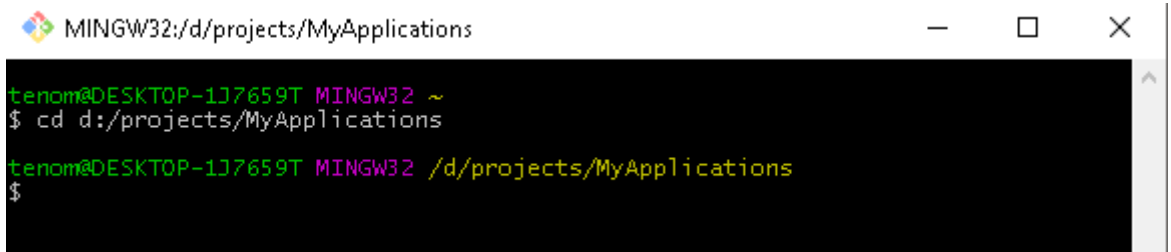


1) Перемещение в директориях.

Команда **cd** позволяет перемещаться между директориями.

Предположим наш рабочий проект находится в

D:/projects/MyApplications. Для того что бы в дальнейшем работать с ним, нам нужно перейти в соответствующую директорию. Для этого мы прописываем **cd D:/projects/MyApplications**. В итоге мы увидим:



```
MINGW32:/d/projects/MyApplications
tenom@DESKTOP-1J7659T MINGW32 ~
$ cd d:/projects/MyApplications
tenom@DESKTOP-1J7659T MINGW32 /d/projects/MyApplications
$
```

2) Инициализация.

Работая локально, Git хранит историю проекта в подкаталоге **.git** вашего рабочего каталога.

Если мы работаем впервые с проектом, нам необходимо сообщить об этом GIT. Он в свою очередь создаст в директории проекта поддиректорию **.git** с необходимыми файлами и конфигурациями. Для этого в программе GIT есть команда **init**. Ее мы можем прописать двумя способами:

1) Перейти в рабочую директорию и прописать **git init**.

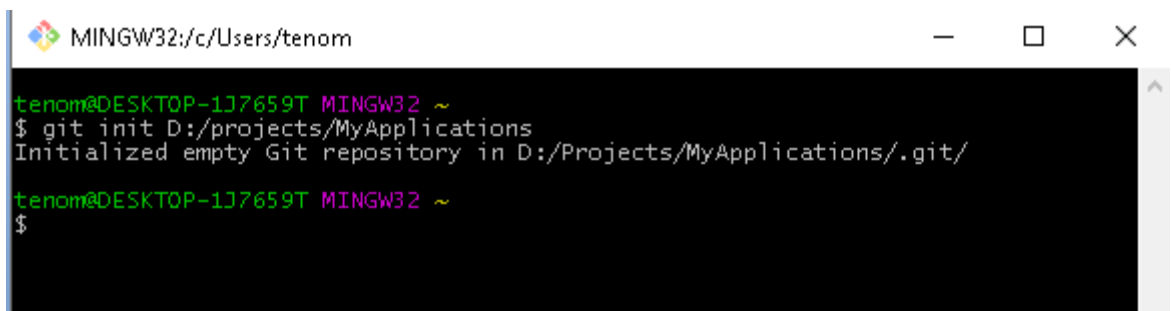
cd D:/projects/MyApplications

git init.

2) Прописать **git init** с указанием рабочей директории:

git init D:/projects/MyApplications

Результат будет одинаков



```
MINGW32:/c/Users/tenom
tenom@DESKTOP-1J7659T MINGW32 ~
$ git init D:/projects/MyApplications
Initialized empty Git repository in D:/Projects/MyApplications/.git/
tenom@DESKTOP-1J7659T MINGW32 ~
$
```

Обязательно перед командами GIT пишем оператор **git**, так как есть команды для работы в командной строке от ОС, такие как **cd**.

3) Отслеживание изменений.

После того как мы инициализировали наш проект в GIT, добавим в него данные с которыми будем работать. Для примера в корневой директории я создал файл Target.class который содержит пример кода на Java. Так же создал под. директорию, в которой находится текстовый файл README. В нем описано назначение содержимого Target.class.

Иными словами мы внесли изменение в проект относительно предыдущего состояния, то-есть относительно инициализации пустого проекта.

Для того что бы отслеживать внесенные изменения в проект использоваться команда **status**.

Примечание:

*Первоочередно нужно перейти в рабочую директорию используя команду **cd**, иначе прописав status мы получим сообщение об ошибке*

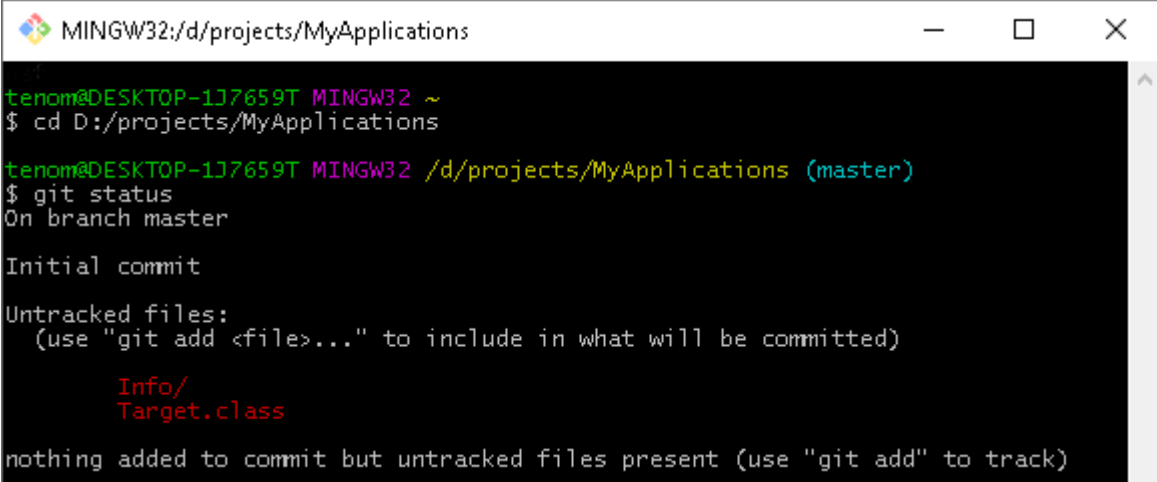
fatal: Not a git repository (or any of the parent directories): .git

Так же мы не можем писать

git status D:/projects/MyApplications

Так как снова получим вышеуказанное сообщением об ошибке.

Итак листинг конечной команды **git status**. Результатом ее использования будут внесенные изменения (см. скриншот ниже). .



```
tenom@DESKTOP-1J7659T MINGW32 ~
$ cd D:/projects/MyApplications

tenom@DESKTOP-1J7659T MINGW32 /d/projects/MyApplications (master)
$ git status
On branch master

Initial commit

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

    Info/
    Target.class

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

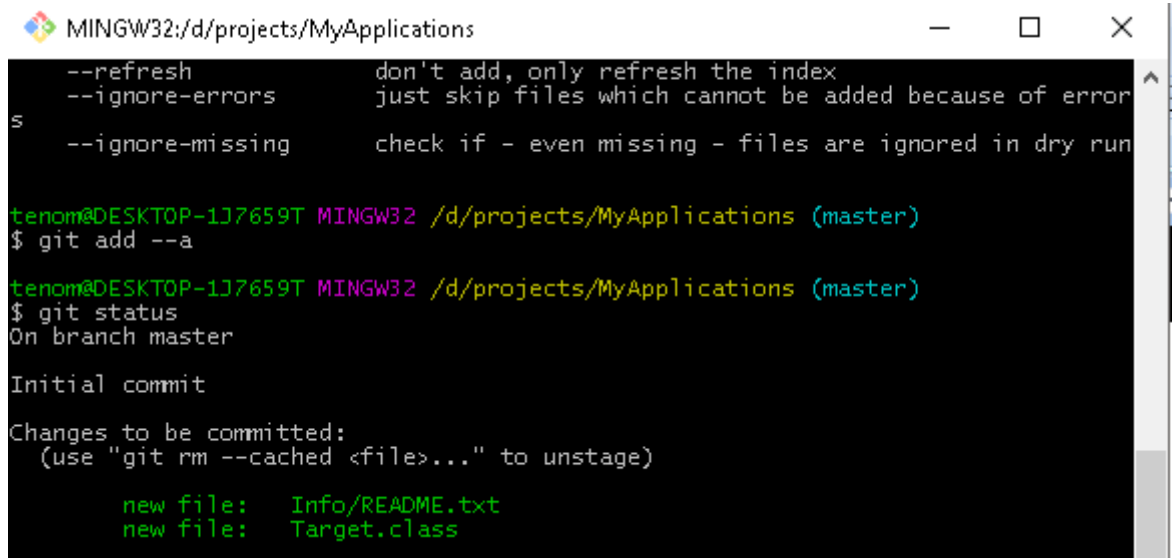
Изменения выделены красным

Проиндексированные изменения выделены зеленым

4) Индексация изменений.

Для каждого отслеживаемого файла, Git записывает такую информацию, как размер, время создания и время последнего изменения, в файле, известном как «индекс». Чтобы определить, был ли файл изменен, Git сравнивает его текущие характеристики с сохраненными в индексе. Если они совпадают, то Git не станет перечитывать файл заново, поскольку считывание этой информации значительно быстрее, чем чтение всего файла. Если мы редактировали лишь несколько файлов, Git может обновить свой индекс почти мгновенно.

Для того что бы проиндексировать изменения используется команда **git add -a**, в которой **--a**, значит проиндексировать все. Снова прописываем **git status** и видим результат.



```
MINGW32:/d/projects/MyApplications
--refresh          don't add, only refresh the index
--ignore-errors    just skip files which cannot be added because of error
--ignore-missing   check if - even missing - files are ignored in dry run

tenom@DESKTOP-1J7659T MINGW32 /d/projects/MyApplications (master)
$ git add --a

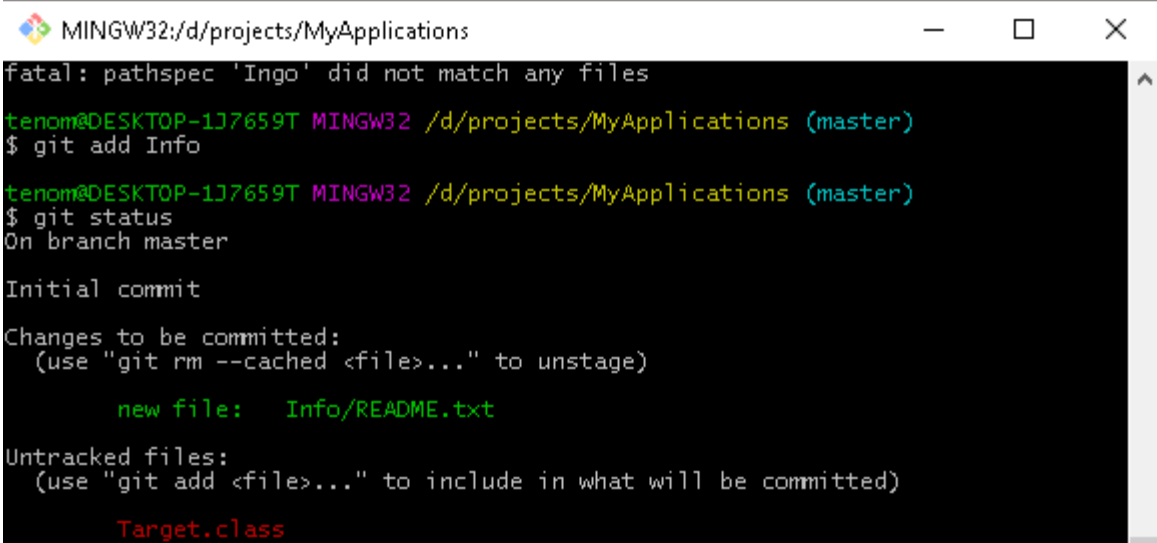
tenom@DESKTOP-1J7659T MINGW32 /d/projects/MyApplications (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   Info/README.txt
        new file:   Target.class
```

Так же мы можем индексировать только один файл или директорию
git add Info



```
MINGW32:/d/projects/MyApplications
fatal: pathspec 'Ingo' did not match any files

tenom@DESKTOP-1J7659T MINGW32 /d/projects/MyApplications (master)
$ git add Info

tenom@DESKTOP-1J7659T MINGW32 /d/projects/MyApplications (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   Info/README.txt

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

        Target.class
```

На скриншоте видно что мы проиндексировали только **Info** в котором есть README.TXT, так же мы видим что у нас есть не проиндексированное изменение в корневой директории – Target.class.

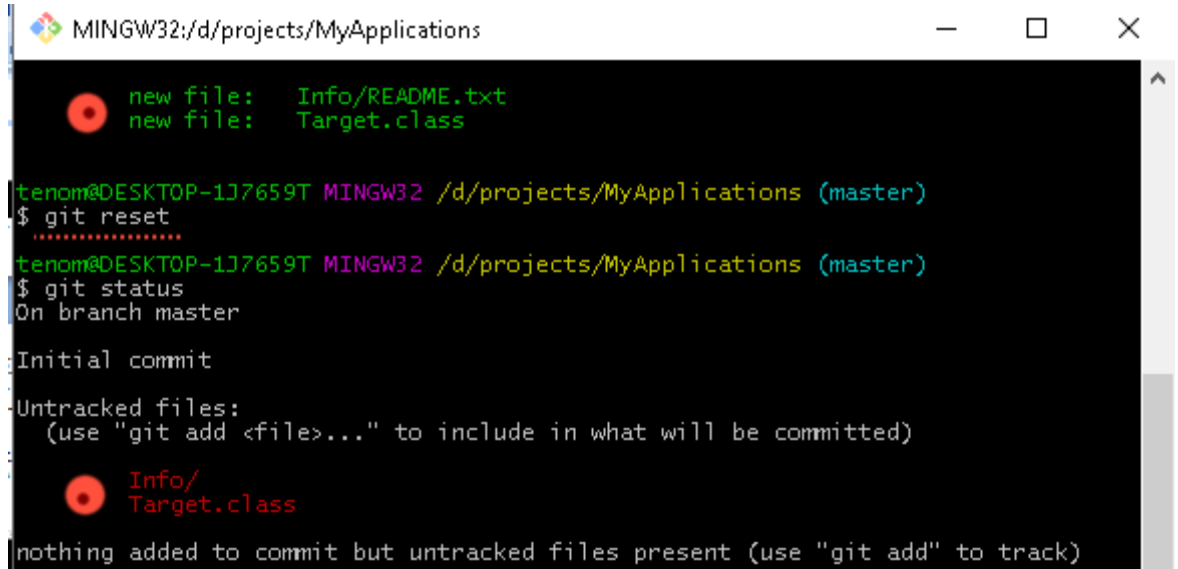
Если в директории **Info** было бы несколько файлов то мы бы увидели к примеру:

```
new filele: Info/README.txt
new filele: Info/README_1.txt
new filele: Info/README_2.txt
```

Можено заметить что равноценно тому, как изменение отображаются красным, проиндексированные файлы отображаются зеленым цветом.

5) Удаление индексации

Если по каким то причинам нам не обходимо удалить индексацию, мы можем использовать команду **git reset**



```
MINGW32:/d/projects/MyApplications

new file:   Info/README.txt
new file:   Target.class

tenom@DESKTOP-1J7659T MINGW32 /d/projects/MyApplications (master)
$ git reset
.....
tenom@DESKTOP-1J7659T MINGW32 /d/projects/MyApplications (master)
$ git status
On branch master

Initial commit

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

Info/
Target.class

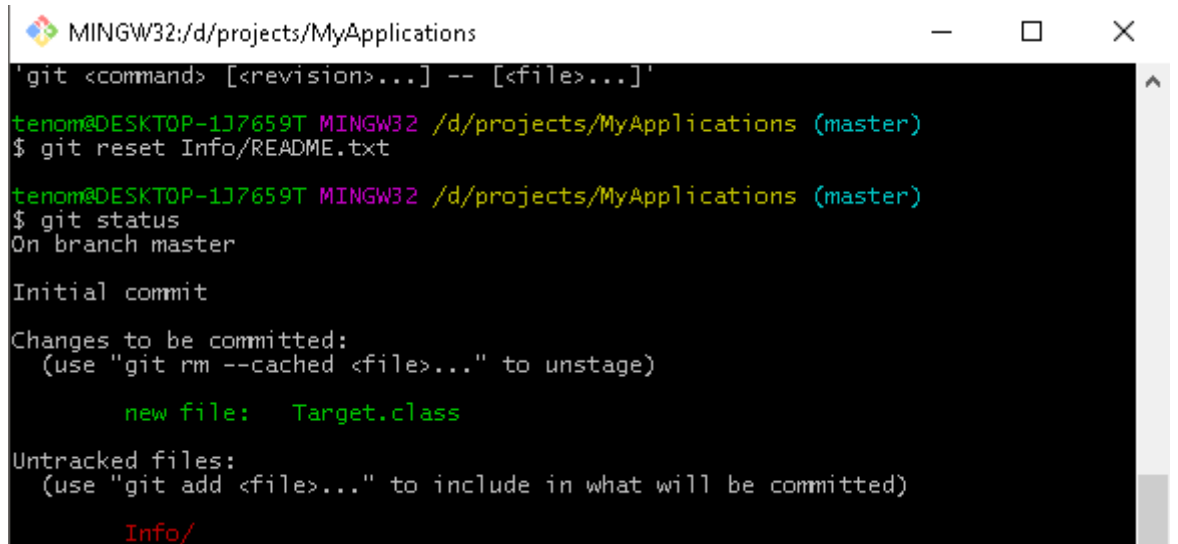
nothing added to commit but untracked files present (use "git add" to track)
```

Как и в случае с индексацией можно разиндексировать один файл или директорию. Мы это сделаем для Info/README.txt

git reset Info/README.txt

затем пропишем git status что бы убедиться в результате.

*Разумеется для того что бы проверить выборочную разиндексацию нам нужно проиндексировать все изменения командой **git add -a** (для наглядности)*



```
MINGW32:/d/projects/MyApplications

'git <command> [<revision>...] -- [<file>...]'

tenom@DESKTOP-1J7659T MINGW32 /d/projects/MyApplications (master)
$ git reset Info/README.txt

tenom@DESKTOP-1J7659T MINGW32 /d/projects/MyApplications (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   Target.class

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

Info/
```

6) Фиксация изменений.

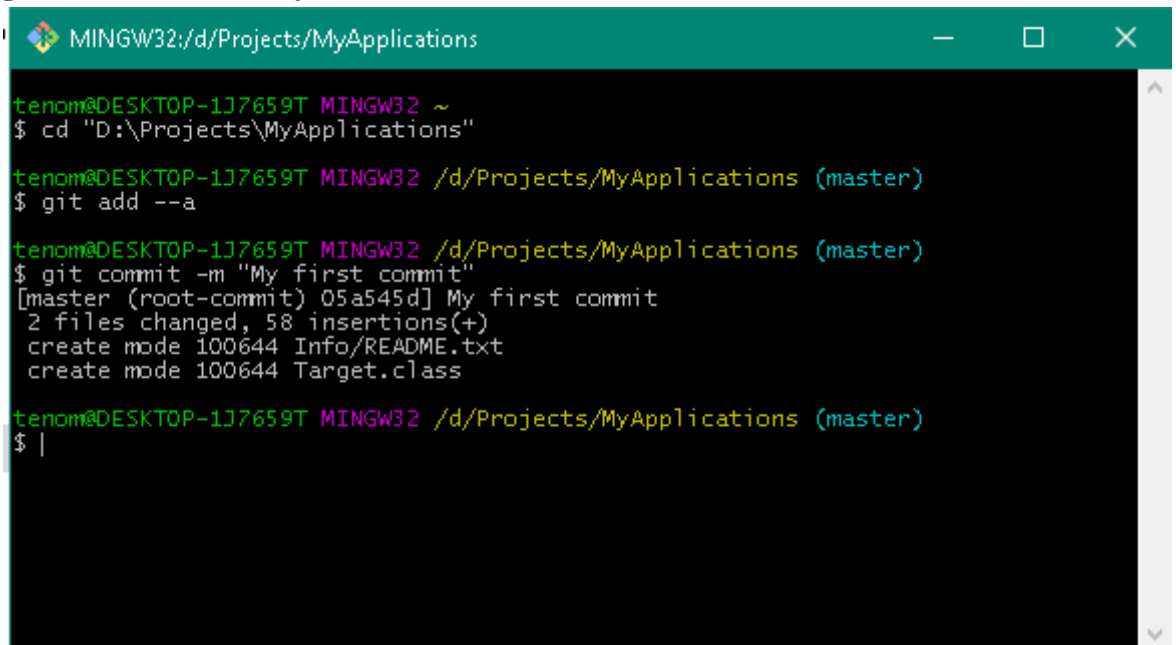
Предположим что мы внесли в проект какие то важные изменения и хотим их зафиксировать, что бы в случае каких то неполадок к ним вернутся. В нашей ситуации, когда мы только инициализировали проект, мы в любом случае должны зафиксировать изменения. В дальнейшем это нам позволит от чего то отталкиваться.

Перво очередно нам необходимо проиндексировать все что мы хотим зафиксировать.

git add --a

Для коммита используется команда **git commit**, в большинстве случаев ее перегружают **-m** (Предоставляющий возможность закоментировать commit). Листинг команды выглядит так:

git commit -m "My first commit"

A screenshot of a terminal window titled 'MINGW32:/d/Projects/MyApplications'. The terminal shows the following commands and output:

```
tenom@DESKTOP-1J7659T MINGW32 ~  
$ cd "D:\Projects\MyApplications"  
  
tenom@DESKTOP-1J7659T MINGW32 /d/Projects/MyApplications (master)  
$ git add --a  
  
tenom@DESKTOP-1J7659T MINGW32 /d/Projects/MyApplications (master)  
$ git commit -m "My first commit"  
[master (root-commit) 05a545d] My first commit  
2 files changed, 58 insertions(+)  
create mode 100644 Info/README.txt  
create mode 100644 Target.class  
  
tenom@DESKTOP-1J7659T MINGW32 /d/Projects/MyApplications (master)  
$ |
```

Отмечу что гит подробно описывает результат команды commit и дам краткий комментарий.

Зафиксированная ветка мастер, коммит корневой, хеш коммита, комментарий

[master (root-commit) 05a545d] My first commit

Было изменено каким то образом 2 файла.

2 files changed, 58 insertions(+)

Подробнее об изменениях: Созданы файлы *README.txt* и *Target.class*

create mode 100644 Info/README.txt

create mode 100644 Target.class

Как видно в листинге, гит кратко описал природу изменений и их содержимое, что удобно для проверки себя.

7) Откат коммита или как вернуться где работает

Система контроля версий предполагает возможность вернуться к предыдущему состоянию проекта, на тот случай если текущее состояние по каким то причинам нас не устраивает. Для этого есть команда

git reset --hard HEAD~x где x на сколько коммитов назад мы хотим откатиться.

git reset --hard HEAD~2

Она удаляет все внесенные изменения включая сам момент фиксирования.

Для тестирования команды сделаем несколько коммитов.

Для этого проверим текущее состояние проекта **git status**, если есть какие то не зафиксированные изменения - индексирем и коммитим, если нет, в корневой директории создадим текстовый файл *Test reset commit.txt*, проиндексирем и сделаем коммит с комментарием *commit two*.

После этого запишем в файл текст, снова аналогичный названию – содержимое не столь важно. На сей раз комментарий коммита будет *commit three*.

В итоге мы сделали еще 2 или более коммита для тестирования **reset**, а так же получили ценный опыт.

Прописываем команду

git reset --hard HEAD~2

A screenshot of a terminal window with a green title bar. The title bar text is "MINGW32:/d/Projects/MyApplications". The terminal content shows the prompt "tenom@DESKTOP-1J7659T MINGW32 /d/Projects/MyApplications (master)" followed by the command "\$ git reset --hard HEAD~2" and the output "HEAD is now at 05a545d My first commit".

```
MINGW32:/d/Projects/MyApplications
tenom@DESKTOP-1J7659T MINGW32 /d/Projects/MyApplications (master)
$ git reset --hard HEAD~2
HEAD is now at 05a545d My first commit
```

Так же можно сделать до какого то определенного коммита по **hash**

git reset --hard HEAD hash

hash можно взять в веб интерфейсе GitHub

Что бы добавить изменение в облако на GitHub, используется команда **push** с перегруженным оператором **-f**. Это говорит о том что мы делаем принудительный коммит.

git push -f origin master

Далее функцию **push** мы рассмотрим более подробно

Другая ситуация когда комит сделан не вовремя, есть еще одно изменение которое мы хотим добавить в проект. Мы можем удалить комит не меняя содержимого.

git reset --soft HEAD

Теперь нам просто заново нужно сделать ком мит.

В аналогичной ситуации можно просто обновить последний комит

git commit -a --amend

Все изменения будут учены в последнем комите.

клон

фьюч

пуш

ветки