**GIT ОСНОВНЫЕ КОМАНДЫ**

## Первоначальная настройка Git

## Можно создать удаленный репозиторий через сайт github это быстрее и проще. Далее создается папка с именем репозитория далее:

## git remote add origin <https://github.com/svmyhome/test99.git>

## Обзязательно что-нибудь создать и закомитить, а потом

## git push -u origin master

**sudo apt install curl** установка curl для возможности создания удаленного репозитория

**curl -u "svmyhome"** [**https://api.github.com**](https://api.github.com/)проверка можем ли мы законнектиться

**curl -u "svmyhome" https://api.github.com/user/repos -d'{"name":"Java"}'** создаем репозиторий с именем Java

**git remote add origin** [**https://github.com/svmyhome/Java.git**](https://github.com/svmyhome/Java.git)

**git push -u origin master**

**git config** утилита контролирует все аспекты работы Git'а и его внешний вид

* 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.
* Файл ~/.gitconfig хранит настройки конкретного пользователя. Этот файл используется при указании параметра --global.
* Конфигурационный файл в каталоге Git'а (.git/config) в том репозитории, где вы находитесь в данный момент. Эти параметры действуют только для данного конкретного репозитория. Настройки на каждом следующем уровне подменяют настройки из предыдущих уровней, то есть значения в .git/config перекрывают соответствующие значения в /etc/gitconfig.

Указываем имя и адрес электронной почты. каждый коммит в Git'е содержит эту информацию, не может быть далее изменена:

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

**git config --global user.email** [**johndoe@example.com**](mailto:johndoe@example.com)

**git config --global color.ui true** окрашивает в разные цвета

**git config --global core.quotepath off -** отключение представления имен файлов в кодировке Unicode

**--global**, настройку достаточно сделать только один раз, в этом случае Git будет использовать эти данные для всего, что вы делаете в этой системе. Если для каких-то отдельных проектов вы хотите указать другое имя или электронную почту, можно выполнить эту же команду без параметра --global в каталоге с нужным проектом.

**git config --global core.editor emacs** выброр текстового редактора в Git'е

**git config --global core.editor "code --wait"** замена стандартного редактора на VSCode

**git config --global core.editor "subl -w"** на sublime

**git config --global merge.tool vimdiff** встроенная diff-утилита, которая будет использоваться для разрешения конфликтов слияния

**git config --list** проверка используемых настроек

**git config user.name** проверка значения конкретного ключа

**git help** варианты помощи

**git help <команда>**

**git <команда> --help**

**man git-<команда>**

**git help config**

## Создание Git-репозитория

Два основных подхода. Первый подход — импорт в Git уже существующего проекта или каталога. Второй — клонирование уже существующего репозитория с сервера.

**Первый подход** - начать использовать Git для существующего проекта, необходимо перейти в проектный каталог и в командной строке ввести

**git init** создаёт в текущем каталоге новый подкаталог с именем .git содержащий все необходимые файлы репозитория — основу Git-репозитория.

добавление под версионный контроль существующих файлов, а затем commit

**git add \***

**git add \*.c**

**git add README**

**git commit -m 'initial project version'**

**Второй** копию существующего репозитория Git, например, проекта, в котором вы хотите поучаствовать, то вам нужна команда

**git clone git://github.com/schacon/grit.git** создаёт каталог с именем grit, инициализирует в нём каталог .git, скачивает все данные.

**git clone git://github.com/schacon/grit.git mygrit** делает всё то же самое, результирующий каталог будет назван mygrit

**git status** определение состояния фалов (не отслеживаемый, под версионным контролем, или нет)

Для игнорирования файлов создать файл .gitignore

**touch .gitignore** создание файла

**cat .gitignore**  показывает содержимое файла

**\*.[oa]** игнорировать любые файлы заканчивающиеся на .o или .a

**\*~** игнорировать все файлы заканчивающиеся на тильду (~), которая используется текстовых редакторах, для обозначения временных файлов

Вы можете также включить каталоги log, tmp или pid

**git checkout --track origin/site-cbr** получение бранча из удаленного репозитория git clone создает копию удаленного (в общем случае) репозитория. Чтобы получить бранч из удаленного репозитория нужно после git clone выполнить:

### Просмотр индексированных и неиндексированных изменений

**git diff** покажет что изменилось, но пока не проиндексировано более информативна

**git diff --cached и git diff --staged** (**staged** — новое назвние для cached)показывают что проиндексировано и что войдёт в следующий коммит

**git diff —color-words** показывает изменеия на одной стоке(более удобно)

**- shift s Enter** показывает текст в сокращенном или полном виде

### Фиксация изменений

**git add \*** индексировать все файлы

**git add -i** интерактивное индексирование, появляется меню выбора

**git commit** откроет выбранный вами текстовый редактор

**git commit -v**  в комментарий будет также помещена дельта/diff изменений

**git commit -m 'initial project version' или**

**git commit -v -m 'initial project version'** после параметра -m указывается комментарий

**git commit -a -v -m 'added new benchmarks'** автоматическое индексирование , позволяя вам обойтись без git add

**git commit --amend -m "сообщение"** закоммитили изменения, позже поняли, что допустили ошибку, или просто необходимо сделать описание более понятным

**git commit --amend --no-edit** вы добавили новые файлы или исправили ошибку, но не хотите добавлять еще одно commit-сообщение

**git push origin <имя ветки> -f** запушили свой код в удаленный репозиторий, а затем поняли, что нужно изменить commit-сообщение. Для этого после внесения изменений сделайте принудительный push

### Удаление файлов

**git rm 1.txt** удаление файла из индекса

**git rm -f 1.txt** если в проиндексированном файле есть непроиндексированные изменения то удаляем с ключем -f

**git rm --cached readme.txt** удалить файл из индекса, оставив его при этом в рабочем каталоге

**git mv file\_from file\_to** переименовать файл

## Просмотр истории коммитов

**git log** история коммитов, все изменения

**git log -p -2** -p показывает дельту (разницу/diff), привнесенную каждым коммитом, -2 что ограничит вывод до 2-х последних записей

**git log -U1 --word-diff** дельта по словам вместо обычной дельты по строкам (работе над длинным текстом, таким как книга или диссертация)

**git log --stat** получить некоторую краткую статистику по каждому коммиту

**git log --pretty=oneline** -oneline выводит каждый коммит в одну строку (oneline, short, full, fuller и format)

**git log --graph** добавляет ASCII-граф, показывающий историю ветвлений и слияний

**git log --since=2 019-01-10** показывает лог начиная с 10.01.19

**git log** **--author= “Kevin“** показывает лог по автору

**git log --grep=“Kevin“ т**екст который хотим найти во всем гит

### [Ограничение вывода команды log](https://git-scm.com/book/ru/v1/Основы-Git-Просмотр-истории-коммитов" \l "Ограничение-вывода-команды-log)

**git log --since=2.weeks** команда выдаёт список коммитов, сделанных за последние две недели

| Опция | Описание |
| --- | --- |
| -(n) | Показать последние n коммитов |
| --since, --after | Ограничить коммиты теми, которые сделаны после указанной даты. |
| --until, --before | Ограничить коммиты теми, которые сделаны до указанной даты. |
| --author | Показать только те коммиты, автор которых соответствует указанной строке.--author="John Doe" |
| --committer | Показать только те коммиты, коммитер которых соответствует указанной строке. |

**gitk** графический инструмент для визуализации истории коммитов

## Отмена изменений

**git commit --amend** изменить комментарий к последнему коммиту

**git reset HEAD Example.java** вывод файла из индексации

**git checkout** **-- Example.java** отменить изменения, вернуть то состояние, в котором он находился во время последнего коммита

**git checkout 0ce7a059997639 -- index.html** возращает на редактирование конкретный файл из конкретного коммита

**git revert 0ce7a059997639e88157c3** отменяет предыдущий коммит путем отмены всех изменений и созданием нового коммита, с соответсвующей записью

**git revert HEAD —-no-edit** удаляет предыдущий коммит

**git revert ff3423615(хэш) --no-commit** удаляет коммит с определенным хешем

**git reset --mixed 96b037c** git reset —soft «потеряет» все коммиты после этого идентификатора (например, 96b037c), но файлы не будут удалены, они будут находиться в промежуточной области -**– hard**

## Работа с удалёнными репозиториями

1. Для публикации в удаленном репозитарии можно использовать следующую команду:
2. $ git push origin dev
3. Total 0 (delta 0), reused 0 (delta 0)
4. To git@github.com:devlabuser/sharp.git
5. \* [new branch] dev -> dev
6. **Однако,** такая команда не сделает ветку dev на удаленном репозитарии отслеживаемой и связанной с локальной. Поэтому, лучше всего воспользоваться дополнительной опцией --set-upstream:
7. $ git push --set-upstream origin dev
8. Или, что то же самое, но с сокращенным именем опции:
9. $ git push -u origin dev
10. После этой команды локальная ветка dev и ветка dev в репозитарии будут связаны между собой. То есть, после такой команды (которая выполняется единожды), можно будет отправлять/принимать изменения лишь выполняя git push из ветки без указания всяких алиасов для сервера и удалённых веток. Это сделано просто для удобства.

**git remote** какие удалённые серверы у вас настроены

**git remote -v** какому URL соответствует сокращённое имя в Git

**git remote add pb git://github.com/paulboone/ticgit.git** добавление нового удалённого Git-репозиторий под именем-сокращением **pb**

**git fetch pb** извлечь (fetch) всю информацию, которая есть в репозитории pb, но нет в вашем

**git pull** автоматически извлекает и затем сливает данные из удалённой ветки в текущую ветку

**git push origin master** отправить ветку master на сервер origin

**git push origin serverfix:awesomebranch** локальная ветка serverfix отправится в ветку awesomebranch удалённого проекта.

**git remote show origin** получить побольше информации об одном из удалённых репозиториев

**git remote rename pb paul**  изменит сокращённое имя, используемое для удалённого репозитория

**git remote rm paul** удалить ссылку

## Работа с метками

**git tag** Просмотр имеющихся меток

**git tag -l 'v1.4.2.\*'** поиск по шаблону только выпусков 1.4.2

**git tag -a v1.4 -m 'my version 1.4'** -a Создание аннотированной метки -m задаёт меточное сообщение, которое будет храниться вместе с меткой. Если не указать сообщение для аннотированной метки, Git запустит редактор, чтоб вы смогли его ввести.

**git show v1.4** посмотреть описание метки вместе с коммитом

**git tag -a v1.2 -m 'version 1.2' 9fceb02** Для добавления метки позже для отметки коммита укажите его контрольную сумму (или её часть сначала числа) в конце команды

**git push origin v1.5** Необходимо явно отправить (push) метки на общий сервер после того, как вы их создали

**git push origin --tags**  все ваши метки отправятся на удалённый сервер

# **Ветвление в Git**

**git branch testing** создадим новую ветку под названием testing

**git branch -m <новое имя ветки>** нужно изменить имя текущей ветки:

**git branch -m <старое имя ветки> <новое имя ветки>** Вы сделали опечатку в имени ветки или хотите изменить ее имя

**git push <имя удаленной ветки> <локальное имя ветки>:<имя удаленной ветки>** имя локальной ветки не совпадает с именем ветки в репозитории

**git checkout testing** переход на существующую ветку testing

**git checkout -b testing** создние ветки и сразу же переход на неё

**git merge testing** сливает текущую ветку и ветку testing (работает если нет конфликтов)

**git branch -d testing** удаление после того как слили ветку

**git branch -v** показывает последний коммит

**git branch --merged** те ветки, которые уже слиты с текущей

**git branch --no-merged** те ветки, которые еще не слиты с текущей

**git fetch origin** команда ищет, какому серверу соответствует origin, извлекает оттуда все данные, которых у вас ещё нет, и обновляет ваше локальное хранилище данных; сдвигает указатель origin/master на новую позицию

**git merge origin/master** сливает ранее скаченную ветку мастер с твоей текущей веткой мастер

**git checkout feature** выполняются вместе, перключаемся на ветку feature и сливаем

**git rebase release** ветку release в feature тем самым выстраиваем последовательно коммиты из веток (Запомните, rebase − только для локального репозитория)

## Перемещение

## Прятанье

**git stash** поменять ветку, но не хотите делать коммит с тем, над чем вы ещё работаете; тогда вы прячете эти изменения

**git stash list** показывает то, что спрятано

**git stash apply** применить последнюю спрятанную

**git stash apply stash@{2}** применить одну из старых заначек

**git stash drop stash@{0}** удаляет спрятанную заначку

**git stash pop** применить спрятанные изменения и удалить

PULL request https://habr.com/post/125999/