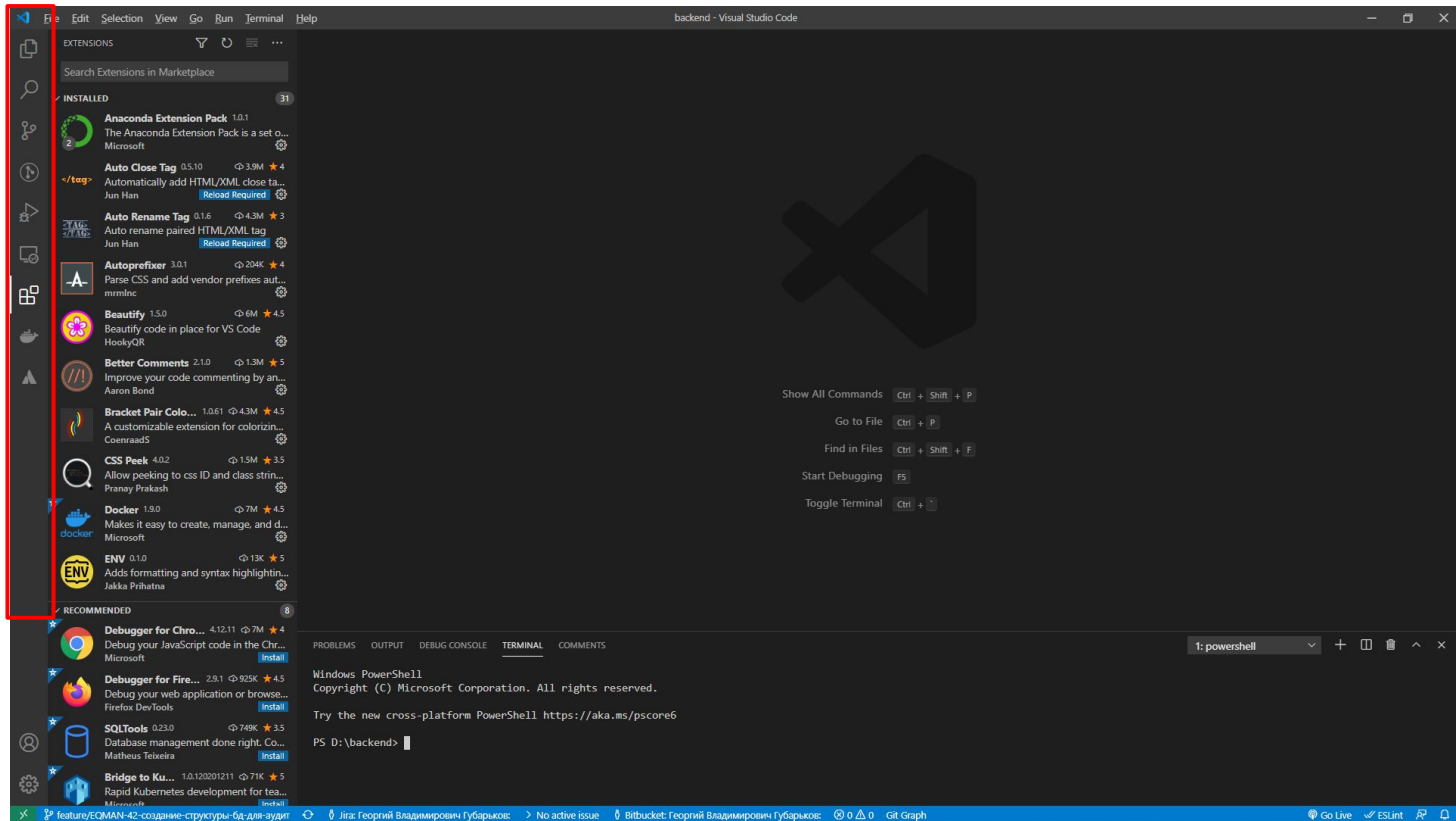


# Настройка ПО

VSCode

# Расширения



# Расширения

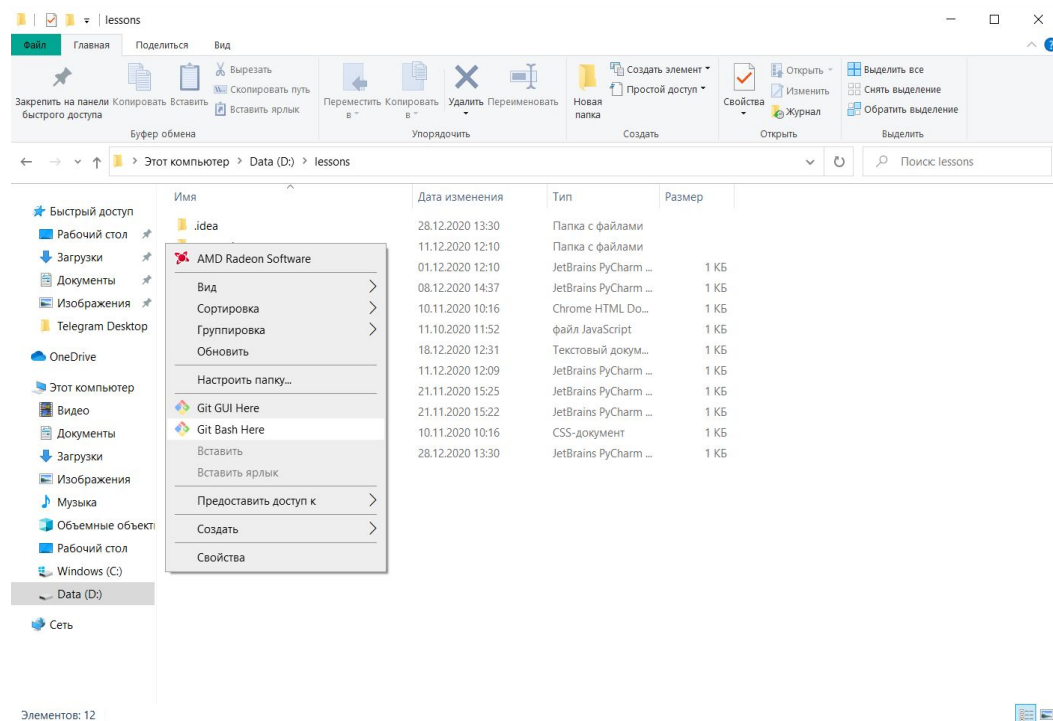
- Auto Close Tag - автоматически закрывает теги html
- Auto Rename Tag- если начать менять название тега, то данное расширение автоматически переименует связанный тег
- Beautify - автоматически исправляет код подстраивая его под “правила хорошего кода”
- Bracket Pair Colorizer - меняет цвет связанных скобок
- Git Graph - позволяет посмотреть историю версий git
- GitLens — Git supercharged - позволяет посмотреть какие изменения произошли в файлах
- HTML CSS Support - подсказывает как можно назвать селекторы
- Live Server - запускает браузер в котором происходят изменения в онлайн
- Path Autocomplete - автозаполнение путей

Проверка git

# Git Bash

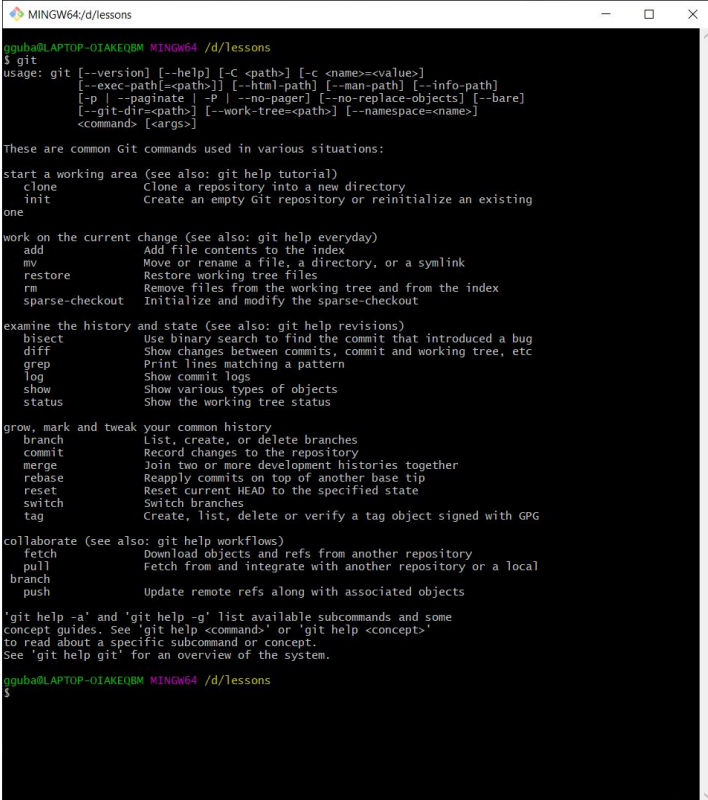
Для того чтобы пользоваться git в Windows - необходимо кликнуть ПКМ в папке в которой вы собираетесь работать и выбрать: "Git Bash Here". Или в cmd перейти в нужную папку.

Для обладателей Unix подобных систем - необходимо в terminal переместиться в директорию в которой собираетесь работать



# Первая команда

Для проверки работы попробуйте ввести git. На консоле появится информация о командах доступных для данной утилиты.



```
MINGW64/d/lessons
aguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons
$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path<= <path>]] [--html-path] [--man-path] [--info-path]
      [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing
            one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index
  sparse-checkout  Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  diff       Show changes between commits, commit and working tree, etc
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status

grow, mark and tweak your common history
  branch     List, create, or delete branches
  commit     Record changes to the repository
  merge      Join two or more development histories together
  rebase     Reapply commits on top of another base tip
  reset      Reset current HEAD to the specified state
  switch     Switch branches
  tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch      Download objects and refs from another repository
  pull       Fetch from and integrate with another repository or a local
            branch
  push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.

aguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons
$
```

# Помощь

Если вам нужна помощь при использовании Git, есть три способа открыть страницу руководства по любой команде Git:

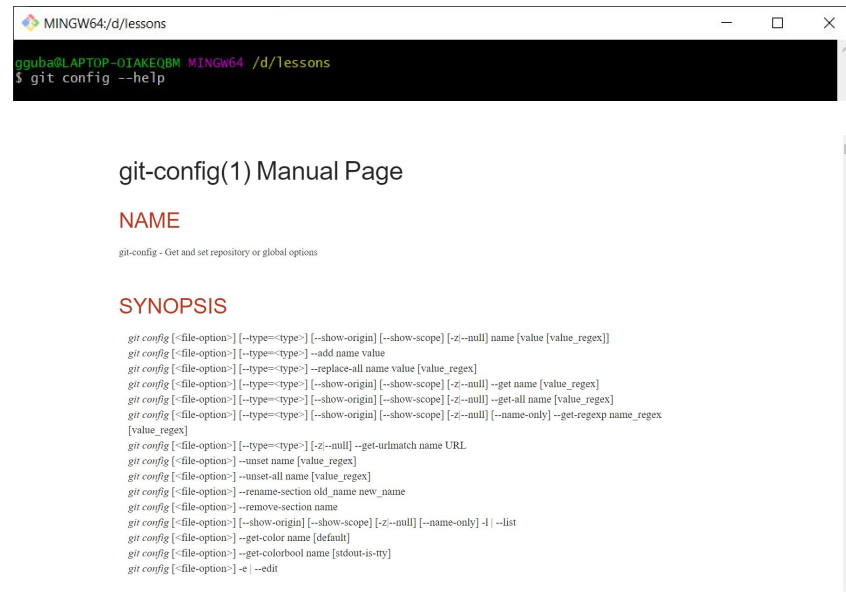
\$ git help <команда>

\$ git <команда> --help

\$ man git-<команда>

Например, так можно открыть руководство по команде git config

\$ git config --help

A screenshot of a terminal window titled 'MINGW64/d/lessons'. The prompt is 'jguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons'. The command '\$ git config --help' has been executed, and the output is the manual page for 'git-config(1)'. The output includes the title 'git-config(1) Manual Page', a description 'git-config - Get and set repository or global options', a synopsis section, and a list of command-line options with their descriptions.

```
MINGW64/d/lessons
jguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons
$ git config --help

git-config(1) Manual Page

NAME
git-config - Get and set repository or global options

SYNOPSIS
git config [-file-option] [--type=<type>] [--show-origin] [--show-scope] [-z|--null] name [value [value_regex]]
git config [-file-option] [--type=<type>] --add name value
git config [-file-option] [--type=<type>] --replace-all name value [value_regex]
git config [-file-option] [--type=<type>] [--show-origin] [--show-scope] [-z|--null] --get name [value_regex]
git config [-file-option] [--type=<type>] [--show-origin] [--show-scope] [-z|--null] --get-all name [value_regex]
git config [-file-option] [--type=<type>] [--show-origin] [--show-scope] [-z|--null] --get-regexp name_regex
[value_regex]
git config [-file-option] [--type=<type>] [-z|--null] --get-urlmatch name URL
git config [-file-option] --unset name [value_regex]
git config [-file-option] --unset-all name [value_regex]
git config [-file-option] --rename-section old_name new_name
git config [-file-option] --remove-section name
git config [-file-option] [--show-origin] [--show-scope] [-z|--null] [--name-only] -l --list
git config [-file-option] --get-color name [default]
git config [-file-option] --get-colorbool name [stdout-is-ty]
git config [-file-option] -e|--edit
```



# Помощь по флагам для команд

Чтобы не читать всю страницу с мануалом по команде, а узнать только информацию по доступным флагам необходимо ввести:

`git <command>`

Например, чтобы узнать доступные флаги и их описание для команды `git config`, необходимо ввести:

`$git config`

```
gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons
$ git config
usage: git config [<options>]

Config file location
--global          use global config file
--system          use system config file
--local           use repository config file
--worktree        use per-worktree config file
-f, --file <file> use given config file
--blob <blob-id>  read config from given blob object

Action
--get             get value: name [value-regex]
--get-all        get all values: key [value-regex]
--get-regexp      get values for regexp: name-regex [value-regex]
--get-urlmatch    get value specific for the URL: section[.var] URL
--replace-all    replace all matching variables: name value [value-regex]
--add             add a new variable: name value
--unset           remove a variable: name [value-regex]
--unset-all      remove all matches: name [value-regex]
--rename-section  rename section: old-name new-name
--remove-section  remove a section: name
-l, --list        list all
-e, --edit        open an editor
--get-color       find the color configured: slot [default]
--get-colorbool   find the color setting: slot [stdout-is-atty]

Type
-t, --type <>    value is given this type
--bool           value is "true" or "false"
--int            value is decimal number
--bool-or-int    value is --bool or --int
--path           value is a path (file or directory name)
--expiry-date    value is an expiry date

Other
-z, --null       terminate values with NUL byte
--name-only      show variable names only
--includes       respect include directives on lookup
--show-origin    show origin of config (file, standard input, blob, command line)
--show-scope     show scope of config (worktree, local, global, system, command)
--default <value> with --get, use default value when missing entry

gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons
$
```

# ОСНОВЫ git

Создание репозитория

# Репозиторий

**Репозиторий Git** представляет собой каталог файловой системы, в котором находятся файлы конфигурации **репозитория**, файлы журналов, хранящие операции, выполняемые над **репозиторием**, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы.

Проще говоря репозиторий - это папка в которой инициализирован git (есть папка .git).

# Как создать репозиторий

Для создания репозитория есть несколько вариантов:

1. Инициализировать git в рабочей директории
2. Клонировать уже созданный репозиторий

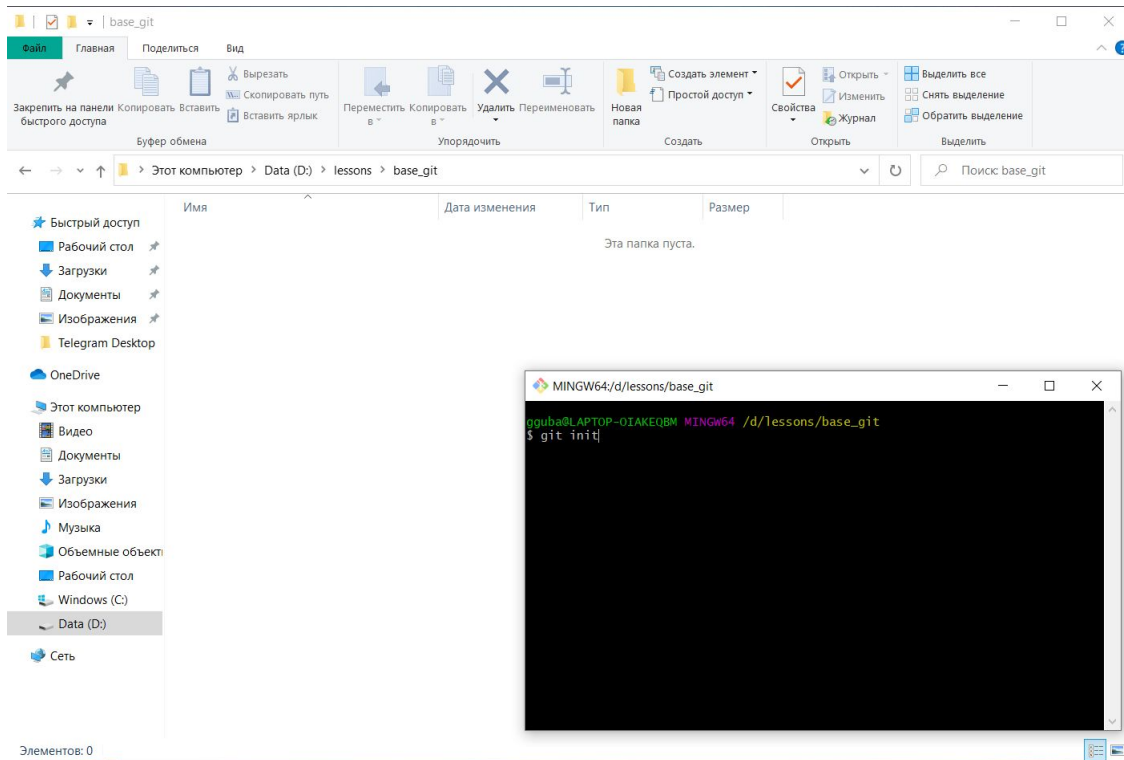
*Далее все команды выполняются в Git Bash. Если вы обладатели ОС Linux/Mac, то вы работаете в terminal.*

# Инициализация репозитория

Для инициализации репозитория (даже если у вас нечего отслеживать) необходимо в рабочей директории открыть Git Bash для обладателей Windows (либо переместиться в директорию в terminal для linux/mac) и ввести команду:

```
$ git init
```

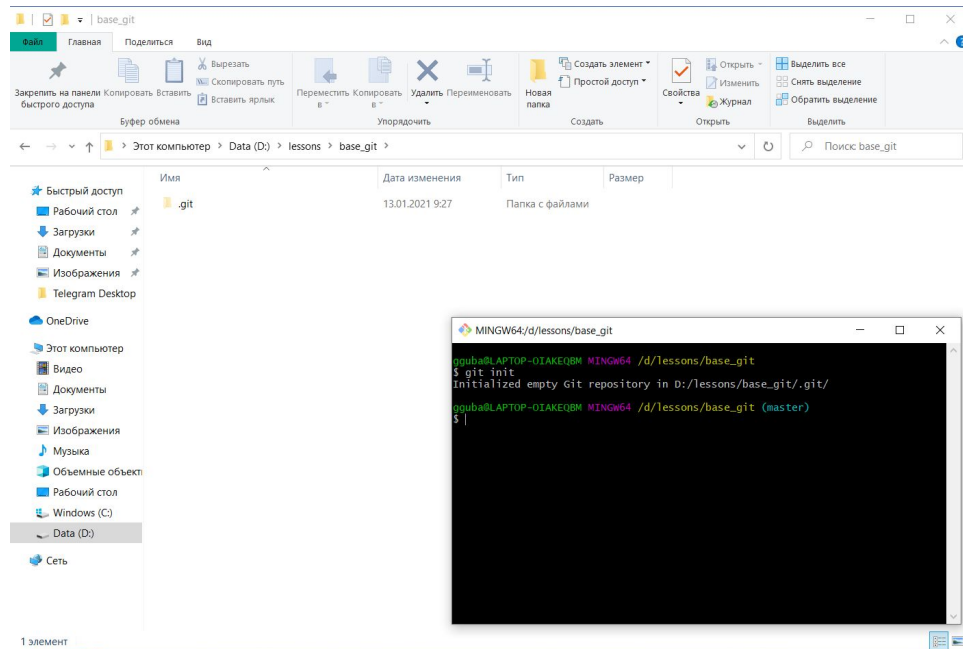
*Все команды обязательно должны выполняться в папке с инициализированным git. Или в его подпапках.*



# Инициализация репозитория

Если всё сделали правильно, то в директории у вас должна была появиться скрытая папка `.git` и чуть-чуть поменяться интерфейс `git bash`.

Теперь всё что находится в директории отслеживается `git`.



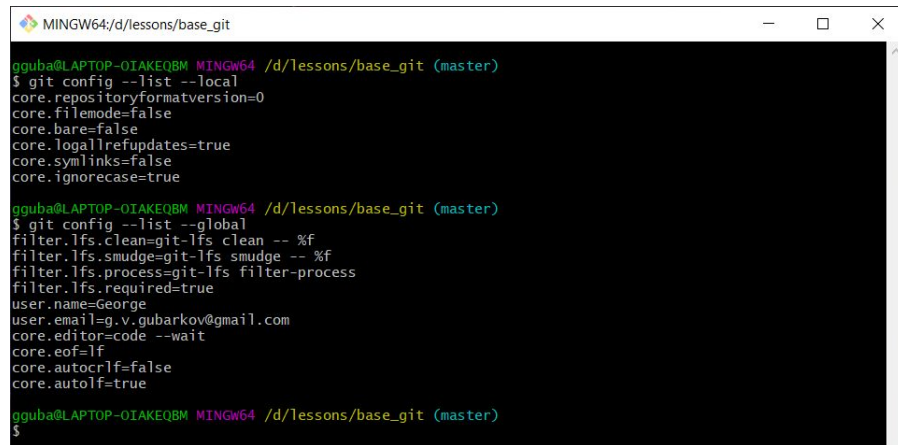
# Просмотр конфига git

Прежде чем начать работать с git, необходимо указать информацию о себе, чтобы если вы разрабатываете в команде вас могли найти и побить.

Настройки можно задать локальные и глобальные. Если вы задаёте настройки локально, то они распространяются только на текущий репозиторий. Глобальные же распространяются на все репозитории на вашем устройстве. Локальные настройки более приоритетны.

Для того чтобы посмотреть информацию о себе необходимо использовать команду (флаги --local & --global используются для глобальных и локальных конфигов):

```
$ git config --list
```



```
MINGW64/d/lessons/base_git
gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git config --list --local
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true

gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git config --list --global
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=George
user.email=g.v.gubarkov@gmail.com
core.editor=code --wait
core.eof=lf
core.autocrlf=false
core.autolf=true

gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$
```

# Изменение конфига

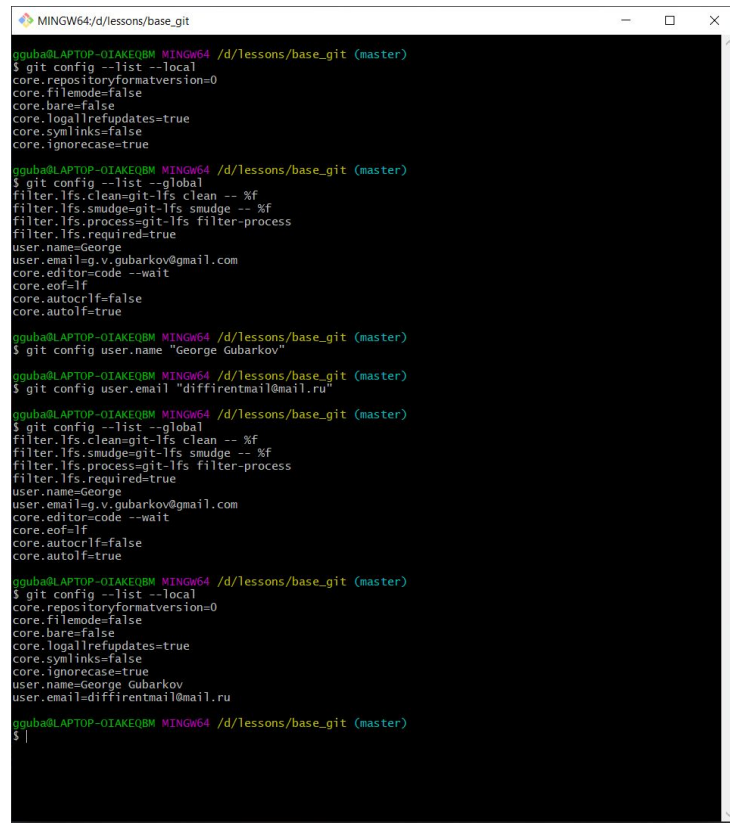
Для того чтобы изменить что-либо в конфиге - необходимо использовать следующий синтаксис (можно также использовать ключи `--local` & `--global`):

*git config <параметр который нужно изменить> <значение>*

Например:

`git config <--global/--local> user.name <name>`

`git config <--global/--local> user.email <email>`



```
MINGW64/d/lessons/base_git
guba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git config --list --local
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symbols=false
core.ignorecase=true

guba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git config --list --global
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=George
user.email=g.v.gubarkov@gmail.com
core.editor=code --wait
core.eol=lf
core.autocrlf=false
core.autolfs=true

guba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git config user.name "George Gubarkov"

guba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git config user.email "diffirentmail@mail.ru"

guba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git config --list --global
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=George
user.email=g.v.gubarkov@gmail.com
core.editor=code --wait
core.eol=lf
core.autocrlf=false
core.autolfs=true

guba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git config --list --local
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symbols=false
core.ignorecase=true
user.name=George Gubarkov
user.email=diffirentmail@mail.ru

guba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$
```

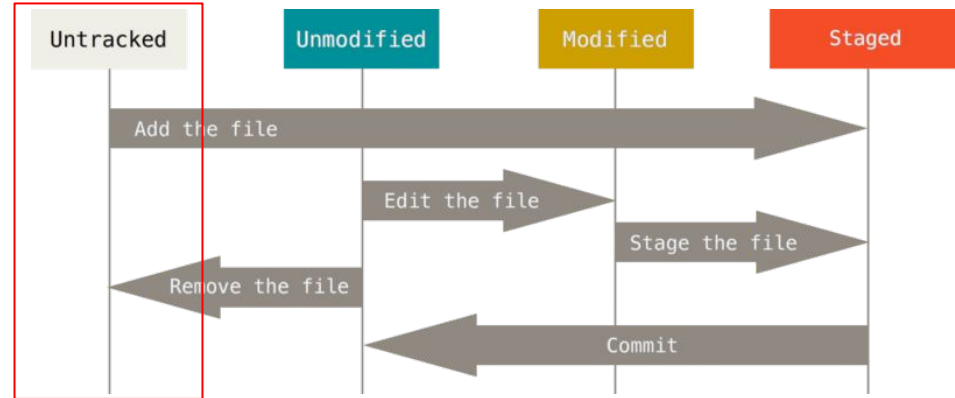


# ОСНОВЫ git

Статус

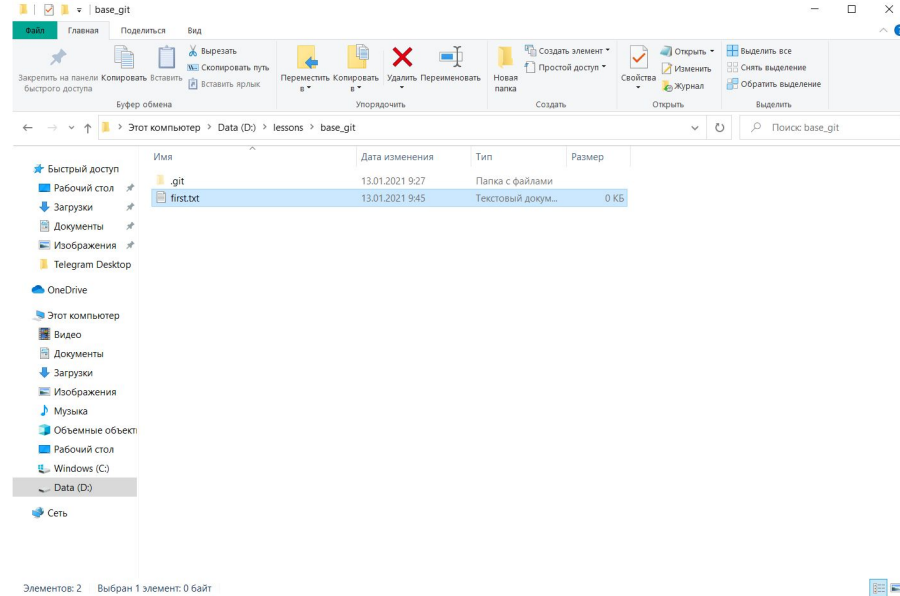
# Неотслеживаемые файлы

У всего что находится в репозитории (поддиректории, файлы) есть несколько состояний. Прежде всего отслеживаемые и неотслеживаемые. Неотслеживаемые файлы - это такие файлы, которые не были в последнем “снимке” git.



# Неотслеживаемые файлы

Создадим новый файл в нашем репозитории. Это вы можете сделать любым удобным способом. Через терминал или же через графический интерфейс.

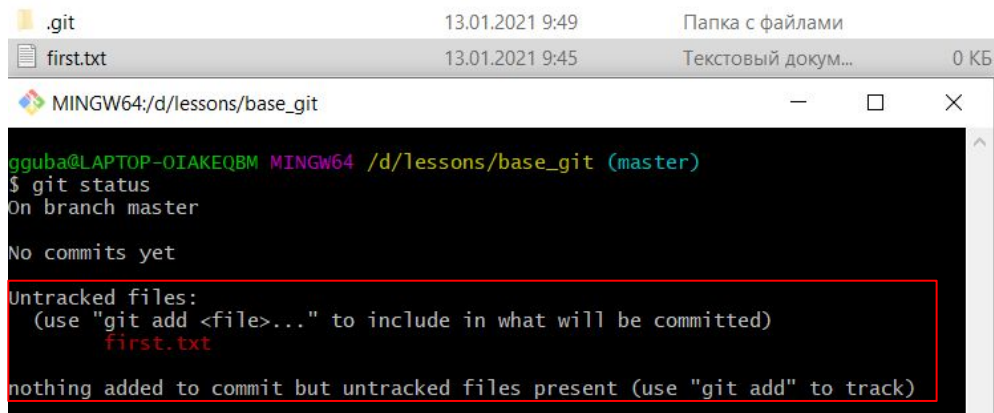


# Состояние файла - неотслеживаемый

Теперь проверим состояние нового файла. Для проверки состояний файлов, папок используется команда:

`$git status`

Для того чтобы начать его отслеживать, его необходимо проиндексировать.



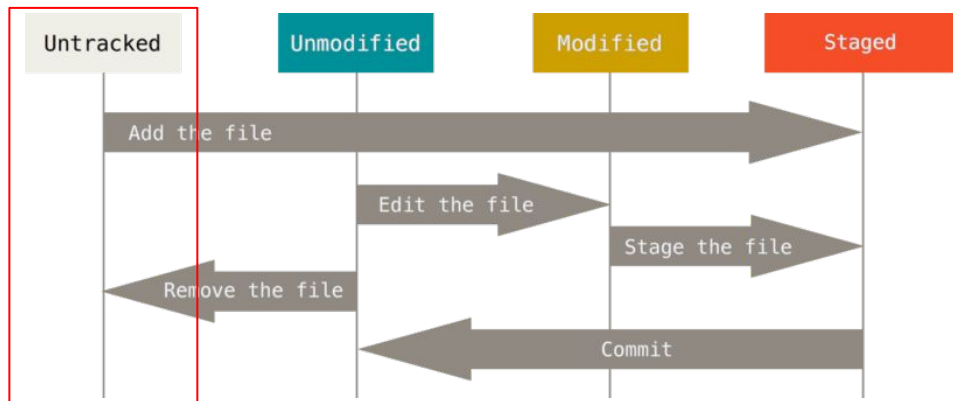
```
.git 13.01.2021 9:49 Папка с файлами
first.txt 13.01.2021 9:45 Текстовый докум... 0 КБ
MINGW64:/d/lessons/base_git

gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git status
On branch master

No commits yet

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

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

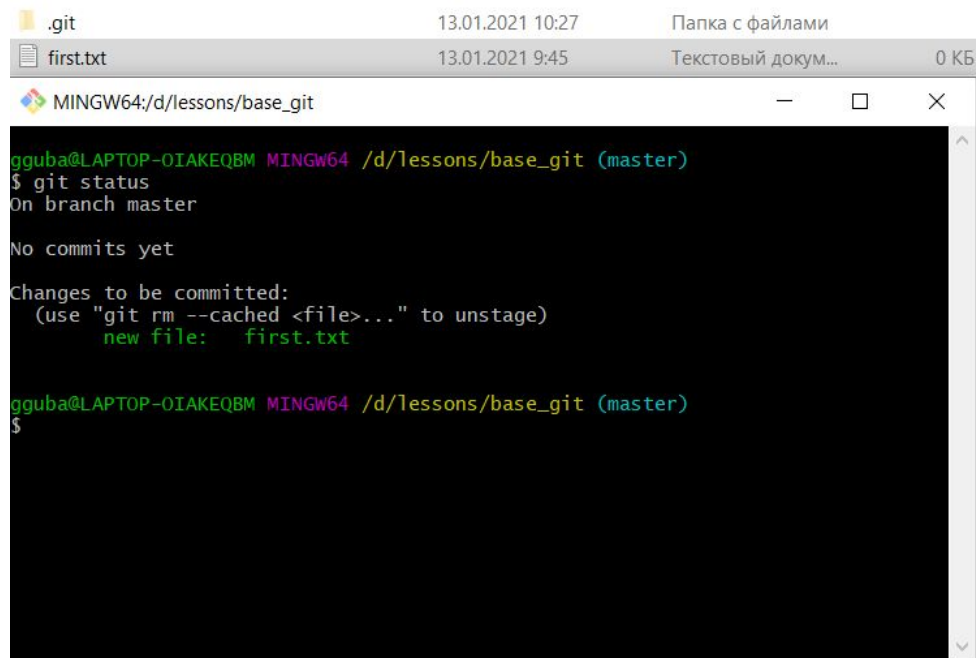


# Отслеживание файлов

Для того чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда:

```
$git add <filename>
```

Если *после этого проверить статус файлов*, то вы увидите, что файл проиндексирован, так как он находится в секции “Changes to be committed”. Как вы помните, когда вы ранее выполнили `git init`, затем вы выполнили `git add` (файлы) — это было сделано для того, чтобы добавить файлы в ваш каталог под версионный контроль. Команда `git add` принимает параметром путь к файлу или каталогу, если это каталог, команда рекурсивно добавляет все файлы из указанного каталога в индекс.



The image shows a Windows File Explorer window at the top with the following details:

Имя	Дата и время	Тип	Размер
.git	13.01.2021 10:27	Папка с файлами	
first.txt	13.01.2021 9:45	Текстовый докум...	0 КБ

Below the File Explorer is a terminal window titled "MINGW64; d/lessons/base\_git". The terminal output is as follows:

```
gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git status
On branch master

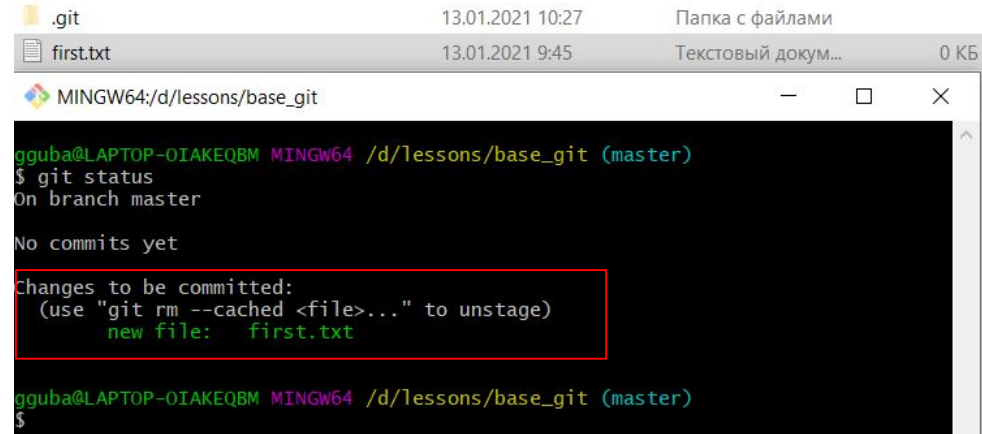
No commits yet

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

gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$
```

# Отслеживаемые файлы

Как только вы проиндексировали файл. Он переместился в Staged и готов к снимку.



The image shows a Windows Explorer window at the top with a file named `first.txt` in a folder named `.git`. Below it is a terminal window titled `MINGW64:/d/lessons/base_git`. The terminal shows the output of the `git status` command:

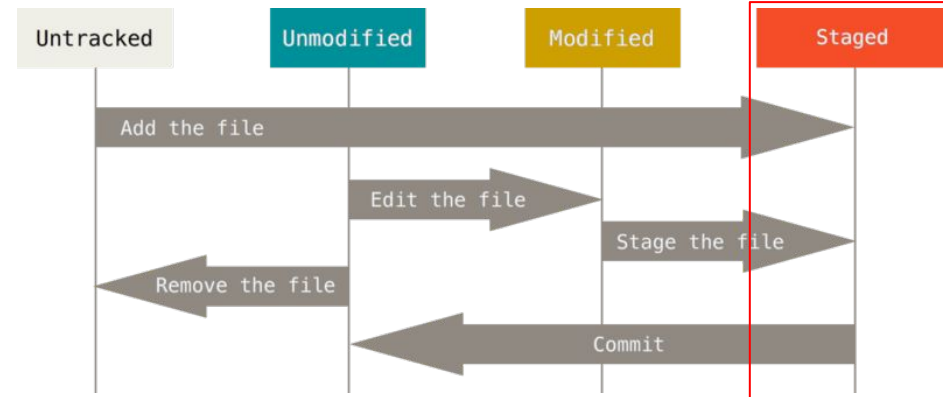
```
gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git status
On branch master

No commits yet

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

gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$
```

A red rectangle highlights the "Changes to be committed" section in the terminal output.

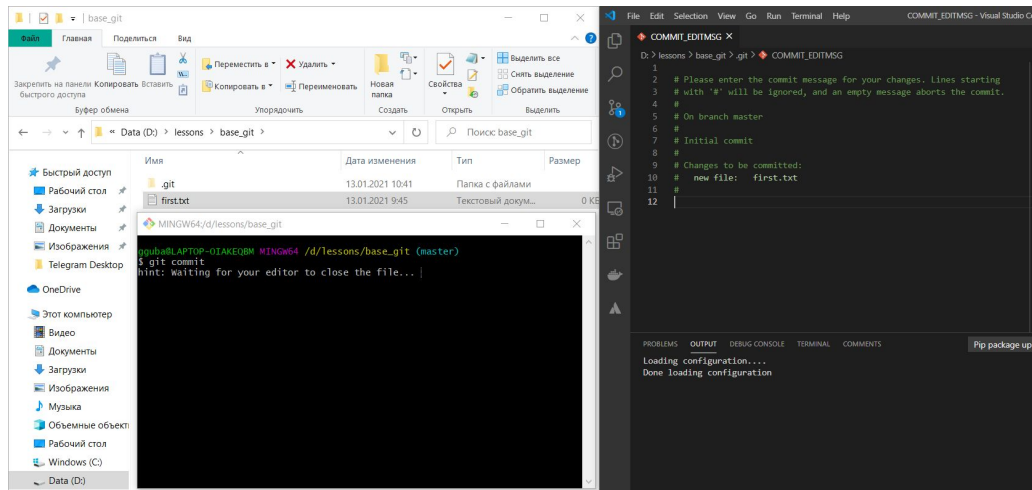


# Мой первый коммит

Теперь, когда ваш индекс находится в таком состоянии, как вам и хотелось, вы можете зафиксировать свои изменения (сделать снимок состояния файлов и папок). Это означает сделать коммит. Коммиты делаются с обязательным сообщением. Сообщение должно отражать проделанную работу. Для того чтобы сделать коммит введите:

```
$ git commit
```

После этой команды у вас откроется какой-то текстовый редактор (в моём случае это vscode). Там будет предложено ввести сообщение для коммита. А также написаны файлы которые подлежат коммиту. Знак # означает комментарий. Если вы по какой-то причине передумали делать коммит, то нажмите ctrl+c в Git Bash. Или выйдите без сохранения из редактора текста.



# Мой первый коммит

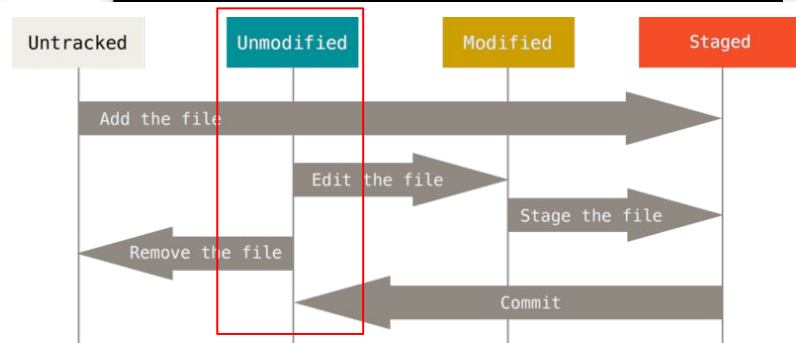
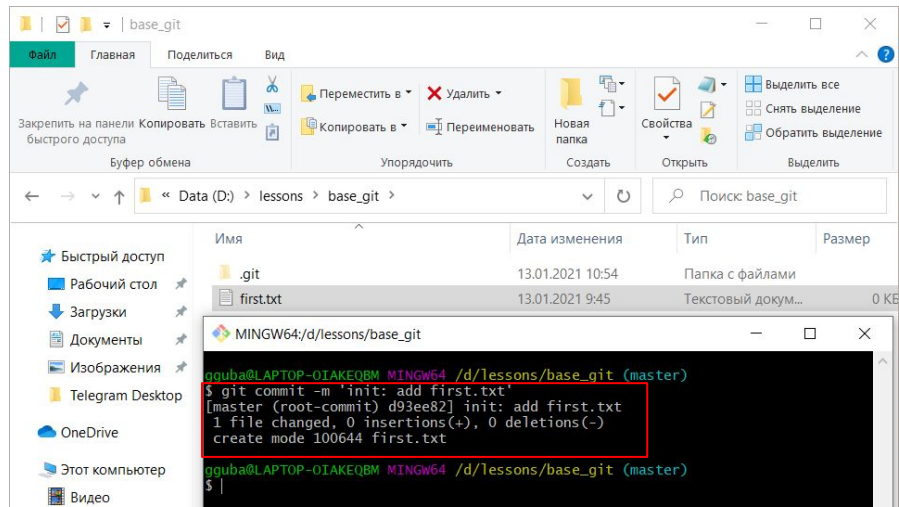
В большинстве случаев можно кратко написать сообщение об изменениях в коммите. Для того чтобы задать краткое сообщение в коммите необходимо ввести команду:

```
$ git commit -m 'message'
```

Если нет никаких конфликтов, то у вас создастся новый коммит.

Теперь посмотрите статус вашего репозитория

Поздравляю с вашим первым  
КОММИТОМ



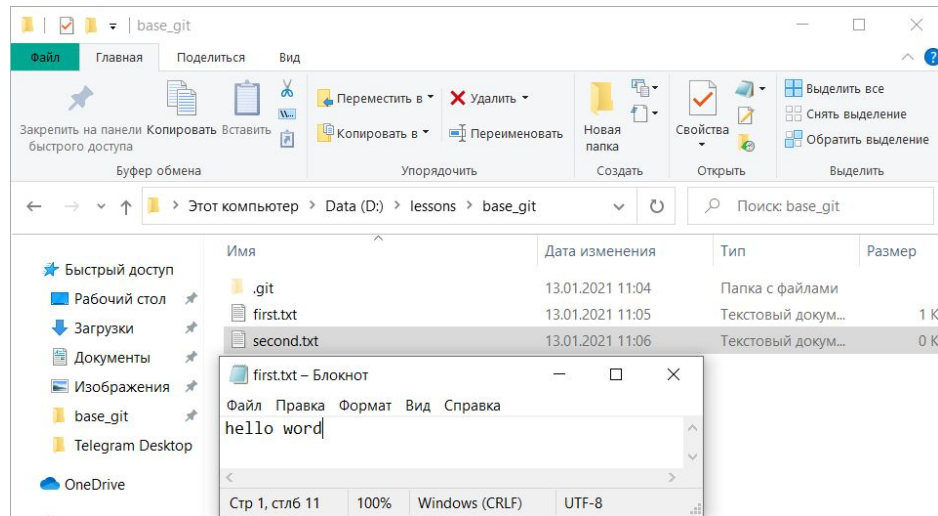


# Добавление для индексации нескольких файлов

Чаще всего в проектах меняется множество файлов прежде, чем их проиндексируют и добавят в коммит. Создайте ещё один файл в вашем репозитории и измените существующий.

Проверьте статус репозитория.

`$ git status`



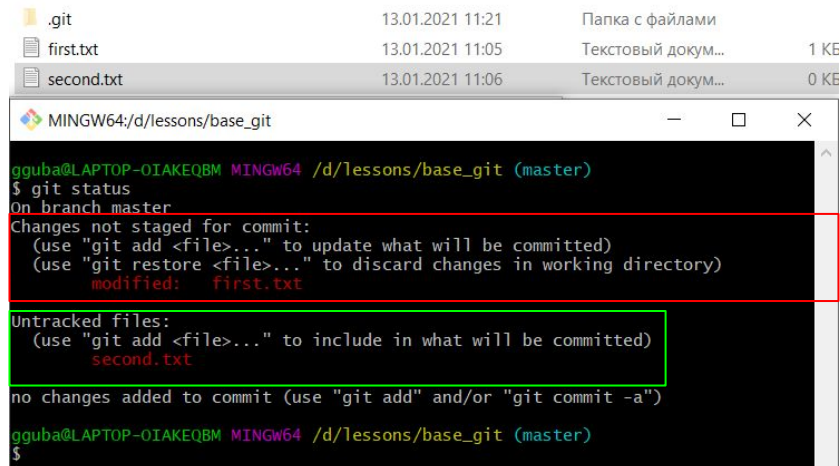
# Добавление для индексации нескольких файлов

Теперь есть файл который не *отслеживается* и *изменённый*.

Теперь эти файлы необходимо проиндексировать. Это можно сделать следующими способами:

1. Последовательно прописать названия файлов в команде `git add <filename> <filename2>...`
2. Либо, если вы хотите проиндексировать все изменения - можно ввести `git add .` или `git add *`

*Попробуйте самостоятельно git add .*



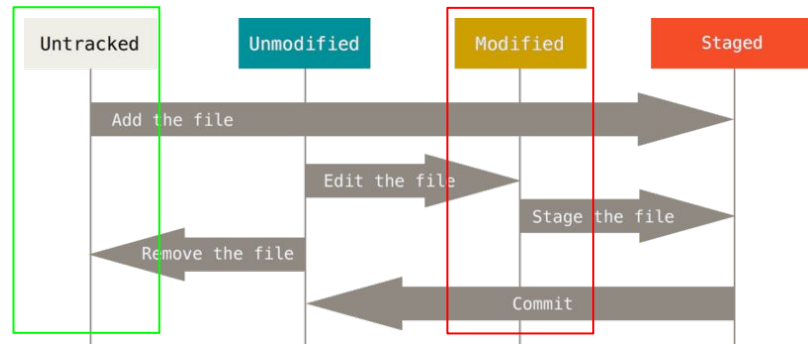
```

.
├── .git
├── first.txt
└── second.txt
13.01.2021 11:21 Папка с файлами
13.01.2021 11:05 Текстовый докум... 1 KB
13.01.2021 11:06 Текстовый докум... 0 KB

MINGW64/d/lessons/base_git
gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   first.txt

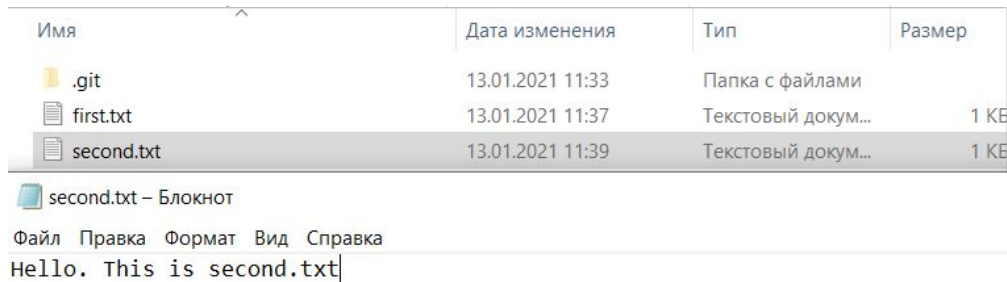
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        second.txt

no changes added to commit (use "git add" and/or "git commit -a")
gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$
```



# Индексация и коммит

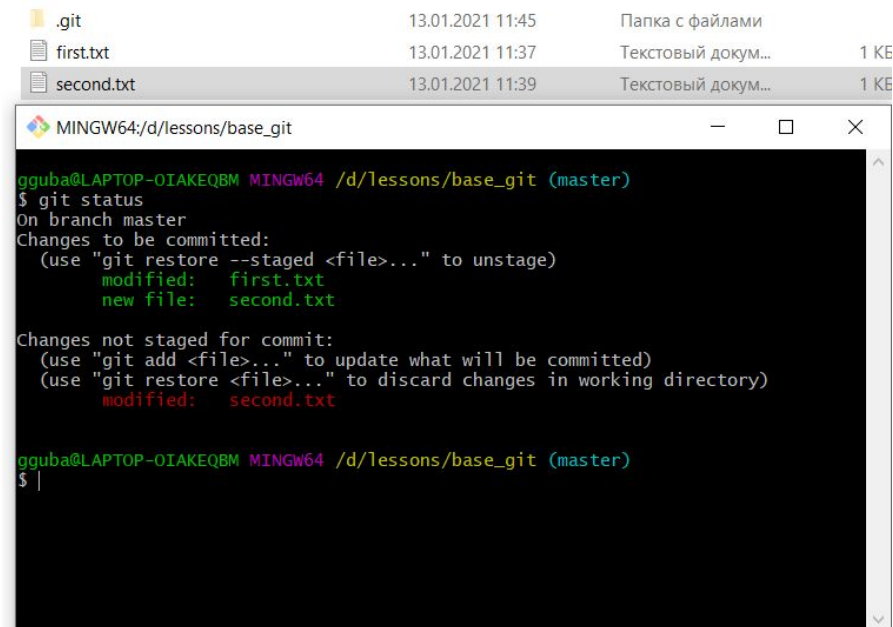
Если вы всё сделали правильно, то ваши изменения должны были проиндексироваться и теперь они готовы к коммиту. Но прежде, измените содержимое любого файла. И проверьте состояние репозитория.



# Индексация и коммит

Теперь файл `second.txt` находится в двух состояниях. Он одновременно проиндексирован и нет. Если выполнить коммит, то в него попадёт пустой файл `second.txt` и если вы будете загружать на хостинг ваш репозиторий, то туда попадет именно пустой файл. Вы можете снова добавить этот файл в индексацию. Никаких ошибок не произойдёт.

Проиндексируйте файл `second.txt` и выполните коммит.

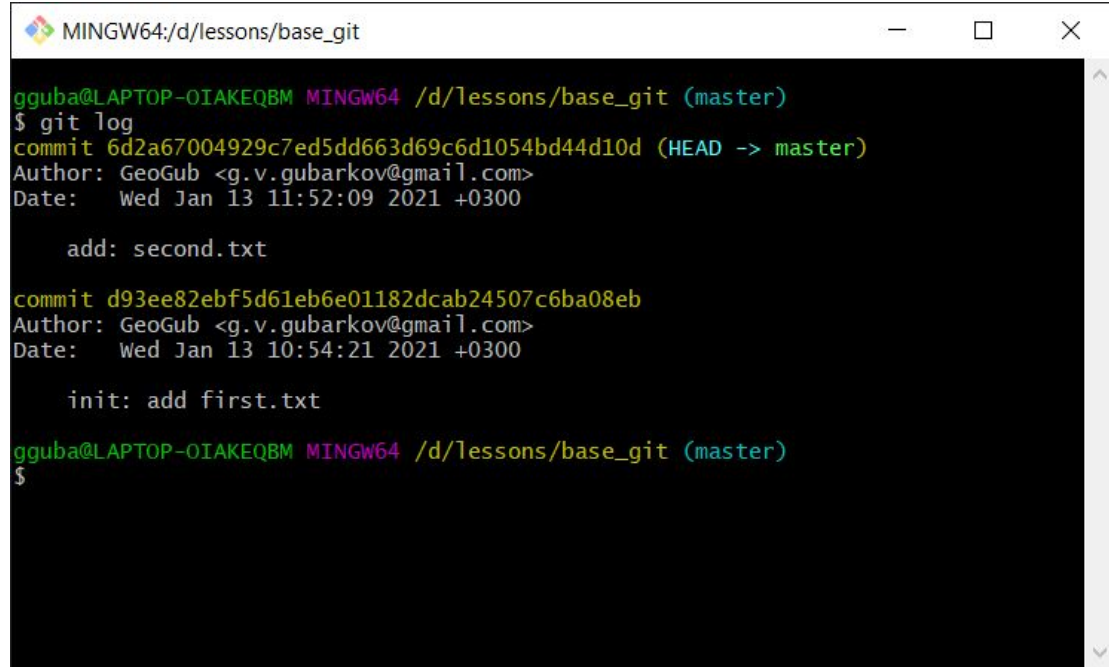


The image shows a file explorer window at the top and a terminal window below it. The file explorer shows a directory with files `.git`, `first.txt`, and `second.txt`. The terminal window shows the output of the `git status` command, indicating that `first.txt` is modified and `second.txt` is a new file, both staged for commit.

```
MINGW64/d/lessons/base_git  
gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)  
$ git status  
On branch master  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    modified:   first.txt  
    new file:   second.txt  
  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
    modified:   second.txt  
  
gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)  
$ |
```

# История коммитов

Для просмотра истории коммитов можно использовать команду `git log`. После ввода данной команды у вас появится информация о сделанных коммитах. Кто их сделал, когда, какое сообщение оставил и одно из самых важных полей - номер коммита. Длинная строка из символов. Благодаря ей можно откатить репозиторий к любому предыдущему коммиту.



```
MINGW64:/d/lessons/base_git

gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$ git log
commit 6d2a67004929c7ed5dd663d69c6d1054bd44d10d (HEAD -> master)
Author: GeoGub <g.v.gubarkov@gmail.com>
Date:   Wed Jan 13 11:52:09 2021 +0300

    add: second.txt

commit d93ee82ebf5d61eb6e01182dcab24507c6ba08eb
Author: GeoGub <g.v.gubarkov@gmail.com>
Date:   Wed Jan 13 10:54:21 2021 +0300

    init: add first.txt

gguba@LAPTOP-OIAKEQBM MINGW64 /d/lessons/base_git (master)
$
```