout feature/add-branch-info.

Скопировать код

BASH

$ git checkout feature/add-branch-info # перешли в новую ветку

Switched to branch 'feature/add-branch-info'

$ git branch # проверили

\* feature/add-branch-info # теперь находимся тут

main

Строчка Switched to branch... (англ. «переключено на ветку…») сообщает, на какую ветку вы только что переключились.

Откройте файл README.md и добавьте в него информацию о командах git branch и git checkout. Получится следующее.

Скопировать код

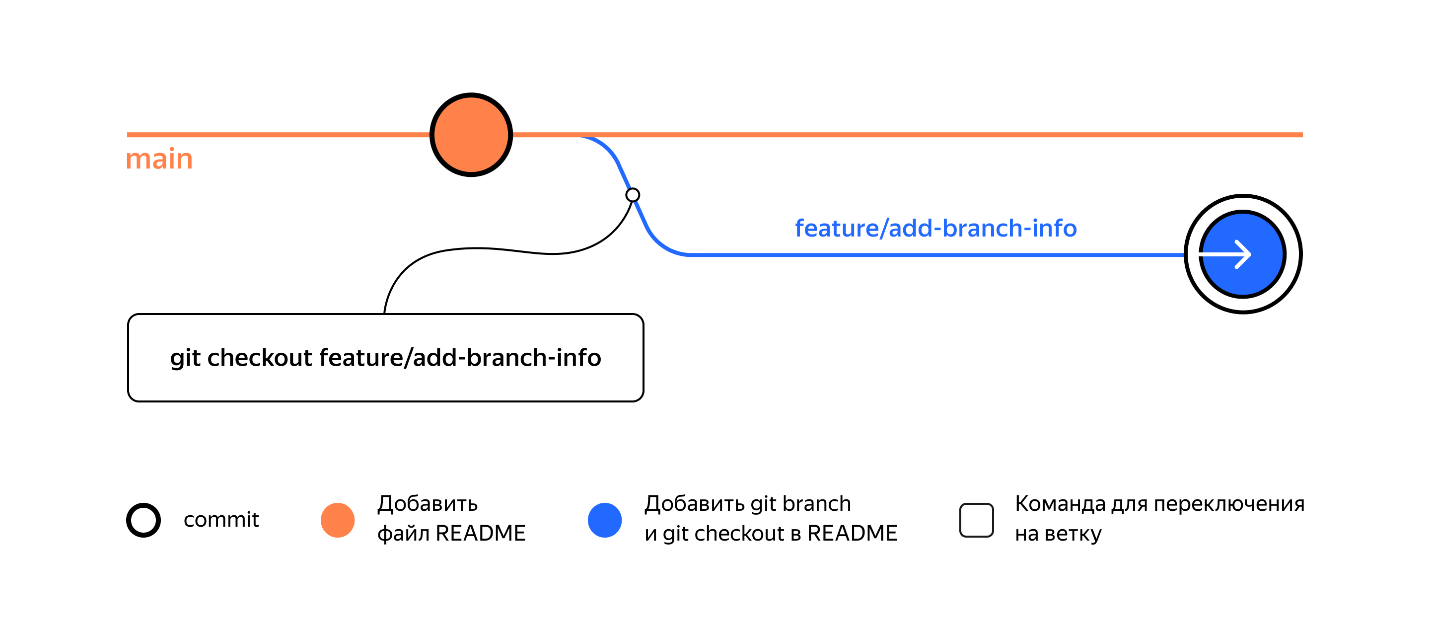
MARKDOWN

# Ветки в Git

Чтобы создать ветку, необходимо выполнить команду `git branch %BRANCH\_NAME%`.

Для перехода в ветку есть команда `git checkout %BRANCH\_NAME%`.

Добавьте изменения в staging area и сделайте коммит с сообщением: git add . && git commit -m "Добавить git branch и git checkout в README". Сейчас история проекта выглядит так.



Теперь вернитесь в главную ветку командой git checkout main (или git checkout master в зависимости от настроек вашей системы) и откройте файл README.md. В нём отобразится только первоначальный текст, который вы добавили в прошлом уроке. Все изменения сохранились в ветку feature/add-branch-info, а состояние главной ветки не изменилось.

### Создать ветку и сразу переключиться на неё — git checkout -b <название\_ветки>

Можно создать ветку и сразу начать в ней работать. За это отвечает команда git checkout с флагом -b (от англ. ***b***ranch) и названием ветки. Эта команда делает то же самое, что и последовательность команд git branch %название-ветки% && git checkout %название-ветки%.

Убедитесь, что вы в основной ветке. Затем создайте ещё одну ветку для исправления ошибок bugfix/fix-branch и сразу переключитесь на неё.

Скопировать код

BASH

$ git checkout main

$ git checkout -b bugfix/fix-branch # создали ветку и сразу на неё переключились

Switched to a new branch 'bugfix/fix-branch'

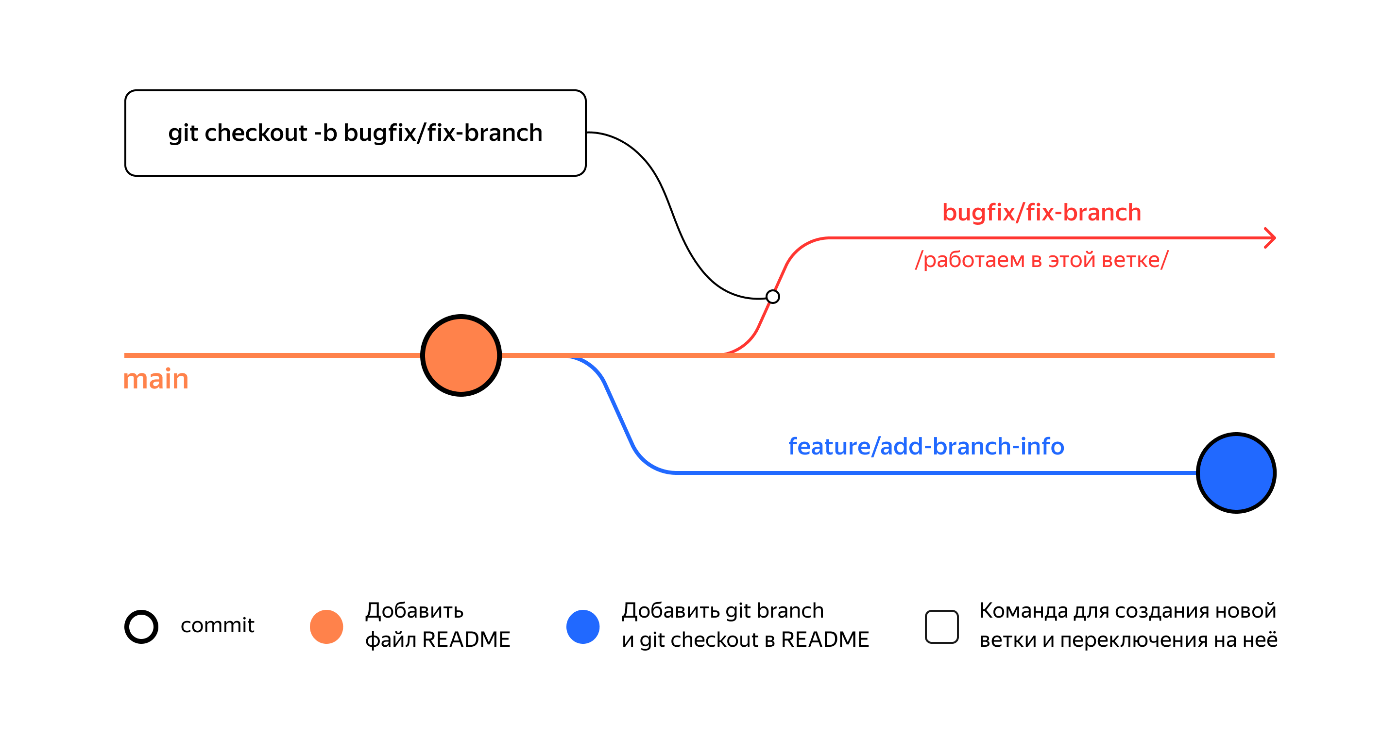
$ git branch

\* bugfix/fix-branch # сразу в нужной ветке

feature/add-branch-info

main

Строчка Switched to a new branch... (англ. «переключено на новую ветку…») сообщает о том, что вы переключились на только что созданную ветку. Теперь история проекта выглядит так.



### На какой коммит указывает bugfix/fix-branch

Ветка в Git — это указатель на коммит. Когда вы делаете новый коммит в ветке, этот указатель передвигается вперёд.

Пока вы не вносили новые коммиты в ветку bugfix/fix-branch, поэтому она указывает на тот же коммит, что и основная ветка. Убедитесь в этом с помощью команды git log --oneline.

Скопировать код

BASH

$ git checkout bugfix/fix-branch

$ git log --oneline

a7eb909 (HEAD -> bugfix/fix-branch, main) Добавить файл README

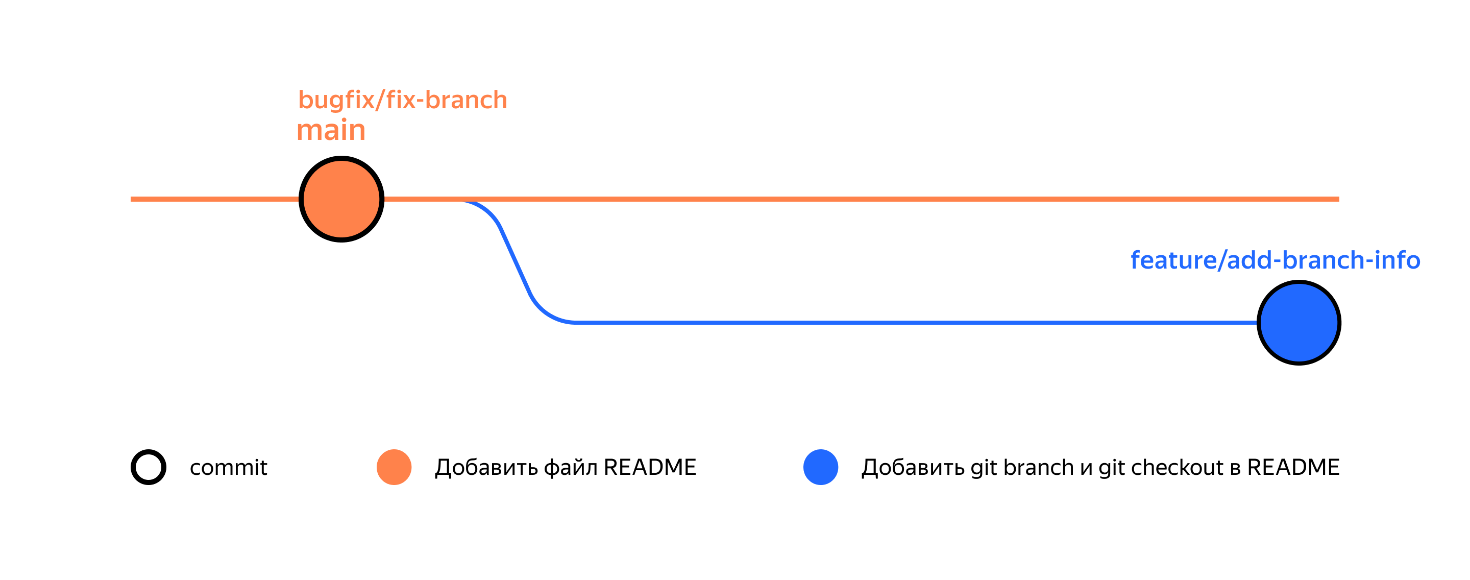
$ git checkout feature/add-branch-info

$ git log --oneline

abd591c (HEAD -> feature/add-branch-info) Добавить git branch и git checkout в README

a7eb909 (main, bugfix/fix-branch) Добавить файл README

Так что можно показать текущее состояние проекта git-branches так: поместить названия веток над коммитами, на которые они указывают.



Такой вариант схем мы будем использовать дальше в модуле.

Начало формы

Выберите верные утверждения.



Команда git checkout <название\_ветки> позволяет создать ветку и сразу перейти в неё.

С помощью этой команды получится только перейти в другую ветку, но не создать её.

Правильный ответ

С помощью git branch <название\_ветки> можно создать ветку.

Эта команда инициализирует новую ветку.



С помощью git checkout -- <название\_ветки> можно создать ветку и перейти в неё.

Ветка в команде checkout пишется без --, но в этом случае команда только переместит вас в другую ветку, но не создаст её.

Правильный ответ

Команда git checkout -b <название\_ветки> создаёт ветку и сразу переключается на неё.

Эта команда объединяет git branch <название\_ветки> и git checkout <название\_ветки>.



Для создания ветки и перехода в неё используют git checkout —-branch <название\_ветки>.

Эта команда не сработает. Для создания и перехода нужно использовать флаг -b.

Неправильный ответ

Две ветки не могут указывать на один коммит: у каждой он должен быть свой.

Могут: например, если в только что созданной ветке ещё не было новых коммитов.

Конец формы

## **Просматриваем все ветки:**git branch -a

Команда git branch показывает только локальные ветки – те, которые существуют на вашем локальном компьютере.

Например, вы склонировали себе репозиторий, в котором котором на сервере GitHub есть несколько веток.

* В интерфейсе GitHub вы увидели, что есть ветка feature/add-branch-info и переключились на неё с помощью команды git checkout feature/add-branch-info.
* Позже вы создали ветку bugfix/fix-branch с помощью команды git checkout -b bugfix/fix-branch.
* Затем вернулись на ветку main: git checkout main. При вызове git branch вы увидите следующий вывод:

Скопировать код

BASH

$ git branch

\* main

feature/add-branch-info

bugfix/fix-branch

Когда вы работаете с несколькими ветками в проекте, полезно уметь просматривать все доступные ветки – хранящиеся как на вашем компьютере, так и на сервере GitHub. Для этого используется команда git branch -a. Флаг -a в команде git branch -a означает “all” (англ. «все»). Использование этого флага позволяет команде отображать все ветки в репозитории:

* Локальные ветки будут указываться как обычно при вызове git branch, например, feature/add-branch-info.
* Удалённые – с префиксом remotes/origin: remotes/origin/feature/add-branch-info.
* Отдельно будет указана основная ветка: строка с ней будет выглядеть как remotes/origin/HEAD -> origin/main. Прочитать эту строку можно как «Главной веткой является main».

Пример:

Скопировать код

BASH

$ git branch -a

\* main

feature/add-branch-info

bugfix/fix-branch

remotes/origin/HEAD -> origin/main

remotes/origin/feature/add-branch-info

remotes/origin/main

В этом выводе вы видите список всех веток, где:

* main, feature/add-branch-info, и bugfix/fix-branch — это локальные ветки.
* remotes/origin/HEAD -> origin/main, remotes/origin/feature/add-branch-info, и remotes/origin/main — это удалённые ветки.

Начало формы

По выводу команды git branch -a определите, какие ветки хранятся **только** локально.

Скопировать код

BASH

$ git branch -a

\* develop

feature/login

hotfix/security-fix

main

remotes/origin/HEAD -> origin/main

remotes/origin/develop

remotes/origin/feature/login

remotes/origin/main



develop



feature/login



hotfix/security-fix



origin/main



remotes/origin/HEAD

Путешествие по веткам было головокружительным! Вот самое важное, о чём мы рассказали:

* Команда git checkout <название\_ветки> позволяет переключаться на другую ветку.
* Разные ветки в одном проекте существуют независимо. Изменения в одной не влияют на изменения в другой.
* В Git можно создать ветку и сразу же перейти в неё командой git checkout -b <название\_ветки>.
* Ветка указывает на коммит, который сделан в ней последним. При этом две ветки могут ссылаться на один и тот же коммит — например, если вы только что создали ветку, но ещё не успели внести в неё коммит.
* Посмотреть и локальные, и удалённые ветки можно с помощью команды git branch -a.

# Сравниваем ветки

В одном из уроков предыдущего модуля мы рассказывали о команде git diff. Она выводит результат сравнения коммитов, modified- и staged-файлов.

В этом уроке расскажем ещё о двух способах применения git diff — для сравнения веток, а также с суффиксом навигации ~. Этот суффикс поможет просмотреть изменения в нескольких последних коммитах.

### Сравнить ветки — git diff <название\_ветки1> <название\_ветки2>

Вернитесь к проекту git-branches. Перейдите в основную ветку через git checkout. Откройте файл README.md и добавьте пункт про команду git branch. Файл получится таким.

Скопировать код

MARKDOWN

# Ветки в Git

Чтобы посмотреть все активные ветки в проекте, нужно вызвать команду `git branch` без аргументов.

Сохраните файл, добавьте изменения в staging area, а затем выполните коммит: git add . && git commit -m "Добавить git branch в README".

Создайте ветку feature/diff и перейдите в неё. Снова откройте файл README.md и добавьте туда новую строку. Должен получиться такой текст.

Скопировать код

MARKDOWN

# Ветки в Git

Чтобы посмотреть все активные ветки в проекте, нужно вызвать команду `git branch` без аргументов.

Для сравнения веток есть команда `git diff`.

Подготовьте файл к сохранению и сделайте коммит: git add . && git commit -m "Добавить git diff в README".

Теперь используйте команду git diff main feature/diff (или git diff master feature/diff), чтобы вывести разницу между двумя ветками. Вы увидите точно такой же вывод, как если бы сравнивали два коммита между собой.

Скопировать код

BASH

$ git diff main feature/diff # сравнили ветки main и feature/diff

diff --git a/README.md b/README.md

index 86b1ff4..fff4920 100644

--- a/README.md

+++ b/README.md

@@ -1,3 +1,5 @@

# Ветки в Git

Чтобы посмотреть все активные ветки в проекте, нужно вызвать команду `git branch` без аргументов.

+

+Для сравнения веток есть команда `git diff`.

При сравнении вы также можете использовать название ветки и хеш коммита. Для этого сначала выполните команду git log --oneline, чтобы вывести список коммитов.

Скопировать код

BASH

$ git log --oneline

2ea56ab (HEAD -> feature/diff) Добавить git diff в README

de8b09b (main) Добавить git branch в README

7ad18bd Добавить файл README

Теперь выполните команду git diff с названием основной ветки и хешем коммита в ветке feature/diff. У нас получилась следующая комбинация: git diff main 2ea56ab. У вас параметры могут быть другими.

Скопировать код

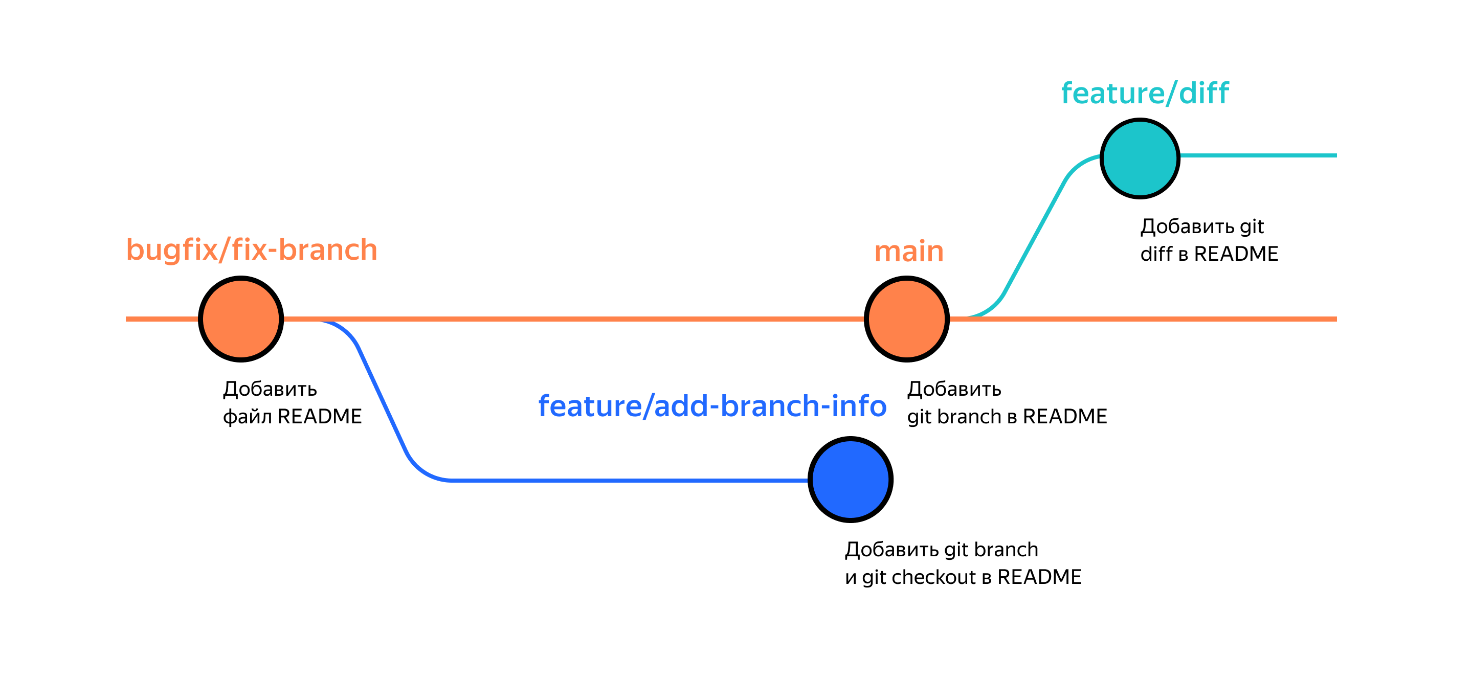
BASH

$ git diff main 2ea56ab

# вывод будет такой же, как при использовании git diff main feature/diff

Вывод будет такой же, как после выполнения команды git diff main feature/diff. Когда вы вызываете git diff <название\_ветки1> <название\_ветки2>, Git находит два коммита, на которые указывает каждая из веток, и сравнивает их. Также с веткой можно сравнивать указатель HEAD.

А вот как схематически выглядит ваш проект git-branches после всех изменений.



Начало формы

Выберите верные утверждения о команде git diff.

Тоже правильный ответ

Команда git diff покажет изменения в «рабочей зоне», то есть в modified-файлах.

Так git diff работает без параметров.

Тоже правильный ответ

git diff HEAD main покажет разницу между коммитом, который сделан последним, и веткой main.

Вместо HEAD может быть и другая ветка или хеш коммита.



git diff HEAD feature/diff покажет разницу между modified-файлами и веткой feature/diff.

Указатель HEAD показывает на коммит, который сделан последним.



git diff main feature/diff покажет изменения, которые нужны, чтобы получить состояние ветки main из ветки feature/diff.

Наоборот: первая ветка в команде main, поэтому эта команда покажет как получить feature/diff из main.

Конец формы

### Суффикс навигации ~

Сравнивать хеши комитов может быть неудобно, ведь в одной ветке их может быть много. Представьте: сначала вы выводите историю через git log, затем ищете в длинном списке хеши тех коммитов, которые хотите сравнить, и только потом выполняете git diff.

Для облегчения этой задачи в Git есть суффикс навигации ~N, где N — это число. Он отсчитывает от заданного коммита N коммитов назад во времени. Нумерация начинается с нуля: commit~0 — это сам коммит, commit~1 — предыдущий, commit~2 — предшествующий предыдущему и так далее.

Например, HEAD~1 — это следующий за текущим коммит. А main~5 — это пятый коммит в ветке main, если считать с последнего выполненного коммита.

На практике чаще нужен либо текущий коммит (HEAD), либо следующий за ним (HEAD~1). Для ~1 есть специальное сокращение ~ (без числа). То есть вместо HEAD~1 обычно пишут просто HEAD~.

💡 Также можно использовать ~0, но большого смысла в этом нет: main~0 — это то же самое, что просто main, а HEAD~0 — это просто HEAD.

Потренируйтесь: чтобы вывести разницу между предыдущим и текущим коммитами, выполните следующую команду.

Скопировать код

BASH

$ git diff HEAD~ HEAD

Или то же самое, но через ~N.

Скопировать код

BASH

$ git diff feature/diff~1 feature/diff

Или можно использовать хеши.

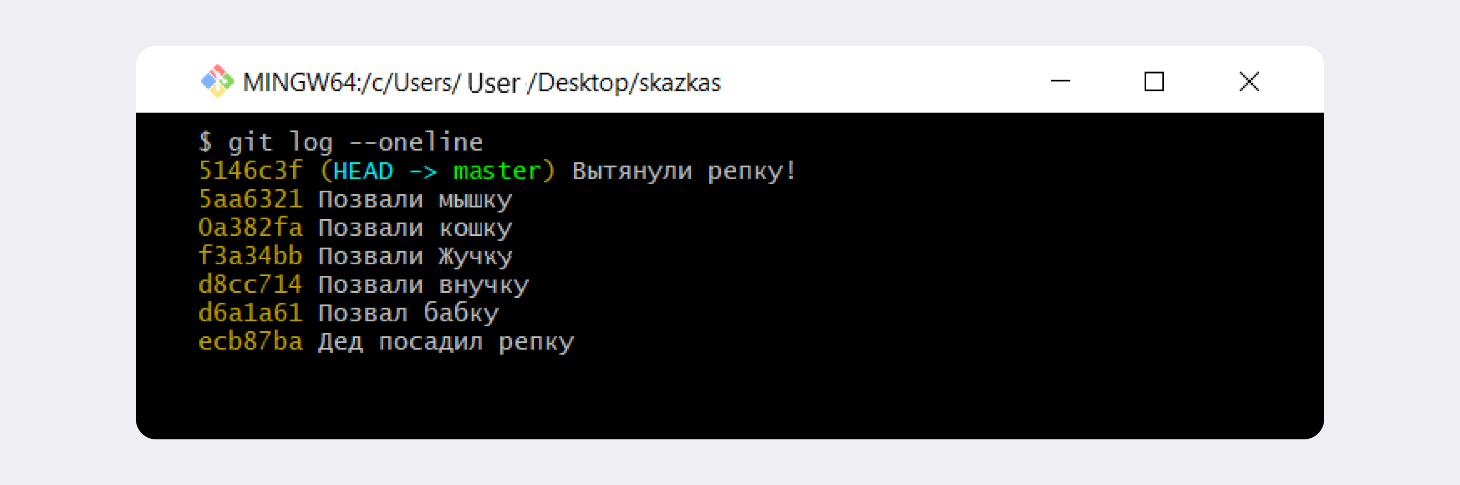
Скопировать код

BASH

$ git diff 2ea56ab~ 2ea56ab

Начало формы

Изучите вывод команды git log --oneline для репозитория skazkas и выполните задание.



Выберите верные утверждения.

Правильный ответ

Команда git diff master master~2 сравнит HEAD и коммит 0a382fa.

master~2 — указатель на коммит, который был два коммита назад, то есть 0a382fa.

Неправильный ответ

Команда git diff main main~1 сравнит коммиты HEAD и 5aa6321.

Посмотрите внимательнее на название веток. Основная ветка проекта skazkas — master.



Команда git diff master master~2 сравнит коммиты HEAD и d8cc714.

HEAD указывает на последний коммит, который находится наверху списка.

Тоже правильный ответ

Команда git diff HEAD~4 HEAD~2 сравнит коммиты про внучку и кошку.

А между ними будет коммит про Жучку!

Тоже правильный ответ

Команда git diff master~3 master~1 покажет, какие изменения нужны, чтобы к Жучке присоединилась мышка.

Команда сравнит коммиты f3a34bb и 5aa6321 соответственно.

Конец формы

Потрясающе! В этом уроке вы узнали о ещё двух полезных функциях команды git diff. А именно:

* git diff может сравнивать ветки по их названиям. Например, команда git diff main feature/my-feature выведет разницу между основной веткой и веткой feature/my-feature.
* Git поддерживает суффикс навигации ~. С его помощью можно сослаться на предыдущие коммиты. Например, если вы находитесь в ветке main и хотите вывести разницу между тем коммитом, который был три коммита назад, и текущим, нужно выполнить git diff main~3 main.

# Практическая работа №2. Помогаем другу разобраться в ветках

В этой практической работе вы погрузитесь в мир веток и потренируетесь не путаться в них. Вперёд!

Ваш друг Боря впечатлён успехами, которых вы с Федей добились в работе над пет-проектом. Боря тоже заинтересовался Git. Но в отличие от вас он пока делает первые шаги в обучении и ещё многого не умеет — например, работать с ветками.

Платформа GitHub стала для Бори настоящим открытием. Она кажется другу чем-то вроде цифровой сокровищницы. Именно там он однажды нашёл проект, в котором ему очень захотелось разобраться.

Проект небольшой: всего несколько веток. Для Бори это вызов, но для вас нет ничего невозможного! Помогите другу понять, что происходит в репозитории.

### Что нужно сделать

1. Откройте [проект, который понравился Боре,](https://github.com/yandex-praktikum/git-branch-case) и склонируйте его.
2. Изучите репозиторий проекта как через интерфейс GitHub, так и локально: обратите внимание на структуру веток и коммиты.

Конец формы