## **Создание Git-репозитория**

Для создания Git-репозитория вы можете использовать два основных подхода. Во-первых, импорт в Git уже существующего проекта или директории. Во-вторых, клонирование существующего репозитория с другого сервера.

### Создание репозитория в существующей директории

Если вы собираетесь начать использовать Git для существующего проекта, то вам необходимо перейти в директорию проекта и в командной строке ввести

$ git init

Эта команда создаёт в текущей директории новую поддиректорию с именем .git, содержащую все необходимые файлы репозитория — основу Git-репозитория.

**Клонирование существующего репозитория**

Для получения копии существующего Git-репозитория, например, проекта, в который вы хотите внести свой вклад, необходимо использовать команду **git clone**

Клонирование репозитория осуществляется командой **git clone [url]**. Например, если вы хотите клонировать библиотеку libgit2, вы можете сделать это следующим образом:

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

### Определение состояния файлов

Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда **git status**. Если вы выполните эту команду сразу после клонирования, вы увидите что-то вроде этого:

$ git status

On branch master

nothing to commit, working directory clean

### Отслеживание новых файлов

Для того чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда **git add**. Чтобы начать отслеживание файла README, вы можете выполнить следующее:

$ git add README

Если вы снова выполните команду **status**, то увидите, что файл README теперь отслеживаемый и индексированный:

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

Вы можете видеть, что файл проиндексирован по тому, что он находится в секции ``Changes to be committed``. Если вы выполните коммит в этот момент, то версия файла, существовавшая на момент выполнения вами команды **git add**, будет добавлена в историю снимков состояния.

### Индексация изменённых файлов

Давайте модифицируем файл, уже находящийся под версионным контролем. Если вы измените отслеживаемый файл ``CONTRIBUTING.md`` и после этого снова выполните команду **git status**, то результат будет примерно следующим:

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: CONTRIBUTING.md

Файл ``CONTRIBUTING.md`` находится в секции ``Changes not staged for commit`` — это означает, что отслеживаемый файл был изменён в рабочем каталоге, но пока не проиндексирован. Чтобы проиндексировать его, необходимо выполнить команду **git add**. Это многофункциональная команда, она используется для добавления под версионный контроль новых файлов, для индексации изменений, а также для других целей, например для указания файлов с исправленным конфликтом слияния. Вам может быть понятнее, если вы будете думать об этом как ``добавить этот контент в следующий коммит``, а не как ``добавить этот файл в проект``. Выполним **git add**, чтобы проиндексировать ``CONTRIBUTING.md``, а затем снова выполним **git status**:

$ git add CONTRIBUTING.md

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

modified: CONTRIBUTING.md

Теперь оба файла проиндексированы и войдут в следующий коммит.

### Коммит изменений

Теперь, когда ваш индекс находится в таком состоянии, как вам и хотелось, вы можете зафиксировать свои изменения. Запомните, всё, что до сих пор не проиндексировано — любые файлы, созданные или изменённые вами, и для которых вы не выполнили **git add** после момента редактирования — не войдут в этот коммит. Они останутся изменёнными файлами на вашем диске. В нашем случае, когда вы в последний раз выполняли **git status**, вы видели что всё проиндексировано, и вот, вы готовы к коммиту. Простейший способ зафиксировать изменения — это набрать **git commit**:

$ git commit

Эта команда откроет выбранный вами текстовый редактор. (Редактор устанавливается системной переменной $EDITOR — обычно это vim или emacs, хотя вы можете установить ваш любимый с помощью команды **git config --global core.editor**

В редакторе будет отображён следующий текст (это пример окна Vim’а):

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

# On branch master

# Changes to be committed:

# new file: README

# modified: CONTRIBUTING.md

#

~

~

~

".git/COMMIT\_EDITMSG" 9L, 283C

Вы можете видеть, что комментарий по умолчанию для коммита содержит закомментированный результат работы ("выхлоп") команды **git status** и ещё одну пустую строку сверху. Вы можете удалить эти комментарии и набрать своё сообщение или же оставить их для напоминания о том, что вы фиксируете. (Для ещё более подробного напоминания, что же именно вы поменяли, можете передать аргумент **-v** в команду **git commit**. Это приведёт к тому, что в комментарий будет также помещена дельта/diff изменений, таким образом вы сможете точно увидеть все изменения которые вы совершили.) Когда вы выходите из редактора, Git создаёт для вас коммит с этим сообщением (удаляя комментарии и вывод diff’а).

Есть и другой способ — вы можете набрать свой комментарий к коммиту в командной строке вместе с командой **commit** указав его после параметра **-m**, как в следующем примере:

$ git commit -m "Story 182: Fix benchmarks for speed"

[master 463dc4f] Story 182: Fix benchmarks for speed

2 files changed, 2 insertions(+)

create mode 100644 README

Итак, вы создали свой первый коммит! Вы можете видеть, что коммит вывел вам немного информации о себе: на какую ветку вы выполнили коммит **(master)**, какая контрольная сумма SHA-1 у этого коммита (463dc4f), сколько файлов было изменено, а также статистику по добавленным/удалённым строкам в этом коммите.

Запомните, что коммит сохраняет снимок состояния вашего индекса. Всё, что вы не проиндексировали, так и висит в рабочем каталоге как изменённое; вы можете сделать ещё один коммит, чтобы добавить эти изменения в репозиторий. Каждый раз, когда вы делаете коммит, вы сохраняете снимок состояния вашего проекта, который позже вы можете восстановить или с которым можно сравнить текущее состояние.

### Игнорирование индексации

Несмотря на то, что индекс может быть удивительно полезным для создания коммитов именно такими, как вам и хотелось, он временами несколько сложнее, чем вам нужно в процессе работы. Если у вас есть желание пропустить этап индексирования, Git предоставляет простой способ. Добавление параметра **-a** в команду **git commit** заставляет Git автоматически индексировать каждый уже отслеживаемый на момент коммита файл, позволяя вам обойтись без **git add**:

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: CONTRIBUTING.md

no changes added to commit (use "git add" and/or "git commit -a")

$ git commit -a -m 'added new benchmarks'

[master 83e38c7] added new benchmarks

1 file changed, 5 insertions(+), 0 deletions(-)

Обратите внимание на то, что в данном случае перед коммитом вам не нужно выполнять **git add** для файла ``CONTRIBUTING.md``.

### Удаление файлов

Для того чтобы удалить файл из Git, вам необходимо удалить его из отслеживаемых файлов (точнее, удалить его из вашего индекса) а затем выполнить коммит. Это позволяет сделать команда **git rm**, которая также удаляет файл из вашего рабочего каталога, так что вы в следующий раз не увидите его как “неотслеживаемый”.

Если вы просто удалите файл из своего рабочего каталога, он будет показан в секции ``Changes not staged for commit`` (измененные, но не проиндексированные) вывода команды **git status**:

$ rm PROJECTS.md

$ git status

On branch master

Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

deleted: PROJECTS.md

no changes added to commit (use "git add" and/or "git commit -a")

Затем, если вы выполните команду **git rm**, удаление файла попадёт в индекс:

$ git rm PROJECTS.md

rm 'PROJECTS.md'

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

deleted: PROJECTS.md

После следующего коммита файл исчезнет и больше не будет отслеживаться. Если вы изменили файл и уже проиндексировали его, вы должны использовать принудительное удаление с помощью параметра **-f**. Это сделано для повышения безопасности, чтобы предотвратить ошибочное удаление данных, которые ещё не были записаны в снимок состояния и которые нельзя восстановить из Git.

Другая полезная штука, которую вы можете захотеть сделать — это удалить файл из индекса, оставив его при этом в рабочем каталоге. Другими словами, вы можете захотеть оставить файл на жёстком диске, и убрать его из-под бдительного ока Git. Это особенно полезно, если вы забыли добавить что-то в файл .gitignore и по ошибке проиндексировали, например, большой файл с логами, или кучу промежуточных файлов компиляции. Чтобы сделать это, используйте опцию **--cached**:

$ git rm --cached README

В команду **git rm** можно передавать файлы, каталоги или glob-шаблоны. Это означает, что вы можете вытворять что-то вроде:

$ git rm log/\\*.log

Обратите внимание на обратный слэш (\) перед \*. Он необходим из-за того, что Git использует свой собственный обработчик имён файлов вдобавок к обработчику вашего командного интерпретатора. Эта команда удаляет все файлы имеющие расширение .log находящиеся в директории log/. Или же вы можете сделать вот так:

$ git rm \\*~

Эта команда удаляет все файлы, чьи имена заканчиваются на ~.

### Перемещение файлов

В отличие от многих других систем версионного контроля, Git не отслеживает перемещение файлов явно. Когда вы переименовываете файл в Git, в нём не сохраняется никаких метаданных, говорящих о том, что файл был переименован. Однако, Git довольно умён в плане обнаружения перемещений постфактум — мы рассмотрим обнаружение перемещения файлов чуть позже.

Таким образом, наличие в Git команды **mv** выглядит несколько странным. Если вам хочется переименовать файл в Git, вы можете сделать что-то вроде:

$ git mv file\_from file\_to

и это отлично сработает. На самом деле, если вы выполните что-то вроде этого и посмотрите на статус, вы увидите, что Git считает, что произошло переименование файла:

$ git mv README.md README

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

renamed: README.md -> README

Однако, это эквивалентно выполнению следующих команд:

$ mv README.md README

$ git rm README.md

$ git add README

Git неявно определяет, что произошло переименование, поэтому неважно, переименуете вы файл так или используя команду **mv**. Единственное отличие состоит лишь в том, что **mv** — это одна команда вместо трёх — это функция для удобства. Важнее другое — вы можете использовать любой удобный способ, чтобы переименовать файл, и затем воспользоваться add/rm перед коммитом.

## **Просмотр истории коммитов**

После того, как вы создали несколько коммитов или же склонировали репозиторий с уже существующей историей коммитов, вероятно вам понадобится возможность посмотреть что было сделано – историю коммитов. Одним из основных и наиболее мощных инструментов для этого является команда **git log**.

Следующие несколько примеров используют очень простой проект “simplegit”. Чтобы склонировать проект, используйте команду:

git clone https://github.com/schacon/simplegit-progit

Если вы запустите команду **git log** в папке склонированного проекта, вы увидите следующий вывод:

$ git log

commit ca82a6dff817ec66f44342007202690a93763949

Author: Scott Chacon <schacon@gee-mail.com>

Date: Mon Mar 17 21:52:11 2008 -0700

changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7

Author: Scott Chacon <schacon@gee-mail.com>

Date: Sat Mar 15 16:40:33 2008 -0700

removed unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6

Author: Scott Chacon <schacon@gee-mail.com>

Date: Sat Mar 15 10:31:28 2008 -0700

first commit

По умолчанию (без аргументов) **git log** перечисляет коммиты, сделанные в репозитории в обратном к хронологическому порядке – последние коммиты находятся вверху. Из примера можно увидеть, что данная команда перечисляет коммиты с их SHA-1 контрольными суммами, именем и электронной почтой автора, датой создания и сообщением коммита.

Команда **git log** имеет очень большое количество опций для поиска коммитов по разным критериям.

## **Операции отмены**

В любой момент вам может потребоваться что-либо отменить. Здесь мы рассмотрим несколько основных способов отмены сделанных изменений. Будьте осторожны, не все операции отмены в свою очередь можно отменить! Это одна из редких областей Git’а, где неверными действиями можно необратимо удалить результаты своей работы.

Отмена может потребоваться, если вы сделали коммит слишком рано, например, забыв добавить какие-то файлы или комментарий к коммиту. Если вы хотите переделать коммит, можно запустить commit с параметром **--amend** (дополнить):

$ git commit --amend

Эта команда использует для дополнения коммита вашу область подготовки (индекс). Если вы ничего не меняли с момента последнего коммита (например, команда запущена сразу после предыдущего коммита), то снимок состояния останется в точности таким же, а изменится лишь комментарий к коммиту.

Запустится тот же редактор комментария к коммиту, но уже с комментарием к предыдущему коммиту. Комментарий можно отредактировать точно так же, как обычно, просто он заменит собой предыдущий.

Например, если вы фиксируете изменения, и понимаете, что забыли проиндексировать изменения в файле, который хотели включить в коммит, можно сделать примерно так:

$ git commit -m 'initial commit'

$ git add forgotten\_file

$ git commit --amend

В итоге получится единый коммит — второй коммит заменит результаты первого.

### Отмена подготовки файла

В следующих двух разделах показано, как разбираться с изменениями вашей области подготовки (staging area) и рабочего каталога. Радует, что команда, которой вы определяете состояние этих областей, также напоминает вам, как отменять их изменения. Например, скажем, вы изменили два файла, и хотите закоммитить их двумя раздельными изменениями, но случайно набрали **git add \***, и добавили оба в индекс. Как отменить добавление одного из них? Команда **git status** напомнит вам:

$ git add .

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

renamed: README.md -> README

modified: CONTRIBUTING.md

Прямо под текстом “Changes to be committed” говорится: **git reset HEAD <file>**... для отмены добавления в индекс. Давайте последуем этому совету, и отменим индексирование файла CONTRIBUTING.md:

$ git reset HEAD CONTRIBUTING.md

Unstaged changes after reset:

M CONTRIBUTING.md

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

renamed: README.md -> README

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: CONTRIBUTING.md

Команда выглядит несколько странно, но — работает! Файл CONTRIBUTING.md изменен, но снова не добавлен в область подготовки к коммиту.

Хотя **git reset** может стать опасной командой, если ее вызвать с **--hard**, в приведенном примере файл в вашем рабочем каталоге не затрагивается. Вызов **git reset** без параметра не опасен — он затрагивает только область подготовки.

## **Работа с удалёнными репозиториями**

Для того, чтобы внести вклад в какой-либо Git-проект, вам необходимо уметь работать с удалёнными репозиториями. Удалённые репозитории представляют собой версии вашего проекта, сохранённые в интернете или ещё где-то в сети. У вас может быть несколько удалённых репозиториев, каждый из которых может быть доступен для чтения или для чтения-записи. Взаимодействие с другими пользователями предполагает управление удалёнными репозиториями, а также отправку и получение данных из них. Управление репозиториями включает в себя как умение добавлять новые, так и умение удалять устаревшие репозитории, а также умение управлять различными удалёнными ветками, объявлять их отслеживаемыми или нет и так далее. В данном разделе мы рассмотрим некоторые из этих навыков.

### Просмотр удалённых репозиториев

Для того, чтобы просмотреть список настроенных удалённых репозиториев, вы можете запустить команду **git remote**. Она выведет названия доступных удалённых репозиториев. Если вы клонировали репозиторий, то увидите как минимум ``origin`` — имя по умолчанию для исходного репозитория:

$ git clone https://github.com/schacon/ticgit

Cloning into 'ticgit'...

remote: Reusing existing pack: 1857, done.

remote: Total 1857 (delta 0), reused 0 (delta 0)

Receiving objects: 100% (1857/1857), 374.35 KiB | 268.00 KiB/s, done.

Resolving deltas: 100% (772/772), done.

Checking connectivity... done.

$ cd ticgit

$ git remote

origin

Вы можете также указать ключ **-v**, чтобы просмотреть адреса для чтения и записи, привязанные к репозиторию:

$ git remote -v

origin https://github.com/schacon/ticgit (fetch)

origin https://github.com/schacon/ticgit (push)

Если у вас больше одного удалённого репозитория, команда выведет их все. Например, для репозитория с несколькими настроенными удалёнными репозиториями в случае совместной работы нескольких пользователей, вывод команды может выглядеть примерно так:

$ cd grit

$ git remote -v

bakkdoor https://github.com/bakkdoor/grit (fetch)

bakkdoor https://github.com/bakkdoor/grit (push)

cho45 https://github.com/cho45/grit (fetch)

cho45 https://github.com/cho45/grit (push)

defunkt https://github.com/defunkt/grit (fetch)

defunkt https://github.com/defunkt/grit (push)

koke git://github.com/koke/grit.git (fetch)

koke git://github.com/koke/grit.git (push)

origin git@github.com:mojombo/grit.git (fetch)

origin git@github.com:mojombo/grit.git (push)

Это означает, что мы можем легко получить изменения от любого из этих пользователей. Возможно, что некоторые из репозиториев доступны для записи, и в них можно отправлять свои изменения, хотя вывод команды не даёт никакой информации о правах доступа.

Обратите внимание на разнообразие протоколов, используемых при указании адреса удалённого репозитория.

### Добавление удалённых репозиториев

В предыдущих разделах мы уже упоминали и приводили примеры добавления удалённых репозиториев, сейчас рассмотрим эту операцию подробнее. Для того, чтобы добавить удалённый репозиторий и присвоить ему имя (shortname), просто выполните команду **git remote add [shortname] [url]**:

$ git remote

origin

$ git remote add pb https://github.com/paulboone/ticgit

$ git remote -v

origin https://github.com/schacon/ticgit (fetch)

origin https://github.com/schacon/ticgit (push)

pb https://github.com/paulboone/ticgit (fetch)

pb https://github.com/paulboone/ticgit (push)

Теперь вместо указания полного пути вы можете использовать **pb**. Например, если вы хотите получить изменения, которые есть у Пола, но нету у вас, вы можете выполнить команду **git fetch pb**:

$ git fetch pb

remote: Counting objects: 43, done.

remote: Compressing objects: 100% (36/36), done.

remote: Total 43 (delta 10), reused 31 (delta 5)

Unpacking objects: 100% (43/43), done.

From https://github.com/paulboone/ticgit

\* [new branch] master -> pb/master

\* [new branch] ticgit -> pb/ticgit

Ветка **master** из репозитория Пола сейчас доступна вам под именем **pb/master**. Вы можете слить её с одной из ваших веток или переключить на неё локальную ветку, чтобы просмотреть содержимое ветки Пола.

### Получение изменений из удалённого репозитория - Fetch и Pull

Как вы только что узнали, для получения данных из удалённых проектов, следует выполнить:

$ git fetch [remote-name]

Данная команда связывается с указанным удалённым проектом и забирает все те данные проекта, которых у вас ещё нет. После того как вы выполнили команду, у вас должны появиться ссылки на все ветки из этого удалённого проекта. Теперь эти ветки в любой момент могут быть просмотрены или слиты.

Когда вы клонируете репозиторий, команда clone автоматически добавляет этот удалённый репозиторий под именем “origin”. Таким образом, **git fetch origin** извлекает все наработки, отправленные (push) на этот сервер после того, как вы склонировали его (или получили изменения с помощью fetch). Важно отметить, что команда **git fetch** забирает данные в ваш локальный репозиторий, но не сливает их с какими-либо вашими наработками и не модифицирует то, над чем вы работаете в данный момент. Вам необходимо вручную слить эти данные с вашими, когда вы будете готовы.

Если у вас есть ветка, настроенная на отслеживание удалённой ветки (обратитесь к главе 3 [Ветвление в Git](https://git-scm.com/book/ru/v2/ch00/ch03-git-branching) за более подробной информацией), то вы можете использовать команду **git pull** чтобы автоматически получить изменения из удалённой ветви и слить их со своей текущей ветвью. Этот способ может для вас оказаться более простым или более удобным. К тому же по умолчанию команда **git clone** автоматически настраивает вашу локальную ветку master на отслеживание удалённой ветки master на сервере, с которого вы клонировали (подразумевается, что на удалённом сервере есть ветка master). 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.

### Отправка изменений в удаленный репозиторий (Push)

Когда вы хотите поделиться своими наработками, вам необходимо отправить (push) их в главный репозиторий. Команда для этого действия простая: **git push [remote-name] [branch-name]**. Чтобы отправить вашу ветку master на сервер **origin** (повторимся, что клонирование, как правило, настраивает оба этих имени автоматически), вы можете выполнить следующую команду для отправки наработок на сервер:

$ git push origin master

Эта команда срабатывает только в случае, если вы клонировали с сервера, на котором у вас есть права на запись, и если никто другой с тех пор не выполнял команду push. Если вы и кто-то ещё одновременно клонируете, затем он выполняет команду push, а затем команду push выполняете вы, то ваш push точно будет отклонён. Вам придётся сначала вытянуть (pull) их изменения и объединить с вашими. Только после этого вам будет позволено выполнить push. См. главу 3 [Ветвление в Git](https://git-scm.com/book/ru/v2/ch00/ch03-git-branching) для более подробного описания, как отправлять (push) данные на удалённый сервер.

### Просмотр удаленного репозитория

Если хотите получить побольше информации об одном из удалённых репозиториев, вы можете использовать команду **git remote show [remote-name]**. Выполнив эту команду с некоторым именем, например, **origin**, вы получите результат, аналогичный следующему:

$ git remote show origin

\* remote origin

Fetch URL: https://github.com/schacon/ticgit

Push URL: https://github.com/schacon/ticgit

HEAD branch: master

Remote branches:

master tracked

dev-branch tracked

Local branch configured for 'git pull':

master merges with remote master

Local ref configured for 'git push':

master pushes to master (up to date)

Она выдаёт URL удалённого репозитория, а также информацию об отслеживаемых ветках. Эта команда любезно сообщает вам, что если вы, находясь на ветке master, выполните **git pull**, ветка master с удалённого сервера будет автоматически влита в вашу сразу после получения всех необходимых данных. Она также выдаёт список всех полученных ею ссылок.

Это был пример для простой ситуации, и наверняка вы встретились с чем-то подобным. Однако, если вы используете Git более интенсивно, вы можете увидеть гораздо большее количество информации от **git remote show**:

$ git remote show origin

\* remote origin

URL: https://github.com/my-org/complex-project

Fetch URL: https://github.com/my-org/complex-project

Push URL: https://github.com/my-org/complex-project

HEAD branch: master

Remote branches:

master tracked

dev-branch tracked

markdown-strip tracked

issue-43 new (next fetch will store in remotes/origin)

issue-45 new (next fetch will store in remotes/origin)

refs/remotes/origin/issue-11 stale (use 'git remote prune' to remove)

Local branches configured for 'git pull':

dev-branch merges with remote dev-branch

master merges with remote master

Local refs configured for 'git push':

dev-branch pushes to dev-branch (up to date)

markdown-strip pushes to markdown-strip (up to date)

master pushes to master (up to date)

Данная команда показывает какая именно локальная ветка будет отправлена на удалённый сервер по умолчанию при выполнении **git push**. Она также показывает, каких веток с удалённого сервера у вас ещё нет, какие ветки всё ещё есть у вас, но уже удалены на сервере. И для нескольких веток показано, какие удалённые ветки будут в них влиты при выполнении **git pull**.

### Удаление и переименование удалённых репозиториев

Для переименования ссылок в новых версиях Git’а можно вылолнить **git remote rename**, это изменит сокращённое имя, используемое для удалённого репозитория. Например, если вы хотите переименовать **pb** в **paul**, вы можете это сделать при помощи **git remote rename**:

$ git remote rename pb paul

$ git remote

origin

paul

Стоит упомянуть, что это также меняет для вас имена удалённых веток. То, к чему вы обращались как **pb/master**, теперь стало **paul/master**.

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**:

$ git remote rm paul

$ git remote

origin

Более подробно о работе с git можно почитать тут: <https://git-scm.com/book/ru/v2>