**Git для начинающих. Часть 3. Настройка Git**

Автор: [Marat Abdrakhmanov](https://devpractice.ru/author/writer/) | 04.04.2017

[Нет комментариев](https://devpractice.ru/git-for-beginners-part-3-config-git/#respond)

Статья посвещена вопросам настройки *Git* в операционных системах *Windows* и *Linux*. Рассмотрены вопросы настройки на системном, пользовательском уровне и уровне репозитория.

Настройка системы *Git* предполагает, в первую очередь, указание имени пользователя и *e-mail*, которые используются для подписи коммитов и отправки изменений в удаленный репозиторий.

В *Git* существует три места, где хранятся настройки:

* **на уровне системы;**
* **на уровне пользователя;**
* **на уровне проекта (репозитория).**

Для того, чтобы сконфигурировать *Git* на том или ином уровне вы можете изменить непосредственно конфигурационные файлы, но для этого нужно знать их формат, либо воспользоваться специальными командами, которые предоставляет *Git*. Мы рекомендуем использовать команды.

**Расположение конфигурационных файлов *Git***

***Windows***

Уровень системы

**\Program Files\Git\mingw64\etc\gitconfig**

Имейте ввиду, что для его изменения вам могут понадобиться права администратора!

Уровень пользователя

**%HOMEPATH%\.gitconfig**

Уровень репозитория

**папка\_с\_проектом\.git\config**

***Linux***

Уровень системы

**/etc/gitconfig**

Уровень пользователя

**~/.gitconfig**

Уровень репозитория

**папка\_с\_проектом/.git/config**

**Конфигурирование *Git* с помощью утилиты командной строки**

Как уже было сказано выше, конфигурирование *Git* с помощью утилиты командной строки – это наиболее удобный и безопасный способ. Независимо от того, на каком уровне вы хотите менять настройки, команда будет начинаться так:

***git config***

Для уровня системы, мы должны написать:

> git config --system

уровня пользователя:

> git config --global

уровня приложения:

> git config

После этой команды указывается параметр и его значение.

Например, зададим имя и *e-mail* разработчика для уровня пользователя.

> git config --global user.name "User"

> git config --global user.email "user@company.com"

Для просмотра введенных изменений воспользуйтесь командой:

> git config --list

Дополнительно вы можете указать текстовый редактор, который будет запускать *Git*, если ему потребуется получить от вас какие-то данные, для этого модифицируйте параметр *core.editor*:

* вариант для *Linux*:
* > git config --global core.editor "nano"
* вариант для *Windows*:
* > git config --global core.editor "notepad.exe"

**Git для начинающих. Часть 5. Создание репозитория и первый коммит**

Автор: [Marat Abdrakhmanov](https://devpractice.ru/author/writer/) | 19.02.2018

[4 комментария](https://devpractice.ru/git-for-beginners-part-5-create-repo-and-commit/#comments)

В этом уроке мы рассмотрим, как создать пустой *git* репозиторий, добавить в него файлы и сделать первый коммит. Также коснемся вопроса просмотра коммитов и состояния рабочего каталога.

* [Создание репозитория *git*](https://devpractice.ru/git-for-beginners-part-5-create-repo-and-commit/#p1)
* [Создание первого коммита](https://devpractice.ru/git-for-beginners-part-5-create-repo-and-commit/#p2)

Создание репозитория

Для того чтобы создать репозиторий, для начала, создайте папку, в которой он будет располагаться. В нашем случае это будет каталог с названием *repo*.

> mkdir repo

Теперь перейдем в этот каталог.

>cd repo

Создадим в нем пустой *git* репозиторий.

> git init

Создание первого коммита

Если мы посмотрим на список коммитов, которые были отправлены в репозиторий, то увидим, что он пустой – это правильно, т.к. мы пока только создали репозиторий и ничего ещё туда не отправляли.

> git log

fatal: your current branch 'master' does not have any commits yet

Для просмотра состояния рабочего каталога воспользуемся командой *git status*.

> git status

On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)

Создадим в нашем каталоге пустой файл.

> touch README.md

Теперь, если мы выполним команду *git status*, то увидим, что в нашем каталоге появился один неотслеживаемый файл: *README.md*.

> git status

On branch master

Initial commit

Untracked files:

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

README.md

nothing added to commit but untracked files present (use "git add" to track)

Добавим, созданный файл в *stage*. *Stage* (или *cache*) – это хранилище для файлов с изменениями, информация о которых попадет в единый коммит. *Stage* является элементом архитектуры трех деревьев, на базе которой построен *git*, более подробно смотрите [здесь](https://devpractice.ru/git-for-beginners-part-4-git-arch/). Для добавления файла *README.md* в *stage* необходимо воспользоваться командой *git add*.

> git add README.md

Если изменение было произведено в нескольких файлах, и мы хотим их все отправить в *stage*, то вместо имени файла поставьте точку.

Выполним *git status* для того, чтобы посмотреть на то, что сейчас происходит в нашем каталоге.

> git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: README.md

Как видно, в *stage* был добавлен один файл с именем *README.md* и теперь представленный набор изменений готов к отправке в репозиторий – т.е. к коммиту. Сделаем это.

> git commit -m "[create repository]"

[master (root-commit) 500067c] [create repository]

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

create mode 100644 README.md

Проверим статус каталога.

> git status

On branch master

nothing to commit, working tree clean

Как видно с момента последнего коммита никаких изменений в рабочем каталоге не производилось.

Теперь взглянем на список коммитов.

> git log

commit 500067cc0b80643d38e2a24e9e0699031ada6be3

Author: Writer <writer@someserver.com>

Date: Mon Feb 12 22:51:14 2018 +0500

[create repository]

Из приведенной информации видно, что был отправлен один коммит, который имеет *ID*: 500067cc0b80643d38e2a24e9e0699031ada6be3, более подробно об идентификаторах будет рассказано в следующих уроках. Автор данного коммита *Writer*, он (коммит) был создан *Mon Feb* 12 22:51:14 2018 +0500, с сообщением:  *[create repository]*. Это довольно подробная информация, когда коммитов станет много, такой формат вывода будет не очень удобным, сокращенный вариант выглядит так.

> git log --oneline

500067c [create repository]

Подведем небольшое резюме вышесказанному.

Создание пустого репозитория.

> git init

Добавление файлов в *stage*.

> git add filename

Создание коммита.

> git commit -m “message”

Просмотр статуса каталога.

> git status

Просмотр коммитов в репозитории.

> git log

Просмотр коммитов в репозитории с сокращенным выводом информации.

> git log --oneline

**Git для начинающих. Часть 6. Просмотр информации по коммитам**

Автор: [Marat Abdrakhmanov](https://devpractice.ru/author/writer/) | 07.03.2018

[Нет комментариев](https://devpractice.ru/git-for-beginners-part-6-git-log-work/#respond)

Рассмотрим инструмент системы контроля версий *git*, который позволяет делать выборку и представлять пользователю коммиты, отправленные в репозиторий, в соответствии с заданными параметрами.

В прошлом [уроке](https://devpractice.ru/git-for-beginners-part-5-create-repo-and-commit/) мы разобрались с тем, как фиксировать изменения в рабочей директории и отправлять коммиты в репозиторий. Рабочий процесс с использованием *git,*в упрощенном виде, выглядит следующим образом (пока не рассматриваем работу с удаленным репозиторием):

1. Внесение изменений в рабочую директорию.
2. Отправка изменений в *stage*.
3. Формирование и отправка коммита на базе того, что лежит в *stage*, в репозиторий.

В процессе работы, в вашем репозитории накопится больше количество коммитов и довольно часто будет возникать необходимость их просматривать. *Git* предоставляет удобный способ просмотра информации по коммитам. Для демонстрации возможностей *git*, создадим репозиторий и добавим в него один файл – *README.md*, о том, как это сделать, можете прочитать в [предыдущем уроке](https://devpractice.ru/git-for-beginners-part-5-create-repo-and-commit/).

Для просмотра информации по сделанным вами (или вашими коллегами) коммитам используется команда ***git log***.

> git log

commit a98cce47b59256d00a853c421af4f7b9f0dc0a29

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:10:51 2018 +0500

[create repository]

Как видно из полученной информации, в репозиторий был отправлен один коммит с сообщением ***“[create repository]”***, этот коммит сделал пользователь с именем ***Writer***, его *email:* ***writer@somecompany.com***, уникальный идентификатор коммита ***a98cce47b59256d00a853c421af4f7b9f0dc0a29***, и дата и время отправки коммита: ***5 марта 2018 в 23:10:51***.

Внесем еще несколько изменений в наш репозитории. Добавим текст в файл *README.md*.

> echo "Project 51" > README.md

Зафиксируем эти изменения в репозитории.

> git add .

> git commit -m "[add]: caption into README file"

Создадим файл *main.c* и добавим его в репозиторий.

> touch main.c

> git add .

> git commit -m "[create]: main file of program"

Таким образом в нашем репозитории уже должно быть три коммита, проверим это.

> git log

commit 2b826bb4929fb1c8166ef05b540ce2cc68f3ebb2

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:17:08 2018 +0500

[create]: main file of program

commit bc067c88c427dbedbb02817f9ae25241dcae4d07

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:15:12 2018 +0500

[add]: caption into README file

commit a98cce47b59256d00a853c421af4f7b9f0dc0a29

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:10:51 2018 +0500

[create repository]

Коммиты располагаются от новых к старым. Сделаем ещё несколько изменений.

> touch main.h

> git add .

> git commit -m "[create]: header for main"

> touch .gitignore

> git add .

> git commit -m "[create]: git ignore file"

> echo "\*.tmp" > .gitignore

> git add .

> git commit -m "[add] ignore .tmp files">

Снова получим список всех коммитов.

> git log

commit cf3d9d8f7b283267a085986e85cc8f152cca420d

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:21:59 2018 +0500

[add] ignore .tmp files

commit a7b88eed6110b6ebb1fc4d96f4399e4cbb8339e7

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:21:11 2018 +0500

[create]: git ignore file

commit c185b80ca916af7d6f068450f6cafb073d955c40

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:20:26 2018 +0500

[create]: header for main

commit 2b826bb4929fb1c8166ef05b540ce2cc68f3ebb2

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:17:08 2018 +0500

[create]: main file of program

commit bc067c88c427dbedbb02817f9ae25241dcae4d07

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:15:12 2018 +0500

[add]: caption into README file

commit a98cce47b59256d00a853c421af4f7b9f0dc0a29

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:10:51 2018 +0500

[create repository]

Количество коммитов в репозитории уже такое, что просматривать информацию о них в том виде, в котором выдает ***git log*** уже неудобно. Для того, чтобы сократить количество показываемой информации можно воспользоваться ключом ***–oneline***, при этом будет выведена часть идентификатора и сообщение коммита.

> git log --oneline

cf3d9d8 [add] ignore .tmp files

a7b88ee [create]: git ignore file

c185b80 [create]: header for main

2b826bb [create]: main file of program

bc067c8 [add]: caption into README file

a98cce4 [create repository]

В таком виде работать с коммитами уже намного удобнее. Если вы хотите просмотреть *n* последних коммитов, то укажите количество коммитов после ключа *-n*. Выведем три последних коммита.

> git log -n 3 --oneline

cf3d9d8 [add] ignore .tmp files

a7b88ee [create]: git ignore file

c185b80 [create]: header for main

Для вывода списка коммитов, начиная с какой-то временной метки, используйте ключ ***–since=”<date> <time>”***. Например, получим все коммиты, сделанные после 5-го марта 2018 года 23:21.

> git log --since="2018-03-05 23:21:00" --oneline

cf3d9d8 [add] ignore .tmp files

a7b88ee [create]: git ignore file

Для вывода списка коммитов до какой-то даты используется ключ ***–until***. Получим список коммитов, сделанных до 5-го марта 2018 года 23:21.

> git log --until="2018-03-05 23:21:00" --oneline

c185b80 [create]: header for main

2b826bb [create]: main file of program

bc067c8 [add]: caption into README file

a98cce4 [create repository]

Еще одним полезным ключом является ***–author***, который позволяет вывести список коммитов, сделанных конкретным автором.

> git log --author="Writer" --oneline

cf3d9d8 [add] ignore .tmp files

a7b88ee [create]: git ignore file

c185b80 [create]: header for main

2b826bb [create]: main file of program

bc067c8 [add]: caption into README file

a98cce4 [create repository]

В приведенном выше примере, мы вывели все коммиты сделанные пользователем с именем *Writer*. Т.к. в нашем репозитории все коммиты сделаны от имени данного автора, то при любых других именах, передаваемых параметру*–author*, мы будем получать пустой список.

И, напоследок, рассмотрим еще один инструмент. Если вы работали с *Linux*, то наверное, сталкивались с такой программой как *grep* – это утилита командной строки, которая, в переданном ей тексте, находит вхождения, соответствующие заданному регулярному выражению. Выведем все коммиты, в которых встречается слово *create*.

> git log --grep="create" --oneline

a7b88ee [create]: git ignore file

c185b80 [create]: header for main

2b826bb [create]: main file of program

a98cce4 [create repository]

Теперь коммиты со словом *add*.

> git log --grep="add" --oneline

cf3d9d8 [add] ignore .tmp files

bc067c8 [add]: caption into README file

Для более продуктивного использования данной команды рекомендуем ознакомиться с возможностями утилиты *grep*. На этом мы закончим обзор команды *git log*.

**Git для начинающих. Часть 7. Поговорим о HEAD и tree-ish**

Автор: [Marat Abdrakhmanov](https://devpractice.ru/author/writer/) | 18.03.2018

[3 комментария](https://devpractice.ru/git-for-beginners-part-7-head-and-tree-ish/#comments)

В этой статье мы коснемся двух понятий *git*, знание и понимание смысла которых позволит вам более эффективно работать с этой системой контроля версий.

* [*HEAD*](https://devpractice.ru/git-for-beginners-part-7-head-and-tree-ish/#p1)
* [*Tree-is*](https://devpractice.ru/git-for-beginners-part-7-head-and-tree-ish/#p2)

***HEAD***

Начнем с *HEAD*. *HEAD* – это указатель, задача которого ссылаться на определенный коммит в репозитории. Суть данного указателя можно попытаться объяснить с разных сторон.

**Во-первых**,*HEAD* – это указатель на коммит в вашем репозитории, который станет родителем следующего коммита. Для того, чтобы лучше понять это, обратимся к репозиторию, созданному в рамках [предыдущей статьи](https://devpractice.ru/git-for-beginners-part-6-git-log-work/), в этом репозитории сделано шесть коммитов, посмотрим на них.

> git log --oneline

cf3d9d8 [add] ignore .tmp files

a7b88ee [create]: git ignore file

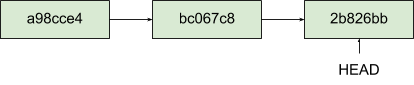
c185b80 [create]: header for main

2b826bb [create]: main file of programm

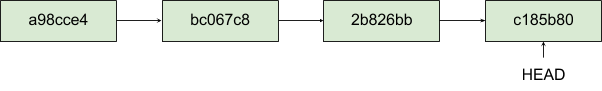
bc067c8 [add]: caption into README file

a98cce4 [create repository]

Эти коммиты создавались в порядке от самого нижнего (*a98cce4*) к самому верхнему (*cf3d9d8*). Каждый раз, когда мы отправляли новый коммит в репозиторий, *HEAD* смещался и указывал на него. Посмотрите на картинку ниже: на ней показана ситуация, когда были отправлены три первых коммита.



После того как вы отправили коммит с *id = 2b826bb*, указатель *HEAD* стал показывать на него, т.е. данный коммит будет родителем для следующего, и когда мы сделаем еще один коммит, *HEAD* сместится.



**Во-вторых**,*HEAD* указывает на коммит, относительного которого будет создана рабочая копия во-время операции *checkout*. Другими словами, когда вы переключаетесь с ветки на ветку (о ветвлении в *git* будет рассказано в одной из ближайших статей), используя операцию *checkout*, то в вашем репозитории указатель *HEAD* будет переключаться между последними коммитами выбираемых вами ветвей.

В нашем репозитории пока только одна ветвь – *master*, но и этого будет достаточно, чтобы показать зависимость между положением указателя *HEAD* и операцией *checkout*.

Текущее состояние репозитория выглядит так, как показано на рисунке ниже.

https://devpractice.ru/wp-content/uploads/2018/03/git-head-pic3.png

Для того, чтобы скопировать снимок репозитория относительно последнего коммита ветки *master*, т.е. того на который указывает *HEAD*, необходимо выполнить следующую команду.

> git checkout master

Switched to branch 'master'

Содержимое репозитория, в данном случае, выглядит так.

> git log --oneline

cf3d9d8 [add] ignore .tmp files

a7b88ee [create]: git ignore file

c185b80 [create]: header for main

2b826bb [create]: main file of programm

bc067c8 [add]: caption into README file

a98cce4 [create repository]

Теперь передвинем указатель *HEAD* на коммит с *id=2b826bb*.

https://devpractice.ru/wp-content/uploads/2018/03/git-head-pic4.png

Для этого передадим команде *checkout* идентификатор коммита.

> git checkout 2b826bb

Note: checking out '2b826bb'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -b with the checkout command again. Example:

git checkout -b <new-branch-name>

HEAD is now at 2b826bb... [create]: main file of programm

Обратите внимание на текст, который напечатал *git*, после того, как была выполнена эта команда. Нас интересует самая последняя строка “*HEAD is now at 2b826bb…*”, теперь *HEAD* указывает на коммит с *id=2b826bb* – именно то, что мы хотели. Посмотрим на текущий список коммитов.

> git log --oneline

2b826bb [create]: main file of programm

bc067c8 [add]: caption into README file

a98cce4 [create repository]

*Git* выводит коммиты, которые были сделаны до того коммита, на который ссылается *HEAD*.

Вернем *HEAD* на прежнее место.

> git checkout cf3d9d8

Previous HEAD position was 2b826bb... [create]: main file of programm

HEAD is now at cf3d9d8... [add] ignore .tmp files

> git log --oneline

cf3d9d8 [add] ignore .tmp files

a7b88ee [create]: git ignore file

c185b80 [create]: header for main

2b826bb [create]: main file of programm

bc067c8 [add]: caption into README file

a98cce4 [create repository]

Все вернулось на прежнее место. Таким образом, вы можете получать в виде рабочей копии содержимое репозитория на момент отправки того или иного коммита. Перейдем в каталог .*git*, в котором находится наш репозиторий, он расположен в корневой директории нашего проекта, и посмотрим его содержимое.

> cd .git

> ls -la

total 21

drwxr-xr-x 1 User 197121 0 мар 18 17:10 ./

drwxr-xr-x 1 User 197121 0 мар 18 17:10 ../

-rw-r--r-- 1 User 197121 24 мар 5 23:21 COMMIT\_EDITMSG

-rw-r--r-- 1 User 197121 184 мар 5 23:10 config

-rw-r--r-- 1 User 197121 73 мар 5 23:10 description

-rw-r--r-- 1 User 197121 41 мар 18 17:10 HEAD

drwxr-xr-x 1 User 197121 0 мар 5 23:10 hooks/

-rw-r--r-- 1 User 197121 441 мар 18 17:10 index

drwxr-xr-x 1 User 197121 0 мар 5 23:10 info/

drwxr-xr-x 1 User 197121 0 мар 5 23:10 logs/

drwxr-xr-x 1 User 197121 0 мар 5 23:21 objects/

drwxr-xr-x 1 User 197121 0 мар 5 23:10 refs/

В данном каталоге содержится файл *HEAD*, в нем находится идентификатор, на который ссылается данный указатель. Посмотрим содержимое файла *HEAD*.

> cat HEAD

cf3d9d8f7b283267a085986e85cc8f152cca420d

*HEAD* указывает на коммит *cf3d9d8*.

**Tree-ish**

Понятие *tree-ish* часто используется в документации по *git*. *Tree-ish* – это то, что указывает на коммит, эту сущность мы можем передавать в качестве аргумента для команд *git*. Вот список того, чем может являться *tree-ish.*

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

|          Tree-ish       | Examples

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

|  1. <sha1>                | dae86e1950b1277e545cee180551750029cfe735

|  2. <describeOutput>      | v1.7.4.2-679-g3bee7fb

|  3. <refname>             | master, heads/master, refs/heads/master

|  4. <refname>@{<date>}    | master@{yesterday}, HEAD@{5 minutes ago}

|  5. <refname>@{<n>}       | master@{1}

|  6. @{<n>}                | @{1}

|  7. @{-<n>}               | @{-1}

|  8. <refname>@{upstream}  | master@{upstream}, @{u}

|  9. <rev>^                | HEAD^, v1.5.1^0

| 10. <rev>~<n>             | master~3

| 11. <rev>^{<type>}        | v0.99.8^{commit}

| 12. <rev>^{}              | v0.99.8^{}

| 13. <rev>^{/<text>}       | HEAD^{/fix nasty bug}

| 14. :/<text>              | :/fix nasty bug

| 15. <rev>:<path>          | HEAD:README.txt, master:sub-directory/

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

Рассмотрим работу с *tree-ish* на примере команды *git show*.

> git show cf3d9d8f -q

commit cf3d9d8f7b283267a085986e85cc8f152cca420d

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:21:59 2018 +0500

[add] ignore .tmp files

> git show -q HEAD

commit cf3d9d8f7b283267a085986e85cc8f152cca420d

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:21:59 2018 +0500

[add] ignore .tmp files

> git show -q master

commit cf3d9d8f7b283267a085986e85cc8f152cca420d

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:21:59 2018 +0500

[add] ignore .tmp files

> git show -q @{5}

commit cf3d9d8f7b283267a085986e85cc8f152cca420d

Author: Writer <writer@somecompany.com>

Date: Mon Mar 5 23:21:59 2018 +0500

[add] ignore .tmp files

Во всех примерах, представленных выше, команде *git show* мы передаем различные *tree-ish*, которые на самом деле указывают на одно и тоже место – последний коммит.

# Git для начинающих. Часть 8. Добавление, удаление и переименование файлов в репозитории

Автор: [Marat Abdrakhmanov](https://devpractice.ru/author/writer/) | 22.03.2018

[1 комментарий](https://devpractice.ru/git-for-beginners-part-8-add-remove-rename/#comments)

В рамках данного урока рассмотрим вопросы, касающиеся добавления, удаления и переименования файлов в *git* репозитории.

* [Добавление файлов в git репозиторий](https://devpractice.ru/git-for-beginners-part-8-add-remove-rename/#p1)
* [Удаление файлов из *git* репозитория и из *stage*](https://devpractice.ru/git-for-beginners-part-8-add-remove-rename/#p2)
  + [Удаление файла из *stage*](https://devpractice.ru/git-for-beginners-part-8-add-remove-rename/#p21)
  + [Удаление файлов из *git* репозитория](https://devpractice.ru/git-for-beginners-part-8-add-remove-rename/#p22)
    - [Первый способ](https://devpractice.ru/git-for-beginners-part-8-add-remove-rename/#p221)
    - [Второй способ](https://devpractice.ru/git-for-beginners-part-8-add-remove-rename/#p222)
* [Переименование файлов в *git* репозитории](https://devpractice.ru/git-for-beginners-part-8-add-remove-rename/#p3)
  + [Первый способ](https://devpractice.ru/git-for-beginners-part-8-add-remove-rename/#p31)
  + [Второй способ](https://devpractice.ru/git-for-beginners-part-8-add-remove-rename/#p32)

## ****Добавление файлов в *git* репозиторий****

Добавление файлов в репозиторий – это достаточно простая операция, мало чем отличающаяся от отправки изменений в отслеживаемых файлах в репозиторий. Мы уже не раз выполняли эту операцию в предыдущих уроках, но сделаем это ещё раз. Создадим новый репозиторий, для этого перейдите в каталог, в котором вы хотите его расположить и введите команду *git init*.

> git init

Создайте в каталоге файл *README.md* любым удобным для вас способом, мы сделаем это с помощью команды *touch*.

> touch README.md

Теперь проверим состояние отслеживаемой директории.

> git status

On branch master

Initial commit

Untracked files:

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

README.md

nothing added to commit but untracked files present (use "git add" to track)

Как вы можете видеть: в рабочей директории есть один неотслеживаемый файл *README.md*. *Git* нам подсказывает, что нужно сделать для того, чтобы начать отслеживать изменения в файле *README.md*: необходимо выполнить команду *git add*, сделаем это.

> git add README.md

Посмотрим ещё раз на состояние.

> git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: README.md

Видно, что информация о появлении нового файла попала в *stage*. Для того чтобы это изменение зафиксировалось в репозитории необходимо выполнить команду *git commit*.

> git commit -m "add README.md file"

[master (root-commit) 0bb6c94] add README.md file

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

create mode 100644 README.md

Теперь в рабочей директории и в *stage* нет объектов, информацию об изменении которых необходимо внести в репозиторий.

> git status

On branch master

nothing to commit, working tree clean

В репозиторий был сделан один коммит.

> git log --oneline

0bb6c94 add README.md file

## ****Удаление файлов из *git* репозитория и из *stage*****

### **Удаление файла из *stage***

Вначале разберемся со *stage*. Создадим ещё один файл.

> touch main.c

“Отправим” файл *main.c* в *stage*.

> git add main.c

Внесем изменения в *README.md*.

> echo "# README" > README.md

Информацию об этом также отправим в *stage*.

> git add README.md

Посмотрим на состояние *stage*.

> git status

On branch master

Changes to be committed:

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

modified: README.md

new file: main.c

Если нам необходимо убрать из *stage*, какой-то из этих файлов (main.c или README.md), то для этого можно воспользоваться командой *git –rm cashed <filename>*, сделаем это для файла *main.c*.

> git rm --cached main.c

rm 'main.c'

Теперь посмотрим на состояние рабочей директории и *stage*.

> git status

On branch master

Changes to be committed:

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

modified: README.md

Untracked files:

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

main.c

Видно, что изменения в файле *README.md* готовы для коммита, а вот файл *main.c* перешел в состояние – неотслеживаемый. Отправим *main.c* в stage и, после этого, сделаем коммит в репозиторий.

> git add main.c

> git commit -m "add main.c and do some changes in README.md"

[master 49049bc] add main.c and do some changes in README.md

2 files changed, 1 insertion(+)

create mode 100644 main.c

### **Удаление файлов из *git* репозитория**

Удалить файл из репозитория можно двумя способами: **первый** – удалить его из рабочей директории и уведомить об этом *git*; **второй** – воспользоваться средствами *git*. **Начнем с первого способа**. Для начала посмотрим, какие файлы у нас хранятся в репозитории.

> git ls-tree master

100644 blob 7e59600739c96546163833214c36459e324bad0a README.md

100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 main.c

Удалим файл *main.c* из рабочей директории.

> rm main.c

> ls

README.md

Уведомим об этом систему *git*.

> git rm main.c

rm 'main.c'

Вместо команды *git rm* можно использовать *git add*, но само слово *add* в данном случае будет звучать несколько неоднозначно, поэтому лучше использовать *rm*. На данном этапе еще можно вернуть все назад с помощью команды *git checkout — <filename>*, в результате, в нашу рабочую директорию будет скопирован файл из репозитория. Создадим коммит, фиксирующий удаление файла.

> git commit -m "remove main.c"

[master d4e22ae] remove main.c

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

delete mode 100644 main.c

Теперь в репозитории остался только один файл *README.md*.

> git ls-tree master

100644 blob 7e59600739c96546163833214c36459e324bad0a README.md

**Второй способ** – это сразу использовать команду *git rm* без предварительного удаления файла из директории. Вновь создадим файл *main.c* и добавим его в репозиторий.

> touch main.c

> git add main.c

> git commit -m "add main.c file"

[master 6d93049] add main.c file

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

create mode 100644 main.c

> git ls-tree master

100644 blob 7e59600739c96546163833214c36459e324bad0a README.md

100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 main.c

Удалим файл из репозитория.

> git rm main.c

rm 'main.c'

> git commit -m "deleted: main.c file"

[master ba7d027] deleted: main.c file

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

delete mode 100644 main.c

Файла *main.c* больше нет в репозитории.

> git ls-tree master

100644 blob 7e59600739c96546163833214c36459e324bad0a README.md

Его также нет и в рабочем каталоге.

> ls

README.md

Удалите файл *README.md* из репозитория самостоятельно.

## ****Переименование файлов в *git* репозитории****

Как и в случае с удалением, переименовать файл в *git* репозитории можно двумя способами – с использованием и без использования средств операционной системы.

**Первый способ**. Создадим файл *test\_main\_file.c* и добавим его в репозиторий.

> touch test\_main\_file.c

> git add test\_main\_file.c

> git commit -m "add test\_main\_file.c"

[master 6cf53ac] add test\_main\_file.c

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

create mode 100644 test\_main\_file.c

Содержимое репозитория после этого будет выглядеть так.

> git ls-tree master

100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 test\_main\_file.c

Переименуем его на *test\_main.c*.

Сделаем это в рабочей директории.

> mv test\_main\_file.c test\_main.c

Теперь отправим изменение в репозиторий.

> git add .

> git commit -m "Rename test\_main\_file.c"

[master 79528c4] Rename test\_main\_file.c

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

rename test\_main\_file.c => test\_main.c (100%)

В репозитории и в рабочей директории будет находится только файл *test\_main.c*.

> git ls-tree master

100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 test\_main.c

> ls

test\_main.c

**Второй способ.**

В рамках второго способа рассмотрим работу с командой *git mv*. Переименуем файл *test\_main.c* в *main.c*. Текущее содержимое репозитория и рабочего каталога.

> git ls-tree master

100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 test\_main.c

> ls

test\_main.c

Переименуем файл *test\_main.c* на *main.c* средствами *git*.

> git mv test\_main.c main.c

> git commit -m "Rename test\_main.c file"

[master c566f0e] Rename test\_main.c file

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

rename test\_main.c => main.c (100%)

Имя файла изменилось как в репозитории так и в рабочем каталоге.

> git ls-tree master

100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 main.c

> ls

main.c

# Git для начинающих. Часть 9. Как удалить коммит в git?

Автор: [Marat Abdrakhmanov](https://devpractice.ru/author/writer/) | 01.04.2018

[2 комментария](https://devpractice.ru/git-for-beginners-part-9-how-undo-commit-in-git/#comments)

Рассмотрим довольно важный вопрос: как удалить коммит в git? Начнем с вопроса отмены изменений в рабочей директории, после этого перейдем к репозиторию. В рамках этой темы изучим вопросы удаления и замены последнего коммита, работу с отдельными файлами и использование команд git revert и git reset.

* [Отмена изменений в файлах в рабочей директории](https://devpractice.ru/git-for-beginners-part-9-how-undo-commit-in-git/#p1)
* [Удаление коммитов в git](https://devpractice.ru/git-for-beginners-part-9-how-undo-commit-in-git/#p2)
  + [Работа с последним коммитом](https://devpractice.ru/git-for-beginners-part-9-how-undo-commit-in-git/#p21)
  + [Отмена изменений в файле в выбранном коммите](https://devpractice.ru/git-for-beginners-part-9-how-undo-commit-in-git/#p22)
  + [Использование git revert для быстрой отмены изменений](https://devpractice.ru/git-for-beginners-part-9-how-undo-commit-in-git/#p23)
  + [Удаление группы коммитов](https://devpractice.ru/git-for-beginners-part-9-how-undo-commit-in-git/#p24)
    - [Удаление коммитов из репозитория (без изменения рабочей директории) (ключ –soft)](https://devpractice.ru/git-for-beginners-part-9-how-undo-commit-in-git/#p241)
    - [Удаление коммитов из репозитория и очистка stage (без изменения рабочей директории) (ключ –mixed)](https://devpractice.ru/git-for-beginners-part-9-how-undo-commit-in-git/#p242)
    - [Удаление коммитов из  репозитория, очистка stage и внесение изменений в  рабочую директорию (ключ –hard)](https://devpractice.ru/git-for-beginners-part-9-how-undo-commit-in-git/#p243)

## ****Отмена изменений в файлах в рабочей директории****

Если вы сделали какие-то изменения в файле и хотите вернуть предыдущий вариант, то для этого следует обратиться к репозиторию и взять из него файл, с которым вы работаете. Таким образом, в вашу рабочую директорию будет скопирован файл из репозитория с заменой. Например, вы работаете с файлом main.c и внесли в него какие-то изменения. Для того чтобы  вернуться к предыдущей версии (последней отправленной в репозиторий)  воспользуйтесь командой git checkout.

> git checkout -- main.c

Ключ “- -” означает, что нас интересует файл в текущем бранче (ветке).

## ****Отмена коммитов в**** git

### Работа с последним коммитом

Для демонстранции возможностей git создадим новый каталог и инициализируем в нем репозиторий.

> git init

Добавим в каталог файл main.c.

> touch main.c

Отправим изменения в репозиторий.

> git add main.c

> git commit -m "first commit"

[master (root-commit) 86f1495] first commit

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

create mode 100644 main.c

Внесем изменения в файл.

> echo "// main.c file" > main.c

И сделаем еще один коммит.

> git add main.c

> git commit -m "second commit"

[master d142679] second commit

1 file changed, 1 insertion(+)

В репозиторий, на данный момент, было сделано два коммита.

> git log --oneline

d142679 second commit

86f1495 first commit

Теперь удалим последний коммит и вместо него отправим другой. Предварительно изменим содержимое файла main.c.

> echo "// author: Writer" > main.c

Отправим изменения в репозиторий с заметой последнего коммита.

> git add main.c

> git commit --amend -m "third commit"

> git log --oneline

18411fd third commit

86f1495 first commit

Как вы можете видеть: из репозитория пропал коммит с id=d142679, вместо него теперь коммит с id=18411fd.

### Отмена изменений в файле в выбранном коммите

Сделаем ещё несколько изменений в нашем файле main.c, каждое из которых будет фиксироваться коммитом в репозиторий.

> echo "// Some text 1" > main.c

> git add main.c

> git commit -m "fourth commit"

[master dcf7253 ] fourth commit

1 file changed, 1 insertion(+), 1 deletion(-)

> echo "// Some text 2" > main.c

> git add main.c

> git commit -m "fifth commit"

[master 7f2eb3a ] fifth commit

1 file changed, 1 insertion(+), 1 deletion(-)

> git log --oneline

7f2eb3a fifth commit

dcf7253 fourth commit

18411fd third commit

86f1495 first commit

Помните, что в предыдущем разделе мы поменяли коммит с сообщением “second commit” на “third commit”, поэтому он идет сразу после “first commit”.

Представим ситуацию, что два последних коммита были неправильными, и нам нужно вернуться к версии 18411fd и внести изменения именно в нее. В нашем примере, мы работаем только с одним файлом, но в реальном проекте файлов будет много, и после коммитов, в рамках которых вы внесли изменения в интересующий вас файл, может быть ещё довольно много коммитов, фиксирующих изменения в других файлах. Просто так взять и удалить коммиты из середины ветки не получится – это нарушит связность, что идет в разрез с идеологией git. Одни из возможных вариантов – это получить версию файла из нужного нам коммита, внести в него изменения и сделать новый коммит. Для начала посмотрим на содержимое файла main.c из последнего, на текущий момент, коммита.

> git checkout main.c

> cat main.c

// Some text 2

Для просмотра содержимого файла в коммите с id=18411fd воспользуемся правилами работы с tree-ish (об этом подробно написано [здесь](https://devpractice.ru/git-for-beginners-part-7-head-and-tree-ish/#p2))

> git show 18411fd:main.c

// author: Writer

Переместим в рабочую директорию файл main.c из репозитория с коммитом id=18411fd.

> git checkout 18411fd -- main.c

> cat main.c

// author: Writer

Мы видим, что теперь содержимое файла main.c соответствует тому, что было на момент создания коммита с  id=18411fd. Сделаем коммит в репозиторий и в сообщении укажем, что он отменяет два предыдущих.

> git add main.c

> git commit -m "return main.c from third commit"

[master cffc5ad] return main.c from third commit

1 file changed, 1 insertion(+), 1 deletion(-)

> git log --oneline

cffc5ad return main.c from third commit

7f2eb3a fifth commit

dcf7253 fourth commit

18411fd third commit

86f1495 first commit

Таким образом мы вернулись к предыдущей версии файла main.c и при этом сохранили всю историю изменений.

### Использование git revert для быстрой отмены изменений

Рассмотрим ещё одни способ отмены коммитов, на этот раз воспользуемся командой git revert.

В нашем примере, отменим коммит с id=cffc5ad. После того как вы введете команду git revert (см. ниже), система git выдаст сообщение в текстовом редакторе, если вы согласны с тем, что будет написано в открытом файле, то просто сохраните его и закройте. В результате изменения будут применены, и автоматически сформируется и отправится в репозиторий коммит.

> git revert cffc5ad

[master 81499da] Revert "return main.c from third commit"

1 file changed, 1 insertion(+), 1 deletion(-)

Если вы хотите поменять редактор, то воспользуйтесь командой.

> git config core.editor "notepad.exe"

Обратите внимание, что в этом случае будут изменены настройки для текущего репозитория. Более подробно об изменении настроек смотрите в “[Git для начинающих. Часть 3. Настройка Git](https://devpractice.ru/git-for-beginners-part-3-config-git/" \t "_blank)”

Проверим, применялась ли настройка.

> git config core.editor

notepad.exe

Посмотрим на список коммитов в репозитории.

> git log --oneline

81499da Revert "return main.c from third commit"

cffc5ad return main.c from third commit

7f2eb3a fifth commit

dcf7253 fourth commit

18411fd third commit

86f1495 first commit

Содержимое файла вернулось к тому, что было сделано в рамках коммита с id=7f2eb3a.

> cat main.c

// Some text 2

> git show 7f2eb3a:main.c

// Some text 2

### Отмена группы коммитов

**ВНИМАНИЕ! Используйте эту команду очень аккуратно!**

Если вы не знакомы с концепцией указателя HEAD, то обязательно прочитайте статью “[Git для начинающих. Часть 7. Поговорим о HEAD и tree-ish](https://devpractice.ru/git-for-beginners-part-7-head-and-tree-ish/" \t "_blank)“. HEAD указывает на коммит в репозитории, с которого будет вестись дальнейшая запись, т.е. на родителя следующего коммита. Существует три опции, которые можно использовать с командой git reset для изменения положения HEAD и управления состоянием stage и рабочей директории, сейчас мы все это подробно разберем.

#### **Удаление коммитов из репозитория (без изменения рабочей директории) (ключ** –soft**)**

Для изменения положения указателя HEAD в репозитории, без оказания влияния рабочую директорию (в stage, при этом, будет зафиксированно отличие рабочей директории от репозитория), используйте ключ –soft. Посмотрим ещё раз на наш репозиторий.

> git log --oneline

81499da Revert "return main.c from third commit"

cffc5ad return main.c from third commit

7f2eb3a fifth commit

dcf7253 fourth commit

18411fd third commit

86f1495 first commit

Содержимое файла main.с в рабочей директории.

> cat main.c

// Some text 2

Содержимое файла main.с в репозитории.

> git show HEAD:main.c

// Some text 2

Теперь переместим HEAD в репозитории на коммит с id=dcf7253.

> git reset --soft dcf7253

Получим следующий список коммитов.

> git log --oneline

dcf7253 fourth commit

18411fd third commit

86f1495 first commit

Содержимое файла main.c в репозитории выглядит так.

> git show HEAD:main.c

// Some text 1

В рабочей директории файл main.c остался прежним (эти изменения отправлены в stage).

> cat main.c

// Some text 2

Для того, чтобы зафиксировать в репозитории последнее состояние файла main.c сделаем коммит.

> git commit -m "soft reset example"

[master db1a8b0] soft reset example

1 file changed, 1 insertion(+), 1 deletion(-)

Посмотрим на список коммитов.

> git log --oneline

db1a8b0 soft reset example

dcf7253 fourth commit

18411fd third commit

86f1495 first commit

Как видите из репозитория пропали следующие коммиты:

81499da Revert "return main.c from third commit"

cffc5ad return main.c from third commit

7f2eb3a fifth commit

#### Удаление коммитов из репозитория и очистка stage (без изменения рабочей директории) (ключ –mixed)

Если использовать команду git reset с аргументом –mixed, то в репозитории указатель HEAD переместится на нужный коммит, а также будет сброшено содержимое stage. Отменим последний коммит.

> git reset --mixed dcf7253

Unstaged changes after reset:

M       main.c

В результате изменилось содержимое репозитория.

> git log --oneline

dcf7253 fourth commit

18411fd third commit

86f1495 first commit

Содержимое файла main.c в последнем коммите выглядит так.

> git show HEAD:main.c

// Some text 1

Файл main.c в рабочей директории не изменился.

> cat main.c

// Some text 2

Отправим изменения вначале в stage, а потом в репозиторий.

> git add main.c

> git commit -m "mixed reset example"

[master ab4ef00] mixed reset example

1 file changed, 1 insertion(+), 1 deletion(-)

#### **Удаление коммитов из репозитория, очистка** stage **и внесение изменений в  рабочую директорию (ключ** –hard**)**

Если вы воспользуетесь ключем –hard, то обратного пути уже не будет. Вы не сможете восстановить данные из рабочей директории. Все компоненты git (репозиторий, stage и рабочая директория) будут приведены к одному виду в соответствии с коммитом, на который будет перенесен указатель HEAD.

Текущее содержимое репозитория выглядит так.

> git log --oneline

ab4ef00 mixed reset example

dcf7253 fourth commit

18411fd third commit

86f1495 first commit

Посмотрим на содержимое файла main.c в каталоге и репозитории.

> cat main.c

// Some text 2

> git show HEAD:main.c

// Some text 2

Содержимое файлов идентично.

Удалим все коммиты до самого первого с id=86f1495.

> git reset --hard 86f1495

HEAD is now at 86f1495 first commit

Содержимое репозитория.

> git log --oneline

86f1495 first commit

Состояние рабочей директории и stage.

> git status

On branch master

nothing to commit, working tree clean

Содержимое файла main.c в репозитории и в рабочей директории.

> cat main.c

> git show HEAD:main.c

Файл main.c пуст.

Т.к. мы воспользовались командой git reset с ключем –hard, то восстановить прежнее состояние нам не получится.