GIT система контроля версий

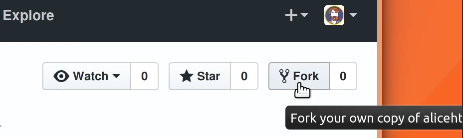
1. Р- GIT
   1. Алгоритм командного взаимодействия

|  |
| --- |
| Общие теги: Разобрать после..., Скилл |

* + 1. Fork чужого проекта

|  |
| --- |
| Общие теги: Скилл |

Открываем чужой проект в github



* + 1. Вносим удаленный репозитория в свой проект

|  |
| --- |
| Общие теги: Скилл |

$ git remote add НАЗВАНИЕ АДРЕС РЕПОЗИТОРИЯ

* + 1. Делаем push изменений в github

|  |
| --- |
| Общие теги: Скилл |

$ git push -u (ветка удаленного репозитория)origin (локальная ветка)share-icons

* + 1. Забираем изменения ветки мастер этого репозитория

|  |
| --- |
| Общие теги: Скилл |

$ git checkout master

$ git pull НАЗВАНИЕ-РЕПОЗИТОРИЯ НАЗВАНИЕ-ВЕТКИ

* + 1. Заходим в проект и проверяем remote

|  |
| --- |
| Общие теги: Скилл |

$git remote -v

* + 1. Клонируем fork по ssh

|  |
| --- |
| Общие теги: Скилл |

$ git clone ADDRESS

* + 1. Открытые PULL-REQUEST всегда показывают актуальную разницу между изменениями

|  |
| --- |
| Общие теги: Скилл |

Если нужно просто что-то доделать, достаточно что-то изменить, сделать Push и другой участник в pull request сразу увидит изменения

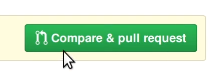
* + 1. Отправляем изменения в свой гитхаб

|  |
| --- |
| Общие теги: Скилл |

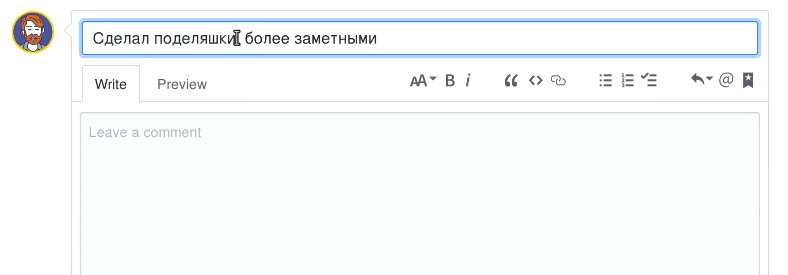
$ git push origin master

* + 1. Предлагаем свои изменения

|  |
| --- |
| Общие теги: Скилл |



Пишем свои заметки по правкам



graphic

* + 1. Чтобы продолжить работу, нужно получить изменения

|  |
| --- |
| Общие теги: Скилл |

Заходим в репозиторий, с которого брали Fork и копируем его SSH адрес

* 1. Однопользовательский режим

|  |
| --- |
| Общие теги: Разобрать после..., Скилл |

* + 1. Клонирование с удаленного репозитория

|  |
| --- |
| Общие теги: Скилл |

$ git clone 'address'

* + 1. Настройка SSH Ключей

|  |
| --- |
| Общие теги: Скилл |

1. Создаем в домашней папке папку: *.ssh* и заходим в нее
2. Генерируем ключи: $ ssh-keygen -t rsa -b 4096 -C "email"
3. Указываем название ключу.
4. Указываем пароль для ключа.
5. Публичный ключ загружаем на github: Accaunt - settings - SSH and GPG keys - New SSH key: название ключа, key - через комманду $ cat открываем публичный ключ и копируем содержимое
6. Тестируем соединение: $ ssh -T -i ~/.ssh/github-2020-02-31 git@github.com
7. В домашней директории открываем файл или создаем: ~/.ssh/config:  
   Host github.com  
    IdentityFile ~/.ssh/github-2020-02-31
8. Пробуем соединиться с GitHub без ключа: $ssh -T git@hub.com
   * 1. Обзор связи локальных веток с удаленными

|  |
| --- |
| Общие теги: Скилл |

$git branch -vv

* + 1. Отправка изменений на УР

|  |
| --- |
| Общие теги: Скилл |

$ git push -u origin 'ВЕТКА'// отправляет наши наборы изменений на удалённый репозиторий.

$ git push origin 'ИМЯ ЛОКАЛЬНОЙ ВЕТКИ':'ИМЯ УДАЛЕННОЙ ВЕТКИ' // Отправка любой локальной ветки в любую удаленную.

$ git push origin :'ИМЯ УДАЛЕННОЙ ВЕТКИ' // Удалить удаленную ветку из удаленного репозитория

* + 1. Подключение

|  |
| --- |
| Общие теги: Скилл |

**Работа с удаленными репозиториями:**

git remote add NAME PATH

git remote –v // Проверка на успешность подключения удаленного репозитория

git remote show NAME

* + 1. Получение изменений с УР

|  |
| --- |
| Общие теги: Скилл |

$ git pull origin master

- не просто «достаёт» изменения, но и пытается применить изменения с удаленного репозитория к нашему (влить удаленную ветку в локальную).

$ git fetch origin // Загружает коммиты и ветки из удаленного репозитория к себе в локальный/

---------------

Если нужно перейти на ветку удаленного репозитория

$ git checkout -b (локальная ветка)comments (удаленная ветка)origin/comments

* + 1. Получение информации об изменениях в репозитории

|  |
| --- |
| Общие теги: Скилл |

* + 1. Связь с веткой УР

|  |
| --- |
| Общие теги: Скилл |

**Checkout**

Вообще эта команда применяется для переключения между ветками. Но мы ее используем для того, чтобы связать нашу ветку master и ветку master удаленного репозитория:

git checkout --track REMOTENAME/master

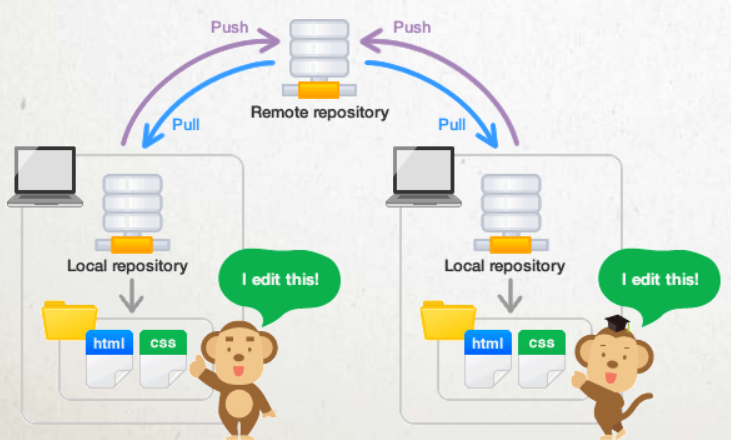
-----------

$ git branch --set-upstream-to=origin/comments // Связывает удаленную ветку с текущей

* + 1. Слияние репозиториев

|  |
| --- |
| Общие теги: Скилл |

git clone CLONE\_URL PROJECT\_FOLDER



* 1.  Git Cheat Sheet 2

См. документ(-ы): <https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>

|  |
| --- |
| Общие теги: Скилл |

* 1. Справочник комманд

|  |
| --- |
| Общие теги: Информация |

* + 1. Версия для печати

См. прикрепленные файлы: [GIT Command.pdf](file:///D:\ARTCLOUD\Проекты\-%20Pr.Freedom%20-\Квартал\material\GIT%20система%20контроля%20версий.docx%20-%20Прикрепленные%20файлы\GIT%20Command.pdf)

* + 1.  Команды Git

См. документ(-ы): [Команды Git](http://www.zlatov.net/publication/komandy-git.html#h23)

* 1.  Внешние ресурсы

|  |
| --- |
| Общие теги: Информация |

* + 1. Git Cheat Sheet

См. документ(-ы): [Git Cheat Sheet](https://www.alexkras.com/getting-started-with-git/)

* + 1. 19 советов по повседневной работе с Git / Блог компании Mail.Ru Group / Хабр

См. документ(-ы): [19 советов по повседневной работе с Git / Блог компании Mail.Ru Group / Хабр](https://habr.com/company/mailru/blog/267595/)

|  |
| --- |
| Общие теги: Разобрать после... |

* + 1. Visualizing Git

См. документ(-ы): [Visualizing Git](http://git-school.github.io/visualizing-git/#upstream-changes)

* + 1. Git снизу вверх / Блог компании Intel / Хабр

См. документ(-ы): [Git снизу вверх / Блог компании Intel / Хабр](https://habr.com/company/intel/blog/344962/)

* + 1.  Git

См. документ(-ы): [Git](https://git-scm.com)

|  |
| --- |
| Уровень освоения: Минимальный |

* + 1.  Git for Windows

См. документ(-ы): [Git for Windows](https://git-for-windows.github.io/)

|  |
| --- |
| Уровень освоения: Минимальный |

* 1. Области видимости

|  |
| --- |
| Общие теги: Информация |

1. Файл **/etc/gitconfig** содержит значения, общие для всех пользователей системы и для всех их репозиториев. Если при запуске git config указать параметр --system, то параметры будут читаться и сохраняться именно в этот файл.

2. Файл **~/.gitconfig** или **~/.config/git/config** хранит настройки конкретного пользователя. Этот файл используется при указании параметра **--global**.

3. Файл config в каталоге **Git’а (т.е. .git/config)** в том репозитории, который вы используете в данный момент, хранит настройки конкретного репозитория.

* 1.  Настройка собственного текстового редактора

См. документ(-ы): [Связывание текстового редактора с Git | Юрий Матюхин](https://ymatuhin.ru/tools/git-default-editor/)

|  |
| --- |
| Общие теги: Метод |
| Уровень освоения: Минимальный |

* 1. Настройки

|  |
| --- |
| Общие теги: Метод |
| Уровень освоения: Минимальный |

Всего две настройки – ваше имя и почта:

$ git config --global user.name "Иван Иванов"

$ git config --global user.email ivanov@example.com

$ git config --list

Проверка настроек

* + 1. Базовые

git config --global color.ui true

git config --global push.default current

git config --global core.editor vim

git config --global user.name "John Doe"

git config --global user.email foo@citrix.com

git config --global diff.tool meld

* + 1. Проверка успешности установки

В консоли ввести

$ git --version

* 1. Справка F1 по командам

|  |
| --- |
| Общие теги: Метод |
| Уровень освоения: Минимальный |

$git help 'command'

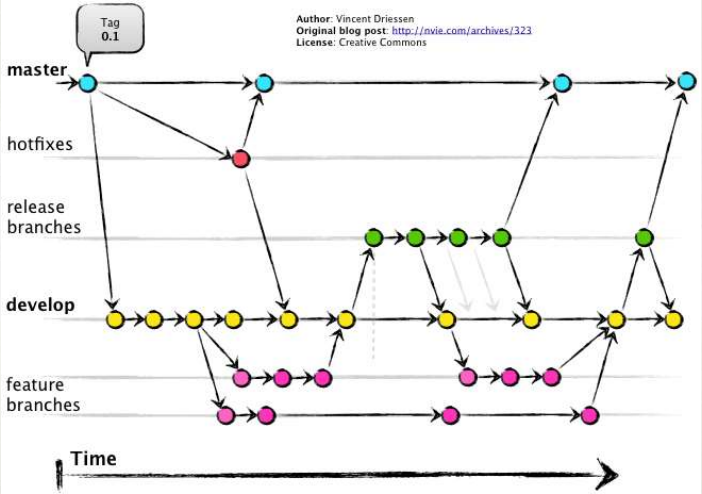
* 1. Методология использование GIT

|  |
| --- |
| Общие теги: Применение |

* + 1. Использовать тематические ветки
    2. Как вообще применять ветки
  1.  GIT FLOW. Методология использования GIT

См. документ(-ы): [Шпаргалка по git-flow](https://danielkummer.github.io/git-flow-cheatsheet/index.ru_RU.html)

|  |
| --- |
| Общие теги: Применение |



* 1. Назначение

|  |
| --- |
| Общие теги: Применение |

Это такая программа (чаще всего – набор скриптов), которая умеет управлять изменениями. Она следит за файлами и папками, ведет историю изменений, умеет ими манипулировать (фиксировать, откатывать, сливать и так далее)

* 1. Основная задача

|  |
| --- |
| Общие теги: Применение |

**Это система контроля версий:**

Ее задача – вести полную историю изменений в некоей папке (репозиторий)

**Изменения:**

* Добавление и удаление файлов
* Модификация содержимого файлов

В историю также записывается кто и когда сделал изменение

Git – распределенная система контроля версий

* Нет «центрального» репозитория
* Любая папка под управлением Git является репозиторием
* Репозитории могут обмениваться изменениями

Git любит «ветки»! Ветка – это альтернативная история изменений

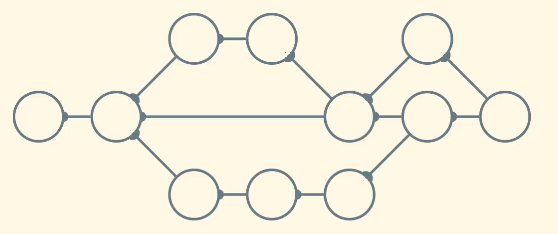
Ветки легко создавать, удалять и сливать

* 1. Пустые папки

|  |
| --- |
| Общие теги: Применение |

GIT не видит пустые папки

1. Р- Ветки



* 1. Работа с локальными ветками

|  |
| --- |
| Общие теги: Скилл |

# See the list of all local branches

git branch

# Switch to existing local branch

git checkout branchname

# Checkout current branch into a new branch, named new-branch-name

git checkout -b new-branch-name

# Merge branch-name into the current branch

git merge branchname

# Merge branch without fast forwarding. This is what pull requests do.

# It helps to preserve history of the changes as relavant to that branch

# It's an advanced feature, but try it out with GUI to see the difference

# between the regular merge and merge --no-ff

git merge --no-ff branchname

# Soft branch delete, will complain if the branch is not merged

git branch -d branchname

# Hard branch delete, will not complain about nothing. Like rm -rf in bash

git branch -D branchname

* 1. Указатель коммита

|  |
| --- |
| Общие теги: Информация |

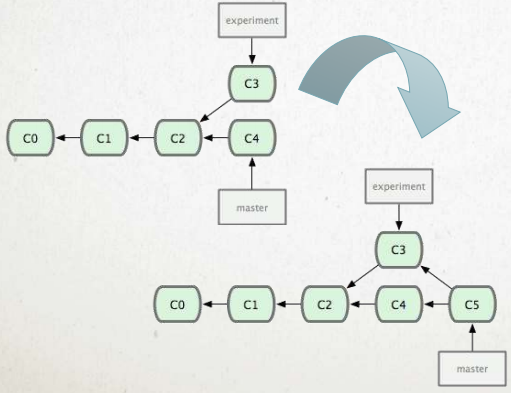
По умолчанию - Master. Указатель шагает по вновь создаваемым коммитам вместе с нами. Указатели можно создавать.

* 1. git REBASE

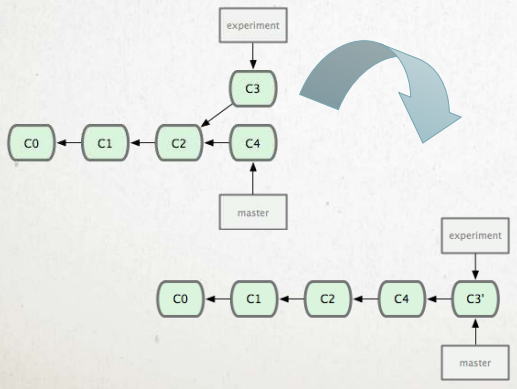
|  |
| --- |
| Общие теги: Метод |

Пожалуй, самый сложный в понимании инструмент Git. Его суть – перемещение изменений из одной ветки в другую

Это – обычный merge:



А это – rebase:



* + 1. Нужно помнить

Есть одно важное правило:

**Не перемещайте коммиты, которые вы уже отправили в публичный репозиторий.**

Если вы будете следовать этому указанию, всё будет хорошо. Если нет — люди возненавидят вас, вас будут презирать ваши друзья и семья.

Когда вы что-то перемещаете, вы отменяете существующие коммиты и создаёте новые, которые похожи на старые, но являются другими.

* 1. Создание

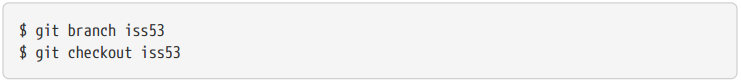
|  |
| --- |
| Общие теги: Метод |

* + 1. Создание

|  |
| --- |
| Уровень освоения: Минимальный |

$ git checkout -b 'NAME' // Создаем новую ветку от текущего местоположения и переключаемся на неё.

Это тоже самое, что: {



}

$ git branch testing

$ git branch -m 'НАЗВАНИЕ' // Переименовывает, удаляет и создает ветки, **но не переходит на них автоматом**

* 1. Редактирование и мониторинг

|  |
| --- |
| Общие теги: Метод |

* + 1. Переключение на новую ветку

|  |
| --- |
| Уровень освоения: Минимальный |

Git хранит специальный указатель, который называется **HEAD**. Это указатель на текущую ветку и коммит

git checkout 'название ветки' / 'HASH коммита'

Эта команда передвинет HEAD на ветку testing То есть переключит репозиторий на другую ветку.

git checkout

Устанавливает текущую ветку и переключается в соответствующее состояние кодовой базы. Ты также можешь перейти в удаленную ветку, в тэг или в конкретный коммит, но текущая ветка будет покинута и ты будешь получать предупреждения о наличии "оторванной HEAD". Это буквально означает, что HEAD (специальное имя, которое всегда указывает на то, с чем ты работаешь) не указывает на ветку, и если ты делаешь новые коммиты, у них не будет ничего, указывающее на них и они могут леко потеряться.

Просмотр всех веток репозитория

|  |
| --- |
| Уровень освоения: Минимальный |

Команда git branch делает несколько больше, чем просто создаёт и удаляет ветки. При запуске без параметров, вы получите простой список имеющихся у вас веток:

фцв

\* - означает ветку, на которой HEAD

* + 1. Закладка

|  |
| --- |
| Уровень освоения: Минимальный |

git stash

Эта команда спрячет все незафиксированные изменения в своеобразный буфер.

git stash apply

Достанет изменения из буфера.

Shelve and restore incomplete changes

$ git stash

Temporarily stores all modified tracked files

$ git stash pop

Restores the most recently stashed files

$ git stash list

Lists all stashed changesets

$ git stash drop

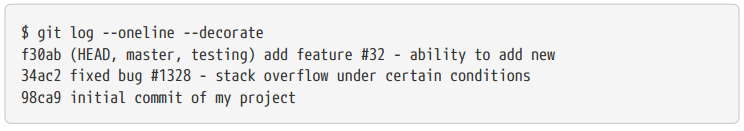
Discards the most recently stashed changeset

* + 1. Просмотр куда указывают ветки

|  |
| --- |
| Уровень освоения: Минимальный |

Она покажет вам, куда указывают указатели веток. Эта опция называется

$ git log --decorate

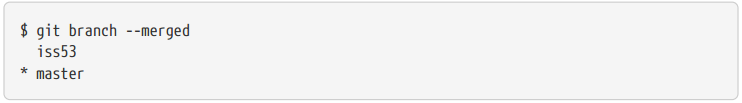


Видны ветки “master” и “testing”, которые указывают на коммит f30ab.

* + 1. Просмотр слитых и неслитых веток

|  |
| --- |
| Уровень освоения: Минимальный |

Ещё одна полезная возможность для выяснения состояния веток состоит в том, чтобы оставить в этом списке только те ветки, которые вы слили (или не слили) в ветку, на которой сейчас находитесь. Для этих целей в Git’е есть опции --merged и --no-merged. Чтобы посмотреть те ветки, которые вы уже слили с текущей, можете выполнить команду git branch --merged:



* 1. Слияние веток

|  |
| --- |
| Общие теги: Метод |
| Уровень освоения: Минимальный |

$ git merge -m 'Message'

// Слияние происходит той ветки в которой мы находимся с той веткой, которую указываем.

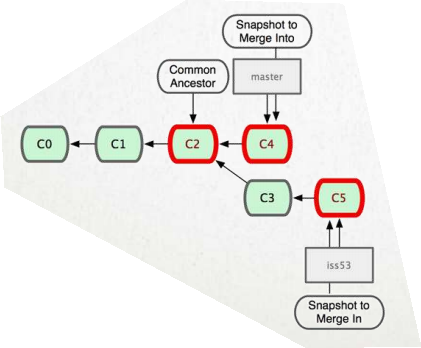
* + 1. Слияние. Комбинирование

|  |
| --- |
| Уровень освоения: Минимальный |

git checkout master

git merge iss53

* Поиск наилучшего общего предка среди коммитов,
* Последовательное применение коммитов к нему,
* Создание нового коммита-слияния.



* + 1. Слияние. Проброс

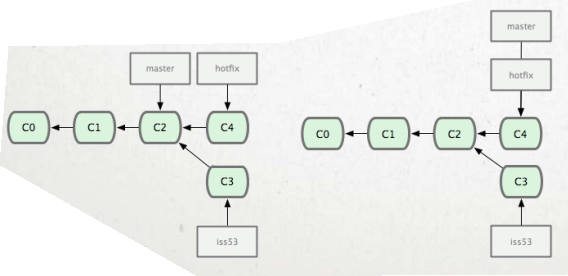
|  |
| --- |
| Уровень освоения: Минимальный |

git checkout master

git merge hotfix

В нашем случае ветки *master* и *hotfix* находятся на одной линии истории.

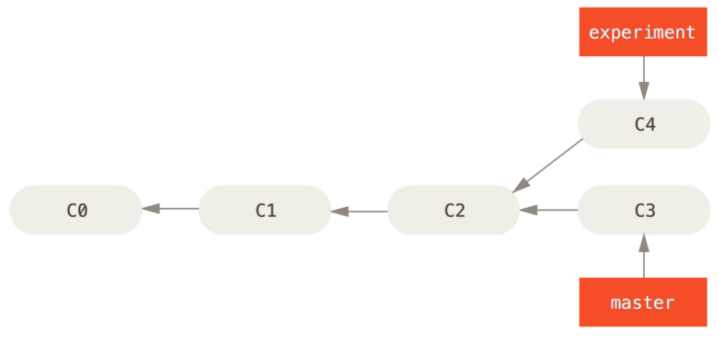
И для их слияния достаточно просто «перемотать» историю вперед, это называется **fast-forward**.



* + 1. Перебазирование

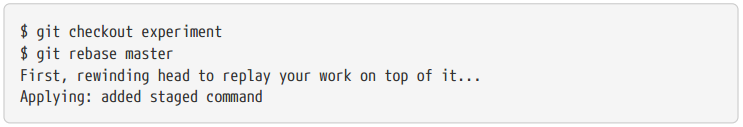
Пример простейшего перебазирования

Расхождение истории по 2 веткам

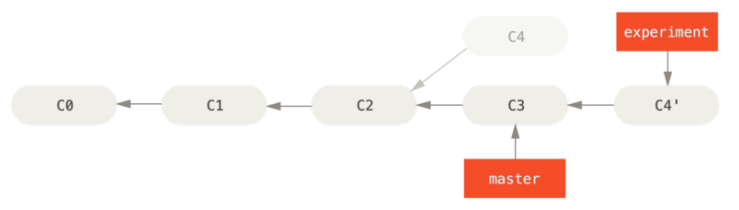


Что сделает перебазирование

Вы можете взять те изменения, что были представлены в C4 и применить их поверх C3. В Git это называется *перебазированием* (rebasing). С помощью команды rebase вы можете взять все изменения, которые были зафиксированы (commited) в одной ветке и применить их к другой ветке.



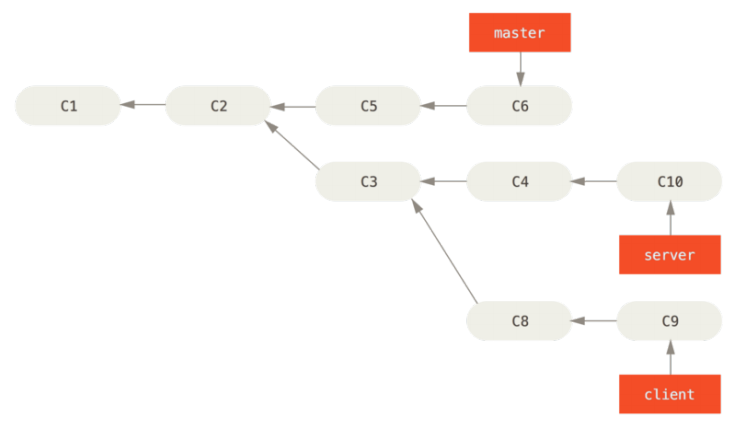
Это работает следующим образом: берется общий родительский снимок (snapshot) двух веток (той, в которой вы находитесь, и той, поверх которой вы выполняете перебазирование); берется дельта (diff) каждого коммита той ветки, на который вы находитесь, эти дельты сохраняются во временные файлы; текущая ветка устанавливается на тот же коммит, что и ветка, поверх которой вы выполняете перебазирование; и, наконец, ранее сохраненные дельты применяются поочереди.



Более сложное перебазирование

Текущая ситуация

Вы создаете тематическую ветку (server), чтобы добавить в проект некоторые функциональности для серверной части, и делаете коммит. Затем вы выполнили ответвление, чтобы сделать изменения для клиентской части, и несколько раз выполнили коммиты. Наконец, вы вернулись на ветку server и сделали ещё несколько коммитов.

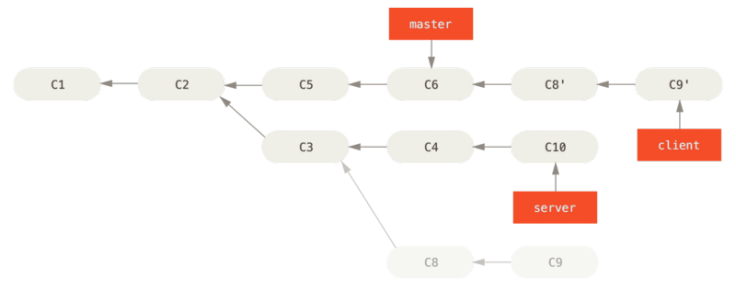


Решение перебазирования

Предположим, вы решили, что хотите внести свои изменения для клиентской части в основную линию разработки для релиза, но при этом хотите оставить в стороне изменения для серверной части до полного тестирования. Вы можете взять изменения из ветки client, которых нет в server (C8 и C9), и применить их на ветке master при помощи опции --onto команды git rebase:

graphic

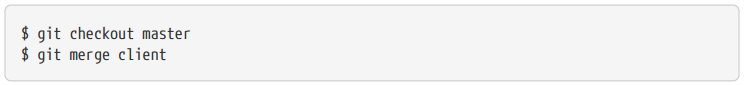
Это прямое указание “переключиться на ветку client, то есть взять изменения от общего предка веток client и server и повторить их на master”. Несмотря на некоторую сложность этого способа, результат впечатляет.

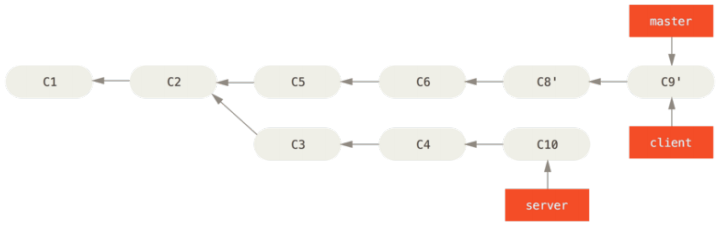


Перемотка

Теперь вы можете выполнить перемотку (fast-forward) для ветки master (см Перемотка ветки

master для добавления изменений из ветки client):

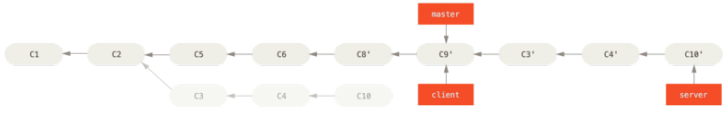




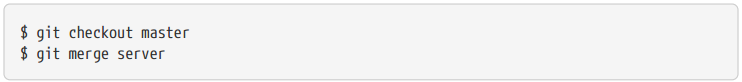
Перебазирование остального кода

Представим, что вы решили добавить наработки и из ветки server. Вы можете выполнить перемещение ветки server на ветку master без предварительного переключения на эту ветку при помощи команды git rebase [осн. ветка] [тем. ветка], которая делает тематическую ветку (в данном случае server) текущей и применяет её изменения к основной ветке (master):

graphic

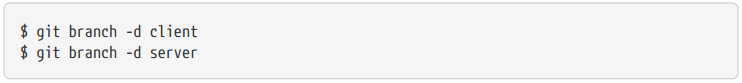


После чего вы сможете выполнить перемотку основной ветки (master):



Теперь вы можете удалить ветки client и server, поскольку весь ваш прогресс уже включен

[в основную линию разработки], и больше нет нужны сохранять эти ветки.





Опасности перемещения (перебазирования)

**Не перемещайте коммиты,**

**уже отправленные в публичный репозиторий**

Слияние vs Перебазирование

Стоит взять лучшее от обоих миров - использовать перемещение (rebase) для локальных изменений, ещё не отправленных на удаленный сервер (push), чтобы навести порядок в вашей истории; но никогда не перемещать (rebase) ничего, что уже было отправлено (push) куда-то.

* 1. Удаление

|  |
| --- |
| Общие теги: Метод |

* + 1. Удаление ветки

git branch -d [branch-name]

* 1. # Что это

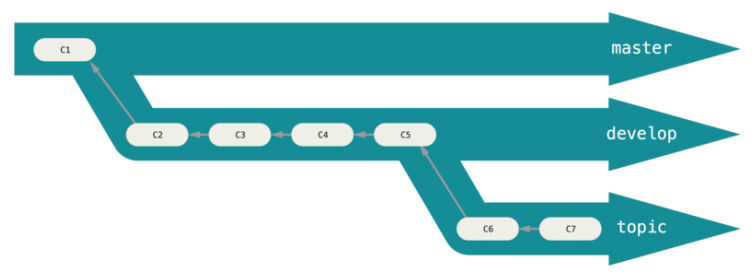
|  |
| --- |
| Общие теги: Применение |

**Что такое «ветка»?**

Ветка в Git – это просто указатель на один из коммитов в истории. При создании репозитория автоматически создается ветка master. Она (то есть указатель) автоматически сдвигается вперед при каждом коммите. Ветка master считается «основной» веткой, хотя это просто договоренность.

* 1. Методология работы с ветками

|  |
| --- |
| Общие теги: Применение |



* 1. Решение конфликтов

|  |
| --- |
| Общие теги: Применение |

**Конфликт** – это невозможность слияния из-за неоднозначности.

• НЕ БОЙТЕСЬ КОНФЛИКТОВ!

• git status покажет файлы с конфликтами. В них вы найдете метки у конфликтых мест

• Чтобы «решить конфликт», вы должны оставить нужный вам вариант или переписать проблемное место заново

• Затем не забудьте

git add ... конфликтного файла – проиндексировать его

• И потом, когда все конфликты решены,

git commit ...

*NB. Решение конфликтов – это та часть «Дао Git», в которой я рекомендую пользоваться графическими инструментами. Меньше возможности ошибиться!*

$ git merge tool

Визуальное средство разрешения конфликтов

*Имеет смысл разобраться в настройке GIT в среде программирования, для решения конфликтов*

* 1. Тематические ветки

|  |
| --- |
| Общие теги: Применение |

А вот такая вещь, как тематические ветки, полезна вне зависимости от величины проекта. Тематической (topic branch) называется временная ветка, создаваемая и используемая для работы над конкретной функциональной возможностью или решения сопутствующих задач.

1. Р- Консоль
   1.  Основы работы с консолью

См. документ(-ы): [Основы линукс: Введение в bash](http://linuxgeeks.ru/bash-intro.htm)

|  |
| --- |
| Общие теги: Разобрать после..., Информация |

* 1. Навигация

|  |
| --- |
| Общие теги: Метод |

$ ls -1

Просмотр списка файлов в активной директории

$ cd АДРЕС

Переход по файлам в системе или в каталоге

$ mkdir НАЗВАНИЕ ПАПКИ

Создание папки

* 1. Переименование файла

|  |
| --- |
| Общие теги: Метод |

$ git mv file\_from file\_to

* 1. Просмотр содержимого

|  |
| --- |
| Общие теги: Метод |

$ cat "ФАЙЛ"

Просмотр содержимого файла

* 1. Создание текстового файла

|  |
| --- |
| Общие теги: Метод |

printf 'Содержимое' > 'название файла.txt'

1. Р- Метки
   1. Метки

|  |
| --- |
| Общие теги: Скилл |

Создание

Легкой

Аннотированной

Для конкретного коммита

Просмотр

Всех меток

Конкретной метки

Удаление метки

Отправка меток на сервер

* 1. Отправка меток на удаленный репозиторий

|  |
| --- |
| Общие теги: Метод |

Метки по умолчанию не передаются в удаленный репозиторий. Используйте команды:

git push origin 1.0

git push origin --tags

* 1. Обзор

|  |
| --- |
| Общие теги: Метод |

* + 1. Просмотр всех меток

$ git tag

Эта команда выведет список всех меток в вашем репозитории

* + 1. Просмотр метки

Вы можете посмотреть данные метки вместе с коммитом, который был помечен, с

помощью команды

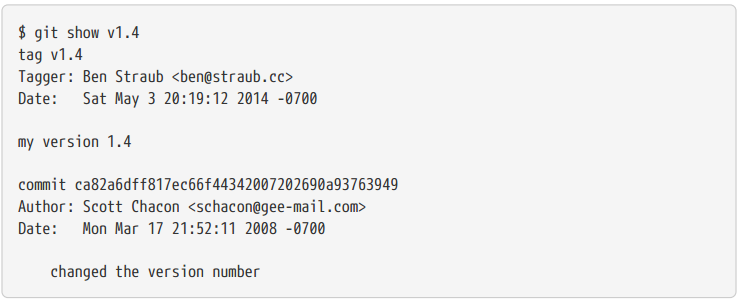
$ git show

git show 1.0

Просмотр информации о заданной метке

git tag -l '1.\*'

Список меток по маске



* 1. Создание

|  |
| --- |
| Общие теги: Метод |

* + 1. Создание аннотированной метки

git tag -a 1.0 -m "Сообщение"

Создает аннотированную метку. В отличие от легкой в ней фиксируется сообщение, дата и время создания, имя создавшего. Более того – аннотированные метки могут быть подписаны GPG.

* + 1. Создание метки для конкретного коммита

Метки можно создавать и позже. Достаточно при создании указать номер уже существующего коммита:

git tag -a 1.0 -m "Сообщение" 1baf

* + 1. Создание легкой метки

git tag 1.0

Эта команда помечает текущий коммит легкой меткой 1.0

* 1. Удаление метки

|  |
| --- |
| Общие теги: Метод |

git tag -d 1.0

Удаление метки. Разумеется, коммит не удаляется!

* 1. Назначение

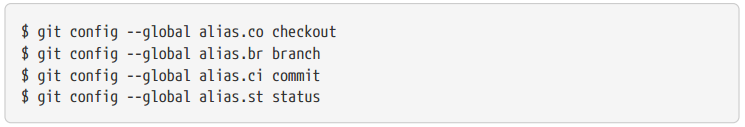
|  |
| --- |
| Общие теги: Применение |

Это метки, которыми можно пометить коммит.

1. Р- Псевдонимы
   1. # Что это?

|  |
| --- |
| Общие теги: Применение |

Это сокращения вводимых комманд:



Это означает, что, например, вместо ввода git commit, вам достаточно набрать только git ci.

* 1. Создание псевдонима

|  |
| --- |
| Общие теги: Метод |

$ git config --global alias.unstage 'reset HEAD --'



Также, обычно, добавляют команду last следующим образом:





1. Р- Репозиторий
   1. Инициализация простейшего репозитория

|  |
| --- |
| Общие теги: Скилл |

1. Переход в папку
2. [Инициализация](#xpointer(/descendant-or-self::ap:Topic[@OId='k27Y7nzTkkG6nLUNjAvIQQ=='])) репозитория
3. [Добавление файлов](#xpointer(/descendant-or-self::ap:Topic[@OId='bfC8JUivBkWQLyBH6Cvh1w=='])) в индекс.
4. [Коммит](#xpointer(/descendant-or-self::ap:Topic[@OId='qXIedaCniky9J4rxTvjUzg==']))

--------------

Далее можем [создать новую ветку](#xpointer(/descendant-or-self::ap:Topic[@OId='u1FfUGg/7kWevHkF6/yT9Q=='])), и начать вносить в ней изменения. Вносим изменения - делаем коммит с обязательным комментарием о внесенных изменениях. Так делаем несколько коммитов, и как только мы понимаем, что цель успешно достигнута, можно [слить](#xpointer(/descendant-or-self::ap:Topic[@OId='zf3ql40D2ku1BjhWxnlF6Q=='])) ветку с основной, и потом [удалить](#xpointer(/descendant-or-self::ap:Topic[@OId='rxVWBAbaeEalnpYecgf5MQ=='])).

--------------

git init

or

git clone url

* 1. Наблюдение за репозиторием

|  |
| --- |
| Общие теги: Скилл |

Чтобы убедиться в каком состоянии сейчас находиться репозиторий (на каком этапе мы находимся) можно воспользоваться [статусом](#xpointer(/descendant-or-self::ap:Topic[@OId='bGlW19vKR0iGtSRxTQktQQ=='])).

Чтобы просмотреть историю репозитория: историю коммитов с комментариями - [мониторингом](#xpointer(/descendant-or-self::ap:Topic[@OId='VwBWAxqbMkutbqMOjVf0gQ=='])).

* 1. Обновление текущей ветки

|  |
| --- |
| Общие теги: Скилл |

# See all commits

git log

# Pretty commit view, you can customize it as much as you want.

# Just google it :)

git log --pretty=format:"%h %s" --graph

# See what you worked on in the past week

git log --author='Alex' --after={1.week.ago} --pretty=oneline --abbrev-commit

# See only changes made on this branch (assuming it was branched form master branch)

git log --no-merges master..

# See status of your current git branch.

# Often will have advice on command that you need to run

git status

# Short view of status. Helpful for seeing things at a glance

git status -s

# Add modified file to be commited(aka stage the file)

git add filename

# Add all modified files to be commited(aka stage all files)

git add .

# Add only text files, etc.

git add '\*.txt'

# Tell git not to track file anymore

git rm filename

# Record changes to git. Default editor will open for a commit message.

# (Visible via git log)

# Once files are commited, they are history.

git commit

# A short hand for commiting files and writing a commit message via one command

git commit -m 'Some commit message'

# Changing the history :) If you want to change your previous commit,

# you can, if you haven't pushed it yet to a remote repo

# Simply make new changes, add them via git add, and run the following command.

# Past commit will be ammended.

git commit --amend

* 1. Откат по истории / Отмена действий

|  |
| --- |
| Общие теги: Скилл |

# Unstage pending changes, the changes will still remain on file system

git reset

# Unstage pending changes, and reset files to pre-commit state. If

git reset --hard HEAD

# Go back to some time in history, on the current branch

git reset tag

git reset <commit-hash>

# Save current changes, without having to commit them to repo

git stash

# And later return those changes

git stash pop

# Return file to it's previous version, if it hasn’t been staged yet.

# Otherwise use git reset filename or git reset --hard filename

git checkout filename

* 1. Сравнение изменений

|  |
| --- |
| Общие теги: Скилл |

# See current changes, that have not been staged yet.

# Good thing to check before running git add

git diff

# See current changes, that have not been commited yet (including staged changes)

git diff HEAD

# Compare current branch to some other branch

git diff branch-name

# Same as diff, but opens changes via difftool that you have configured

# -d tells it to open it in a directory mode, instead of having to open

# each file one at a time.

git difftool -d

# See only changes made in the current branch (compared to master branch)

# Helpful when working on a stand alone branch for a while

git difftool -d master..

# See only the file names that has changed in current branch

git diff --no-commit-id --name-only --no-merges origin/master...

# Similar to above, but see statistics on what files have changed and how

git diff --stat #Your diff condition

* 1. Коммит и снапшот

|  |
| --- |
| Общие теги: Информация |

Репозиторий хранит в себе информацию о состояниях репозитория. Они называются «снапшоты». Каждое состояние – это коммит. Коммит – это снапшот и данные об авторстве.

Коммит снабжен указателями на его родительские коммиты:

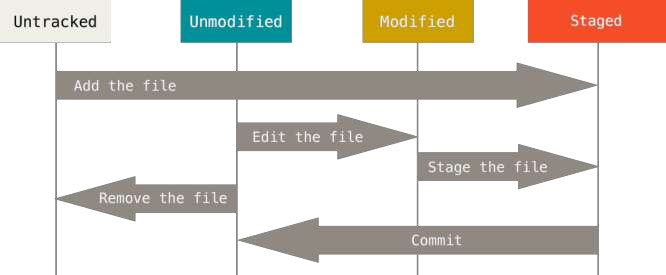
• Ноль – если это первый коммит

• Один – если это «обычный»

• Несколько – в случае слияния изменений (обсудим это позже).

* 1. Основные состояния файлов

|  |
| --- |
| Общие теги: Информация |



неотслеживаемый

не изменившийся, в индексе

изменившийся по сравнению с индексом

подготовленный для фиксации изменений

* 1. Субмодули

|  |
| --- |
| Общие теги: Информация |

**СУБМОДУЛЬ** – это, фактически, репозиторий внутри вашего репозитория.

git submodule add РЕПОЗИТОРИЙ ПАПКА

Добавляет в ваш репозиторий субмодуль.

Выглядит субмодуль как папка, связанная с другим удаленным репозиторием.

После добавления субмодуля нужно зафиксировать изменения, обычной командой git commit.

git submodule init

Инициализирует субмодули в репозитории, где это еще не сделано (например сразу после клонирования)

git submodule update

Загружает файлы субмодулей

* 1. Игнорирование файлов

|  |
| --- |
| Общие теги: Информация |

вы можете создать файл .gitignore. с перечислением шаблонов соответствующих таким файлам. Вот пример файла .gitignore:

$ cat .gitignore

\*.[oa]

\*~

К шаблонам в файле .gitignore применяются следующие правила:

• Пустые строки, а также строки, начинающиеся с #, игнорируются.

• Можно использовать стандартные glob шаблоны.

• Можно начать шаблон символом слэша (/) чтобы избежать рекурсии.

• Можно заканчивать шаблон символом слэша (/) для указания каталога.

• Можно инвертировать шаблон, использовав восклицательный знак (!) в качестве

первого символа.

Glob-шаблоны представляют собой упрощённые регулярные выражения, используемые командными интерпретаторами. Символ (\*) соответствует 0 или более символам; последовательность [abc] — любому символу из указанных в скобках (в данном примере a, b или c); знак вопроса (?) соответствует одному символу; и квадратные скобки, в которые заключены символы, разделённые дефисом ([0-9]), соответствуют любому символу из интервала (в данном случае от 0 до 9). Вы также можете использовать две звёздочки, чтобы указать на вложенные директории: a/\*\*/z соответствует a/z, a/b/z, a/b/c/z, и так далее.

Вот ещё один пример файла .gitignore:

# no .a files

\*.a

# but do track lib.a, even though you're ignoring .a files above

!lib.a

# only ignore the root TODO file, not subdir/TODO

/TODO

# ignore all files in the build/ directory

build/

# ignore doc/notes.txt, but not doc/server/arch.txt

doc/\*.txt

# ignore all .txt files in the doc/ directory

doc/\*\*/\*.txt

* 1. 1. Создание

|  |
| --- |
| Общие теги: Метод |

* + 1. Индексирование файлов, включение в GIT

|  |
| --- |
| Уровень освоения: Минимальный |

git add – универсальная команда:

Добавляет новый файл под контроль git, Сообщает git, что файл изменился и подготавливает изменения (индексирует файл)

$ git add .

Индексация всей папки

git add css/main.css css/button.css

Внесение файлов в индекс по отдельности.

говорит Git о новых файлах, созданных тобой, которые нужны тебе в следующем коммите.

* + 1. Создание репозитория

|  |
| --- |
| Уровень освоения: Минимальный |

Создаем репозиторий в пустой папке:

$ git init [project name]

В результате мы создали в нужной нам папке репозиторий – то есть служебную папку .git с необходимой информацией.

* 1. 2. Редактирование

|  |
| --- |
| Общие теги: Метод |

* + 1. Восстановление файла из коммита

git checkout №hash ФАЙЛ / Восстановление файла из коммита. Сброс файла к состоянию последнего коммита / Восстановление удаленного файла.

*NB. Помните, что всё, что является частью коммита в Git'е, почти всегда может быть восстановлено. Всё, что никогда не попадало в коммит, вы скорее всего уже не увидите снова.*

* + 1. Дописать инф. в последний коммит

git commit --amend

– добавление изменений к последнему коммиту

«Догнать» последний коммит. Используется, чтобы добавить в него забытые изменения или изменить его message. Всё, что подготовлено для фиксации, будет зафиксировано в последнем коммите, не создавая новый.

* + 1. Исключение файла из индекса

|  |
| --- |
| Уровень освоения: Минимальный |

$ git reset HEAD *index.html*

Исключает файл из индексации

* + 1. Исправление сообщения коммита

git commit --amend -m 'Сообщение'

То же самое, с той лишь разницей что нет необходимости добавлять файлы в коммит. Просто укажите новое название:

git commit --amend -m "Новое название"

* + 1. Откат изменений

|  |
| --- |
| Уровень освоения: Минимальный |

git reset --soft [commit]

«Мягкий» откат репозитория в состояние указанного коммита. Все изменения, бывшие ранее коммитами после указанного, останутся в индексе. И вы снова можете их зафиксировать.

git reset --hard [commit]

Откат репозитория в состояние указанного коммита. Обратите внимание – коммиты после указанного пропадают из git log, но на самом деле все еще хранятся в репозитории.

git reset [commit]

Отмена всех изменений после указанного коммита

* + 1. Отмена индексации файла

|  |
| --- |
| Уровень освоения: Минимальный |

git reset HEAD ФАЙЛ

Отмена индексации, то есть отмена подготовки файла к фиксации. Сам файл при этом не меняется!

* + 1. Отмена истории изменений

git reset HEAD

- откат к последнему коммиту (отмена индексации)

git reset --hard <commit>

- самая мощная и самая опасная команда – откат к указанному коммиту с потерей всех изменений!

<commit> == HEAD~3

- откат на 3 коммита назад

* + 1. Удаление файла из индекса / Рабочей папки

|  |
| --- |
| Уровень освоения: Минимальный |

$ git rm FILE –

удаляет файл из индекса и из рабочей папки

$ git rm -f FILE

принудительное удаление проиндексированного файла

$ git rm --cached FILE –

удаляет файл только из индекса git.

*Оба вида удаления требуют фиксации!*

Можно передавать файлы, каталоги или glob-шаблон

* 1. 3. Мониторинг

|  |
| --- |
| Общие теги: Метод |

* + 1. История изменений

|  |
| --- |
| Уровень освоения: Минимальный |

$ git log

просмотр коммитов (фиксаций изменений, истории).

$ git log -1

Просмотр колличества последних записей, лимит коммитов

$ git log --oneline

просмотр истории в одну строку

$ git log --oneline --all

просмотр всей истории

$ git cat-file -p HASHCOMMIT

Просмотр инфы о коммите

$ git log --oneline --all --graph

Графическое представление ветвления

$ git log -p

Покажет разницу, внесенную в каждый коммит

$ git log --pretty

Меняет формат вывода лог

git log --oneline --graph --decorate

Удобочитаемый формат вывода лога

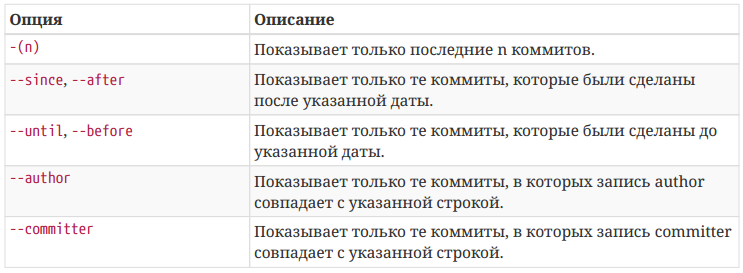
git log --follow [file]

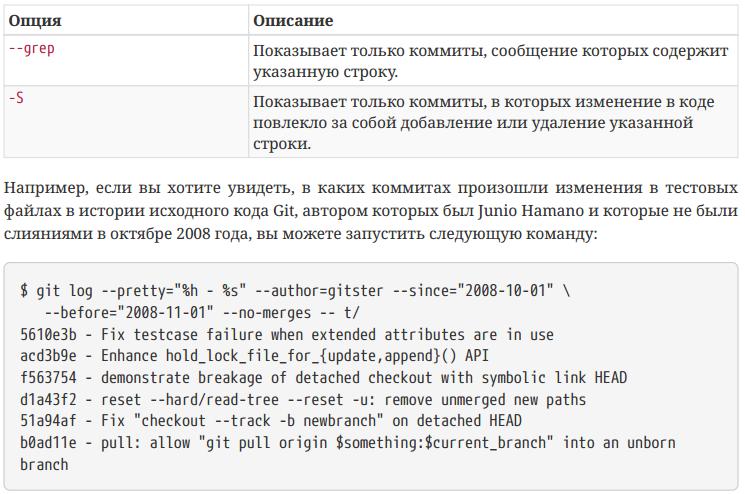
Покажет историю версий файла, включая переименования

git diff [first-branch]...[second-branch]

Покажет разницу в контенте между двумя ветками

Доп параметры





* + 1. История переходов по коммитам HEAD

$ git reflog

Просмотр истории изменения коммитов

* + 1. Просмотр коммита

|  |
| --- |
| Уровень освоения: Минимальный |

$ git show №*hash*

*Покажет изменения зафиксированные в коммите*

* + 1. Просмотр статуса

|  |
| --- |
| Уровень освоения: Минимальный |

$ git status - рассказывает тебе о текущем состоянии твоей кодовой базы: в какой ветке ты находишься, какие изменения ты сделал и т.д.

$ git diff - Покажет разницу между внесенными изменениями и начальным состоянием коммита.

$ git diff --staged Покажет проиндексированные изменения

$ git status -s или $ git status --short вы получите гораздо более упрощенный вывод.

* 1. 5. Коммит

|  |
| --- |
| Общие теги: Метод |

* + 1. Коммит

|  |
| --- |
| Уровень освоения: Минимальный |

$ git commit –m "комментарий" // Кратко

$ git commit // Через редактор Как поменять [редактор](#xpointer(/descendant-or-self::ap:Topic[@OId='y6VMtxh2ekOtus9aovADBA==']))

$ git commit -a -m "MESSAGE" // Автоиндексация отслеживаемых файлов с добавлением коммита

* + 1. Обновить коммит

1) Внести требуемые правки в файлы

2) Проиндексировать изменения и ввести

git commit --amend

Обновление текущего коммита.

* 1. # Что это

|  |
| --- |
| Общие теги: Применение |

* Репозиторий – это просто папка с файлами, в которой есть скрытая папка .git
* В этой скрытой папке содержится вся нужная служебная информация – история всех изменений нашего репозитория
* А сам git – это просто набор скриптов, иначе – команд в консоли!

1. Р- Репозиторий удаленный
   1. Удаленный Р

|  |
| --- |
| Общие теги: Скилл |

* + 1. А- Удаленный Репозиторий

Пример работы с удаленной веткой

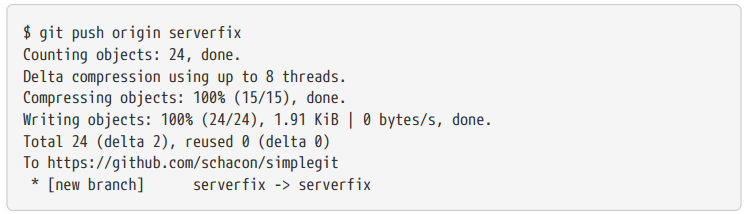
А если несколько веток и УР подключено?

Отправка изменений на сервер

|  |
| --- |
| Уровень освоения: Минимальный |

Когда вы хотите поделиться веткой с окружающими, вам необходимо отправить (push) её на удалённый сервер, на котором у вас есть права на запись. Ваши локальные ветки автоматически не синхронизируются с удалёнными серверами — вам нужно явно отправить те ветки, которыми вы хотите поделиться. Таким образом, вы можете использовать свои личные ветки для работы, которую вы не хотите показывать, и отправлять только те тематические ветки, над которыми вы хотите работать с кем-то совместно.

Если у вас есть ветка serverfix, над которой вы хотите работать с кем-то ещё, вы можете отправить её точно так же, как вы отправляли вашу первую ветку. Выполните **git push (удал. сервер) (ветка)**:



В иную ветку

|  |
| --- |
| Уровень освоения: Минимальный |

Вы также можете выполнить

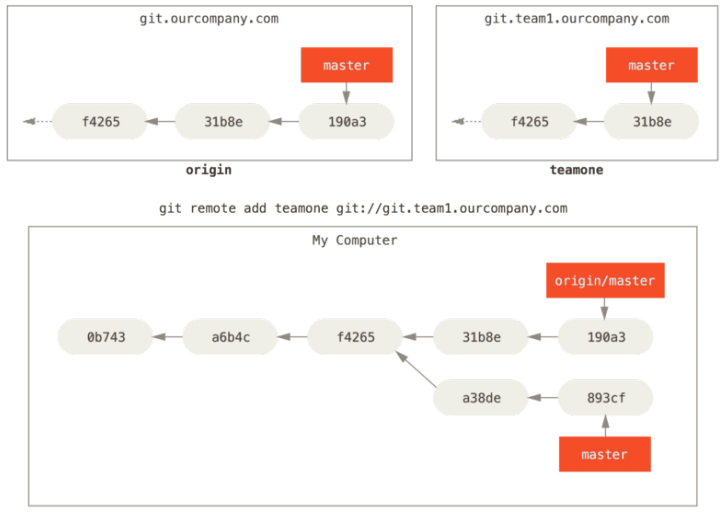
**$ git push origin serverfix:serverfix**

— произойдёт то же самое — здесь говорится “возьми мой serverfix и сделай его удалённым serverfix.”

Можно использовать этот формат для отправки локальной ветки в удалённую ветку с другим именем

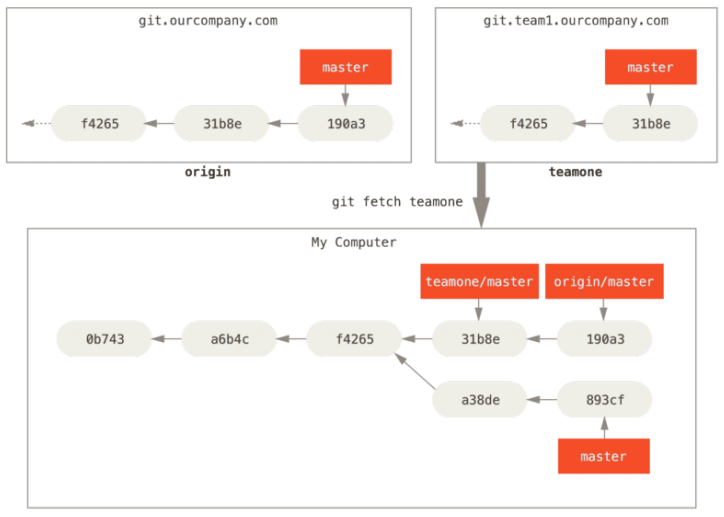
Подключение УР в качестве ссылки

Предположим, что у вас есть ещё один внутренний Git-сервер, который используется для разработки только одной из ваших команд разработчиков. Этот сервер находится на git.team1.ourcompany.com. Вы можете добавить его в качестве новой удалённой ссылки на проект, над которым вы сейчас работаете с помощью команды git remote add так же, как было описано в Основы Git. Дайте этому удалённому серверу имя teamone, которое будет сокращением для полного URL.



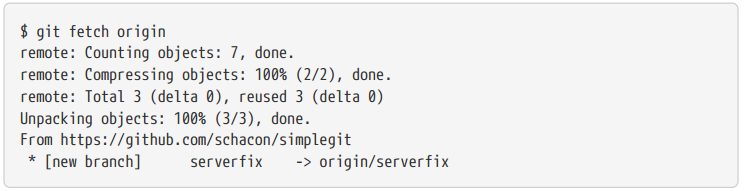
Получение всех сведений с УР

Теперь можете выполнить **git fetch teamone**, чтобы извлечь всё, что есть на сервере и нет у вас. Так как в данный момент на этом сервере есть только часть данных, которые есть на сервере *origin*, Git не получает никаких данных, но выставляет удалённую ветку с именем *teamone/master*, которая указывает на тот же коммит, что и ветка *master* на сервере *teamone*.



Получение изменений с сервера коллегой

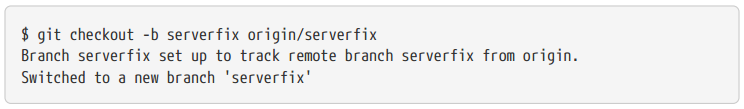
В следующий раз, когда один из ваших соавторов будет получать обновления с сервера, он получит ссылку на то, на что указывает *serverfix* на сервере, как удалённую ветку **origin/serverfix:**



Создание локальной ветки и соединение с удаленной

Важно отметить, что когда при получении данных у вас появляются новые удалённые ветки, вы не получаете автоматически для них локальных редактируемых копий. Другими словами, в нашем случае вы не получите новую ветку serverfix — только указатель origin/serverfix, который вы не можете менять.

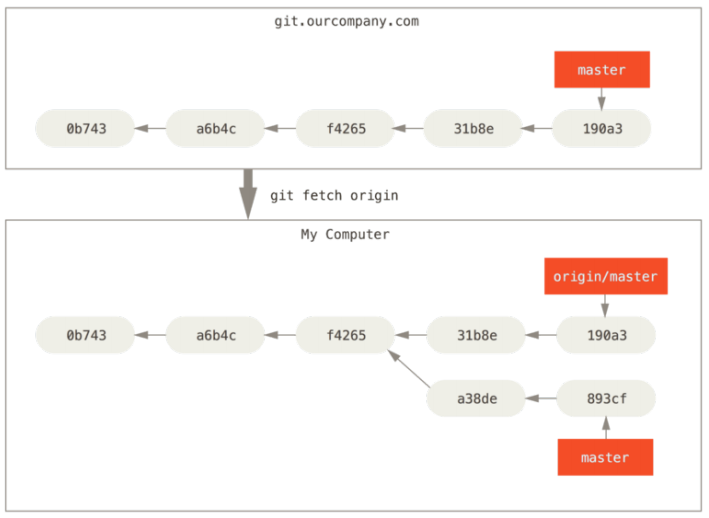
Чтобы слить эти наработки в свою текущую рабочую ветку, выполните git merge origin/serverfix. Если вам нужна своя собственная ветка serverfix, над которой вы сможете работать, то вы можете создать её на основе удалённой ветки:



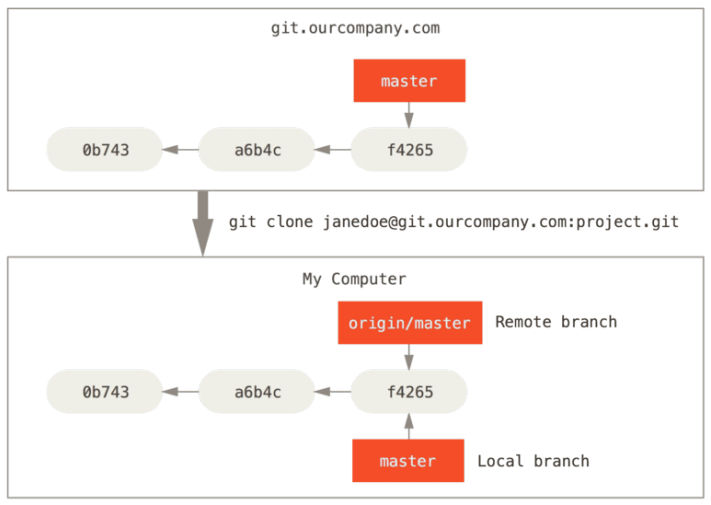
Это даст вам локальную ветку, на которой можно работать. Она будет начинаться там, где и **origin/serverfix**

Как синхронизировать вашу работу?

Для синхронизации вашей работы выполняется команда **git fetch origin**. Эта команда ищет, какому серверу соответствует **“origin”** (в нашем случае это git.ourcompany.com); извлекает оттуда все данные, которых у вас ещё нет, и обновляет ваше локальное хранилище данных; сдвигает указатель **origin/master** на новую позицию.



Клонирование УР



Отслеживание веток

|  |
| --- |
| Уровень освоения: Минимальный |

Если, находясь на отслеживаемой ветке, вы наберёте git pull, Git уже будет знать, с какого сервера получить все удалённые ссылки и сделает слияние с соответствующей удалённой веткой. Аналогично выполнение git pull на одной из таких веток, сначала получает все удалённые ссылки, а затем автоматически делает слияние с соответствующей удалённой веткой.

Запуск локальной ветки с отслеживанием, и другим именем

Принудительный запуск отслеживания удаленной ветки

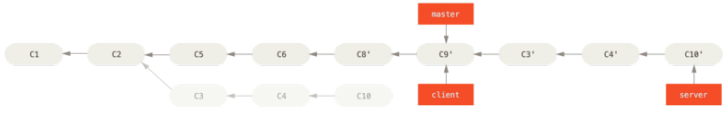
Перебазирование

Более сложное перебазирование

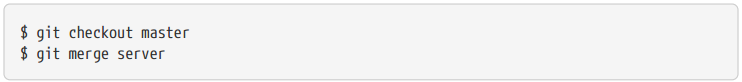
Перебазирование остального кода

Представим, что вы решили добавить наработки и из ветки server. Вы можете выполнить перемещение ветки server на ветку master без предварительного переключения на эту ветку при помощи команды git rebase [осн. ветка] [тем. ветка], которая делает тематическую ветку (в данном случае server) текущей и применяет её изменения к основной ветке (master):

graphic

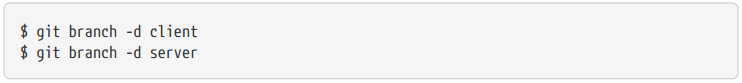


После чего вы сможете выполнить перемотку основной ветки (master):



Теперь вы можете удалить ветки client и server, поскольку весь ваш прогресс уже включен

[в основную линию разработки], и больше нет нужны сохранять эти ветки.

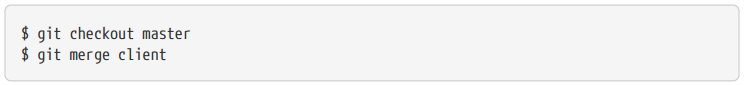


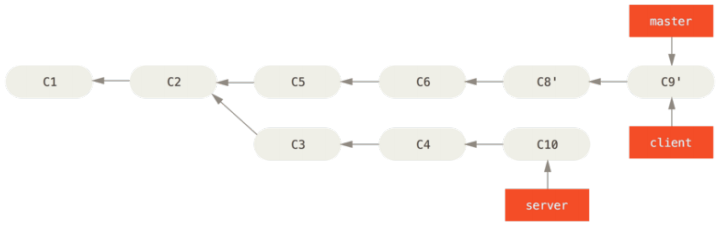


Перемотка

Теперь вы можете выполнить перемотку (fast-forward) для ветки master (см Перемотка ветки

master для добавления изменений из ветки client):



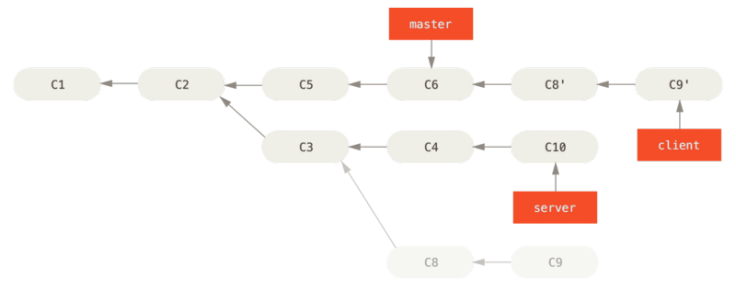


Решение перебазирования

Предположим, вы решили, что хотите внести свои изменения для клиентской части в основную линию разработки для релиза, но при этом хотите оставить в стороне изменения для серверной части до полного тестирования. Вы можете взять изменения из ветки client, которых нет в server (C8 и C9), и применить их на ветке master при помощи опции --onto команды git rebase:

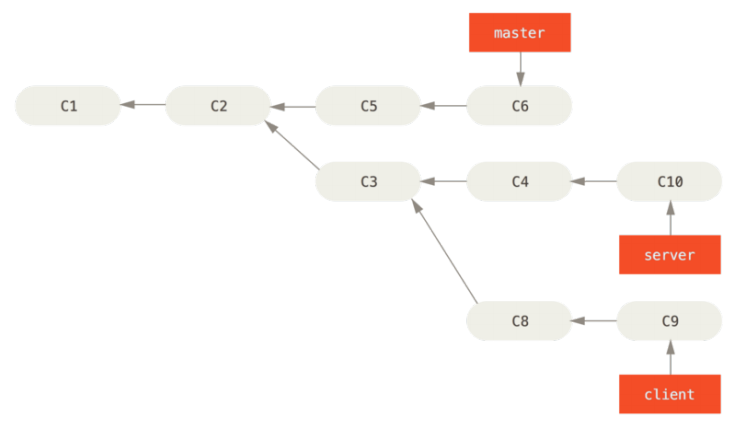
graphic

Это прямое указание “переключиться на ветку client, то есть взять изменения от общего предка веток client и server и повторить их на master”. Несмотря на некоторую сложность этого способа, результат впечатляет.



Текущая ситуация

Вы создаете тематическую ветку (server), чтобы добавить в проект некоторые функциональности для серверной части, и делаете коммит. Затем вы выполнили ответвление, чтобы сделать изменения для клиентской части, и несколько раз выполнили коммиты. Наконец, вы вернулись на ветку server и сделали ещё несколько коммитов.



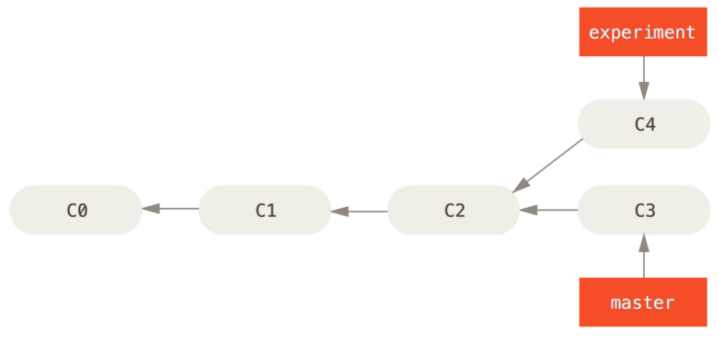
Опасности перемещения (перебазирования)

**Не перемещайте коммиты,**

**уже отправленные в публичный репозиторий**

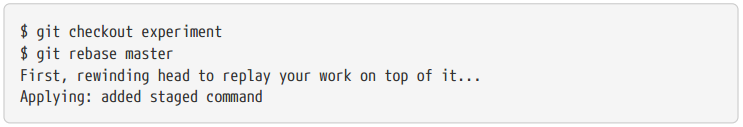
Пример простейшего перебазирования

Расхождение истории по 2 веткам

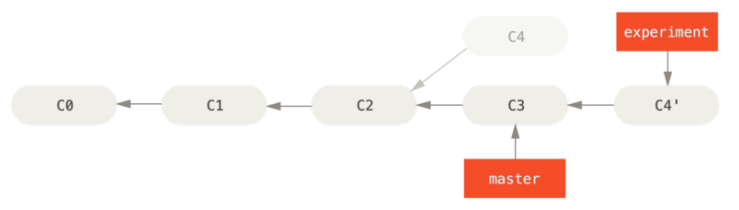


Что сделает перебазирование

Вы можете взять те изменения, что были представлены в C4 и применить их поверх C3. В Git это называется *перебазированием* (rebasing). С помощью команды rebase вы можете взять все изменения, которые были зафиксированы (commited) в одной ветке и применить их к другой ветке.



Это работает следующим образом: берется общий родительский снимок (snapshot) двух веток (той, в которой вы находитесь, и той, поверх которой вы выполняете перебазирование); берется дельта (diff) каждого коммита той ветки, на который вы находитесь, эти дельты сохраняются во временные файлы; текущая ветка устанавливается на тот же коммит, что и ветка, поверх которой вы выполняете перебазирование; и, наконец, ранее сохраненные дельты применяются поочереди.



Слияние vs Перебазирование

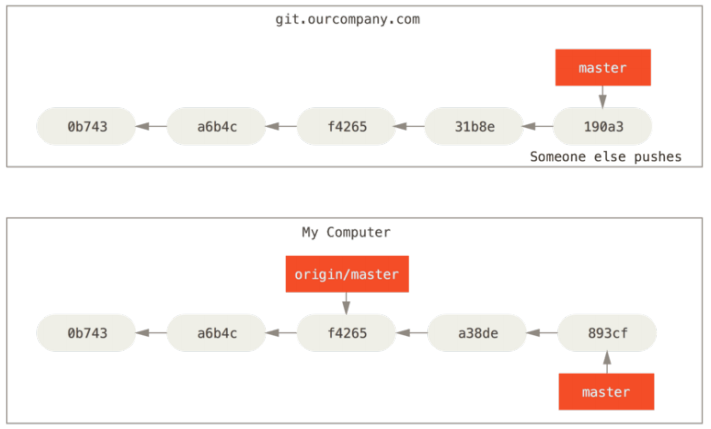
Стоит взять лучшее от обоих миров - использовать перемещение (rebase) для локальных изменений, ещё не отправленных на удаленный сервер (push), чтобы навести порядок в вашей истории; но никогда не перемещать (rebase) ничего, что уже было отправлено (push) куда-то.

Получение изменений

Pull

Сценарий событий

Если вы сделаете что-то в своей локальной ветке master, а тем временем кто-то ещё отправит (push) изменения на git.ourcompany.com и обновит там ветку master, то ваши истории продолжатся по-разному. Ещё, до тех пор, пока вы не свяжетесь с сервером origin, ваш указатель origin/master не будет сдвигаться.



Удаление ветки с сервера

Скажем, вы и ваши соавторы закончили с нововведением и слили его в ветку master на удалённом сервере (или в какую-то другую ветку, где хранится стабильный код). Вы можете удалить ветку на удалённом сервере, используя опцию --delete для git push.

В основном всё, что делает эта строка, — удаляет указатель на сервере. Как правило, Git- сервер оставит данные на некоторое время, пока не запустится сборщик мусора. Итак, если ветка случайно была удалена, чаще всего ее легко восстановить.

Хотим посмотреть какие ветки отслеживаются и актуальное состояние

Итак, здесь мы можем видеть, что наша iss53-ветка отслеживает origin/iss53 и она опережает на два изменения, означающее, что мы имеем два локальных коммита, которые не отправлены на сервер. Мы может также увидеть, что наша master-ветка отслеживает origin/master и она в актуальном состоянии. Далее мы можем видеть, что наша serverfix -ветка отслеживает server-fix-good-ветку на нашем teamone-сервере и опережает на три коммита и отстает на один, означающее, что есть один коммит на сервере, который мы еще не слили и три локальных коммита, которые вы еще не отправили. В конце мы видим, что наша testing-ветка не отслеживает удаленную ветку.

Важно отметить, что эти цифры — только с каждого сервера, которые последний раз были извлечены. Эта команда не обращается к серверам, она говорит вам о том, что в кэше есть локальная информация с серверов. Если вы хотите полностью быть в курсе опережающих и отстающих коммитов, вам необходимо извлечь данные из всех ваших удаленных серверов перед запуском этой команды. Вы можете сделать нечто подобное:

$ git fetch --all

$ git branch -vv

Соединение / Клонирование УР

Клонирование УР

Если вы склонируете удаленный репозиторий, Git-команда clone:

1. автоматически назовёт его origin;
2. заберёт оттуда все данные;
3. создаст указатель на то, на что там указывает ветка master;
4. и назовёт его локально origin/master;
5. Git также сделает вам вашу собственную локальную ветку master, которая будет начинаться там же, где и ветка master в origin;

так что вам будет с чем работать.

Подключение сервера

* + 1. Подключение
    2. Работа с удаленными ветками

Отправка своей ветки на сервер

Подключение отлеживания удаленной ветки

Создание отслеживаемой удаленной ветки

* + 1. Редактирование У.Р.

Клонирование

Переименование

Получение сведений

Со связанной ветки

Просмотр

Удаление

* 1. Работа с удаленной веткой

|  |
| --- |
| Общие теги: Скилл |

# See list of remote repos available. If you did git clone,

# you'll have at least one named "origin"

git remote

# Detailed view of remote repos, with their git urls

git remote -v

# Add a new remote. I.e. origin if it is not set

git remote add origin <https://some-git-remote-url>

# Push current branch to remote branch (usually with the same name)

# called upstream branch

git push

# If a remote branch is not set up as an upstream, you can make it so

# The -u tells Git to remember the parameters

git push -u origin master

# Otherwise you can manually specify remote and branch to use every time

git push origin branchname

# Just like pushing, you can get the latest updates from remote.

# By defaul Git will try to pull from "origin" and upstream branch

git pull

# Or you can tell git to pull a specific branch

git pull origin branchname

# Git pull, is actually a short hand for two command.

# Telling git to first fetch changes from a remote branch

# And then to merge them into current branch

git fetch && git merge origin/remote-branch-name

# If you want to update history of remote branches, you can fetch and purge

git fetch -p

# To see the list of remote branches

# -a stands for all

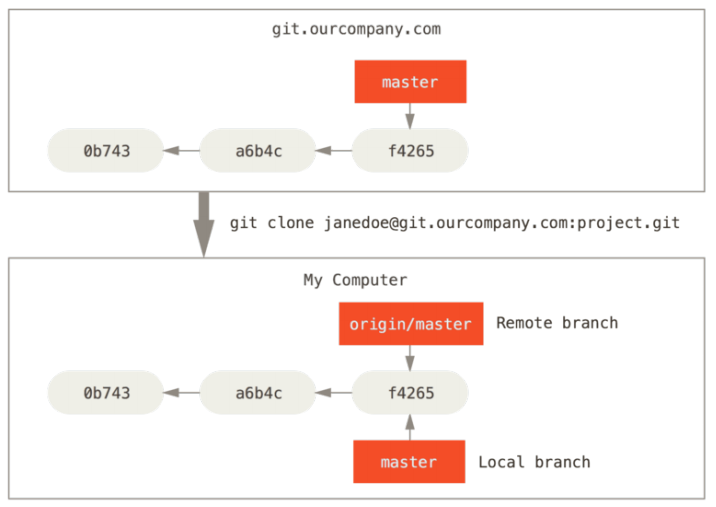
git branch -a

* 1. Как можно работать с У.Р.

|  |
| --- |
| Общие теги: Информация |

Скажем, у вас в сети есть свой Git-сервер на git.ourcompany.com. Если вы с него что-то склонируете, Gitкоманда clone автоматически назовёт его origin, заберёт оттуда все данные, создаст указатель на то, на что там указывает ветка master, и назовёт его локально origin/master. Git также сделает вам вашу собственную локальную ветку master, которая будет начинаться там же, где и ветка master в origin, так что вам будет с чем работать.

***“origin”*** *— это не специальное название. Подобно тому, как название ветки “master” не имеет какого-либо специального значения в Git, также и название “origin”. В то время как “master” — это название по умолчанию для исходной ветки, когда вы запускаете git init, по единственной причине, что широко используется, “origin” — это название по умолчанию для удаленной ветки, когда вы запускаете git clone. Если вы запустите git clone -o booyah, так вы будете иметь booyah/master как вашу удаленную ветку по умолчанию.*



Если вы сделаете что-то в своей локальной ветке master, а тем временем кто-то ещё отправит (push) изменения на git.ourcompany.com и обновит там ветку master, то ваши истории продолжатся по-разному. Ещё, до тех пор, пока вы не свяжетесь с сервером origin, ваш указатель origin/master не будет сдвигаться.

* 1. 1. Подключение

|  |
| --- |
| Общие теги: Метод |

* + 1. Клонирование репозитория

|  |
| --- |
| Уровень освоения: Минимальный |

$ git clone [https://github.com/libgit2/libgit2]

Комманда добавит репозиторий под именем **Origin**

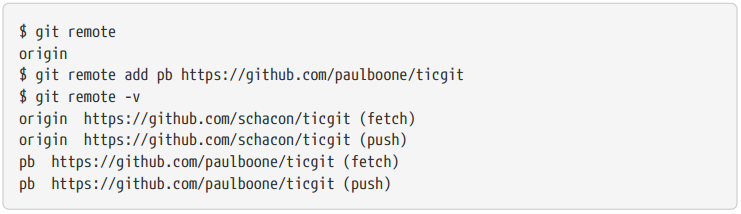
* + 1. Подключение к репозиторию

**Поключение ссылки на удаленный проект**

git remote add [NAME] [URL]

git remote –v // Проверка на успешность подключения удаленного репозитория

git remote show NAME



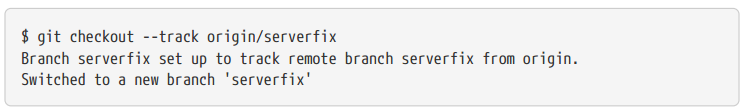
* + 1. Подключение отслеживания других удаленных веток

|  |
| --- |
| Уровень освоения: Минимальный |

Простой пример, как это сделать:

git checkout -b [ветка] [удал. сервер]/[ветка]

Существует общепринятая операция, которую git предоставляет, --track:



* + 1. Подключение удаленного репозитория

|  |
| --- |
| Уровень освоения: Минимальный |

**Создание нового репозитория и связывание:**

echo "# Barbershop\_training" >> README.md //

git init

git add README.md

git commit -m "first commit"

git remote add origin git@github.com:Shalunoff-m/Barbershop\_training.git

git push -u origin master

-----------

**Отправка существующего репозитория и связывание:**

…or push an existing repository from the command line

git remote add origin git@github.com:Shalunoff-m/Barbershop\_training.git

git push -u origin master

* 1. 2. Обмен информацией

|  |
| --- |
| Общие теги: Метод |

* + 1. Push / Pull

|  |
| --- |
| Уровень освоения: Минимальный |

Если у вас есть ветка, настроенная на отслеживание удалённой ветки, то вы можете использовать команду $ git pull чтобы автоматически получить изменения из удалённой ветви и слить их со своей текущей ветвью.

$ git clone автоматически настраивает вашу локальную ветку master на отслеживание удалённой ветки master на сервере. Выполнение $ git pull, как правило, извлекает (fetch) данные с сервера, с которого вы изначально склонировали, и автоматически пытается слить (merge) их с кодом, над которым вы в данный момент работаете.

Когда вы хотите поделиться своими наработками, вам необходимо отправить (push) их в главный репозиторий. Команда для этого действия простая:

$ git push [remote-name] [branchname]

graphic

* + 1. Push локальной ветки на удаленную со связыванием

|  |
| --- |
| Уровень освоения: Минимальный |

Если сделать git push ветки без ключа -u, Git не свяжет локальную ветку с веткой удалённого репозитория. Последующая операция git pull в этой ветке будет неудачной, так как Git не будет знать, из какой удалённой ветки подтягивать изменения, это придётся указать явно при помощи команды git branch --set-upstream-to=origin/master. Смысл использовать ключ -u есть только при пуше новых веток, для существующих (связанных с удалёнными) веток каждый раз перезаписывать связку необязательно.

--------------

С ключом -u (полный вариант --set-upstream) вы создаёте (если ещё не существует) в удалённом репозитории ветку, соответствующую вашей локальной и связываете их:

*В remote/branchname будет производиться push в локальную ветку branchname*

*Из remote/branchname будет производиться pull в локальную ветку branchname*

Для master это актуально, только если вы пушите в пустой репозиторий. Если клонировали - то соответствие ветвей уже настроено. А вот как только вы создадите новую локальную ветвь

# есть у нас локальная ветка  
git checkout -b mybranch

# Создаем ветку на remote  
git push -u origin mybranch

# Можно выбрать другое имя для создаваемой ветки на remote.  
git push -u origin mybranch\_with\_other\_name и захотите её запушить на remote, вам понадобится ключ -u.

* + 1. Аутентификация КЭШ

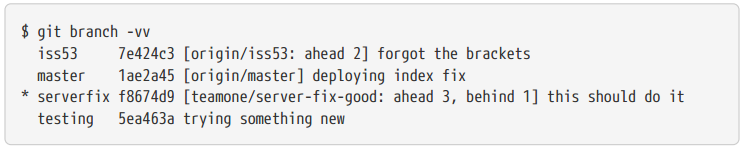
Не вводите каждый раз свой пароль Если вы используете HTTPS URL для отправки изменений, Git-сервер спросит имя пользователя и пароль для аутентификации. По умолчанию вам будет предложено ввести в терминале эту информацию, чтобы сервер мог сказать, что вам разрешена отправка изменений.

Если вы не хотите каждый раз вводить ваши данные, когда вы отправляете изменения, вы можете установить “кэш учетных данных”. Проще всего просто держать их в памяти несколько минут, вы можете легко настроить с помощью команды

$ git config --global credential.helper cache.

* + 1. Обзор установленных отслеживаемых веток

|  |
| --- |
| Уровень освоения: Минимальный |



git branch -r

Просмотр всех удаленных веток

git branch -a

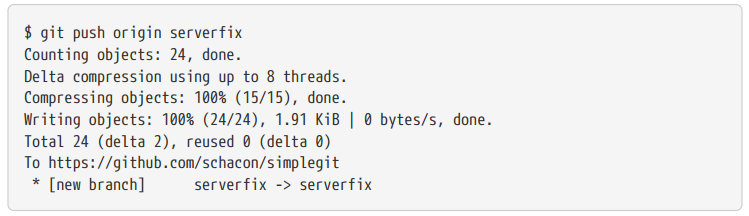
Просмотр всех локальных и удаленных веток

* + 1. Отправка изменений

|  |
| --- |
| Уровень освоения: Минимальный |

Когда вы хотите поделиться веткой с окружающими, вам необходимо отправить (push) её на удалённый сервер, на котором у вас есть права на запись. Ваши локальные ветки автоматически не синхронизируются с удалёнными серверами — вам нужно явно отправить те ветки, которыми вы хотите поделиться. Таким образом, вы можете использовать свои личные ветки для работы, которую вы не хотите показывать, и отправлять только те тематические ветки, над которыми вы хотите работать с кем-то совместно.

Если у вас есть ветка serverfix, над которой вы хотите работать с кем-то ещё, вы можете отправить её точно так же, как вы отправляли вашу первую ветку. Выполните git push (удал. сервер) (ветка):



* + 1. Получение изменений

|  |
| --- |
| Уровень освоения: Минимальный |

Существует команда под названием git pull, которая является по существу командой git fetch, непосредственно за которой следует команда git merge, в большинстве случаев.

* + 1. Получение информации с У.Р.

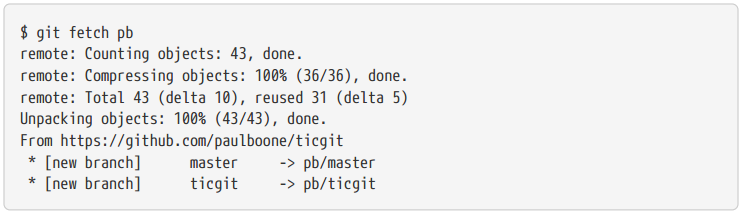
|  |
| --- |
| Уровень освоения: Минимальный |

**Fetch**

Получение из удаленного репозитория информации, которой еще нет в локальном, например ссылки на ветки, которых у нас еще нет.

**Внимание! Команда git fetch никак не изменяет наши файлы!**

$ git fetch [NAME]



$ git fetch origin

Команда git fetch загрузит с сервера все изменения, которых у вас еще нет, но пока не будет изменять вашу рабочую директорию. Эта команда просто получает данные для вас и позволяет вам самостоятельно сделать слияние.

* + 1. Получение удаленных веток c отслеживанием

|  |
| --- |
| Уровень освоения: Минимальный |

Важно отметить, что когда при получении данных у вас появляются новые удалённые ветки, вы не получаете автоматически для них локальных редактируемых копий. Другими словами, в нашем случае вы не получите новую ветку serverfix — только указатель origin/serverfix, который вы не можете менять. Чтобы слить эти наработки в свою текущую рабочую ветку, выполните

$ git merge origin/serverfix

Если вам нужна своя собственная ветка serverfix, над которой вы сможете работать, то вы можете создать её на основе удалённой ветки:

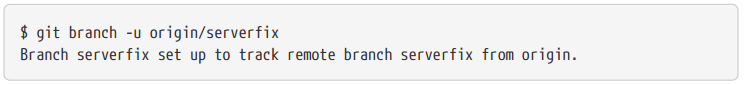


Это даст вам локальную ветку, на которой можно работать. Она будет начинаться там, где и origin/serverfix.

* + 1. Принудительное соединение с удаленной веткой в любое время

|  |
| --- |
| Уровень освоения: Минимальный |

Если у вас уже есть локальная ветка и вы хотите настроить ее на удаленную ветку, которую вы только получили, или хотите изменить *upstream-ветку*, которую вы отслеживаете, вы можете воспользоваться ключами -u или --set-upstream-to с командой git branch, чтобы явно установить ее в любое время.



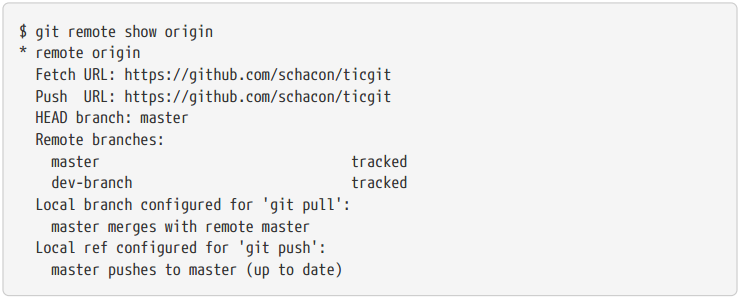
Если у вас есть установленная отслеживаемая ветка, вы можете ссылаться на нее с помощью @{upstream} или @{u} сокращенно. Итак, если вы находитесь на master-ветке, а отслеживается origin/master, вы можете вызвать что-то вроде git merge @{u} вместо git merge origin/master если хотите.

* + 1. Просмотр удаленного репозитория / всех репозиториев

git remote

Покажет какие есть удаленные репозитории

git remote show [remote-name]



* 1. 3. Редактирование

|  |
| --- |
| Общие теги: Метод |

* + 1. Запуск локальной ветки с отличающимся именем от удаленной ветки

Чтобы настроить локальную ветку с именем, отличным от имени удалённой ветки, вы можете легко использовать первую версию с другим именем локальной ветки:



Теперь ваша локальная ветка sf будет автоматически получать (pull) изменения из origin/serverfix.

* + 1. Отправка ветки на сервер под другим именем

|  |
| --- |
| Уровень освоения: Минимальный |

$ git push origin serverfix:serverfix

произойдёт то же самое — здесь говорится “возьми мой serverfix и сделай его удалённым serverfix.”

$ git push origin serverfix:awesomebranch.

Так ваша локальная ветка serverfix отправится в ветку awesomebranch удалённого проекта

* + 1. Переименование удаленного репозитория

$ git remote rename



* 1. 4. Удаление

|  |
| --- |
| Общие теги: Метод |

* + 1. Удаление веток с сервера



* + 1. Удаление удаленного репозитория

Если по какой-то причине вы хотите удалить ссылку (вы сменили сервер или больше не используете определённое зеркало, или, возможно, контрибьютор перестал быть активным), вы можете использовать

$ git remote rm



* 1. Удаленные ветки - что это

|  |
| --- |
| Общие теги: Применение |

Удалённые ветки — **это ссылки (pointers)** на состояние веток в ваших удалённых репозиториях. Это локальные ветки, которые нельзя перемещать; они двигаются автоматически всякий раз, когда вы осуществляете связь по сети.

Они выглядят как **(имя удал. репоз.)/(ветка)**

1. Р- Хостинг репозиториев. GITHUB
   1. Доступ по SSH

|  |
| --- |
| Общие теги: Скилл |

* + 1. Генерация открытого SSH Ключа

Генерируем ключ

$ ssh-keygen

Сначала необходимо указать расположение файла для сохранения ключа (.ssh/id\_rsa), затем дважды ввести пароль, который, впрочем, можно оставить пустым, если вы не хотите его вводить каждый раз, когда используете ключ.

Как прочитать ключ



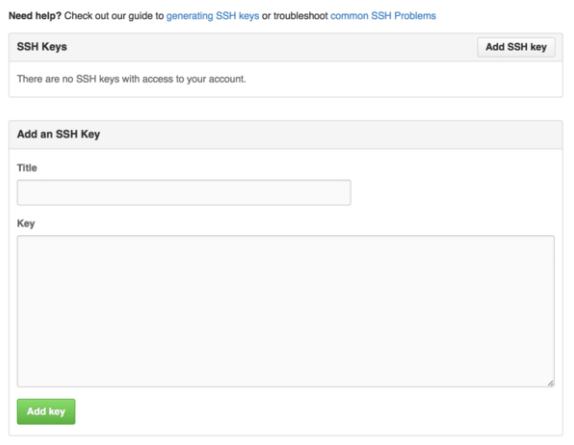
Переходим в папку .ssh

$ cd ~/.ssh

$ ls

* + 1. Зайти в настройки личного кабинета и внести ранее созданный ключ

Раздел SSH Keys -> Add key



Затем нажмите на кнопку "Добавить ключ SSH"("Add an SSH key"), задайте имя ключа а так же скопируйте и вставьте сам публичный ключ из ~/.ssh/id\_rsa.pub (ну или как бы у вас не назывался этот файл) в текстовое поле, затем нажмите “Добавить ключ”(“Add key”).

*Задавайте такое имя SSH ключа, которое вы в состоянии запомнить. Называйте каждый из добавляемых ключей по-разному (к примеру "Мой Ноутбук" или "Рабочая учётная запись"), для того чтобы в дальнейшем, при аннулировании ключа быть уверенным в правильности своего выбора.*

* 1. Создание аккаунта

|  |
| --- |
| Общие теги: Скилл |

Зарегистрироваться на сайта GitHub.ru

* 1. Создание репозитория на GITHUB

|  |
| --- |
| Общие теги: Метод |

1. Регистрируемся на github.com

2. Создаем репозиторий. При создании просим сразу создать в нём файл

README (это упростит нам дальнейшую работу)

3. Запоминаем адрес нашего репозитория (clone URL)

4. Клонируем этот репозиторий к себе на компьютер:

git clone CLONE\_URL PROJECT\_FOLDER

*В результате мы получаем у себя репозиторий, являющийся точной копией удалённого репозитория. Более того – они связаны невидимой связью и мы этим воспользуемся!*