

BCP 2.3 №4

Git – система контроля версий, используемая разработчиками по всему миру, и в рамках данного задания будет рассказано об основных принципах работы с Git через терминал. Сам по себе Git без графического интерфейса уже можно использовать как систему для контроля версий и стоит понимать, что Git — это инструмент, а GitHub — конкретная платформа, часто используемая в связке с Git. В рамках задания все команды будут выполняться локально, но для наглядности все изменения будут публиковаться в репозиториях на GitHub.

Прежде чем выполнять все действия, можно убедиться что Git вообще установлен: поможет нам в этом команда

```
git --version
```

```
fedorcagin@MacBook-Pro13 ~ % git --version
git version 2.39.5 (Apple Git-154)
```

(ну и если его нет, то надо скачать его с официального сайта или воспользовавшись нужной командой, в зависимости от ОС). Также возможно, что командами

```
git config --global user.name «ВашеИмя» и
git config --global user.email "ВашаПочта@example.com"
```

понадобится настроить ваше имя и почту, но в данном случае всё уже готово.

На этом этапе стоит проверить самое важное — подключение к GitHub. Команда

```
ssh -T git@github.com
```

```
fedorcagin@MacBook-Pro13 ~ % ssh -T git@github.com
Hi Fedor-Chagin! You've successfully authenticated, but GitHub does not provide shell access.
```

не выполняет никаких действий с репозиториями, а просто проверяет, узнаёт ли GitHub мой компьютер по секретному SSH-ключу. Если настроить Git на своём сервере, то можно отключить это – но GitHub и многие другие аналогичные сервисы не дают сбросить эти меры безопасности. Если SSH ключа нет – то его можно легко добавить или использовать менее безопасные токены. Однако это отдельная тема с подробно расписанными инструкциями на GitHub: в данном случае можно увидеть надпись «Hi Fedor-Chagin! You've successfully authenticated, but GitHub does not provide shell access.» это значит что здесь уже всё настроено, и можно приступать к непосредственной работе с проектом. Для этого создадим папку проекта и перейдём в неё:

```
mkdir git-demo_for_task
cd git-demo_for_task
```

```
fedorcagin@MacBook-Pro13 ~ % mkdir git-demo_for_task
fedorcagin@MacBook-Pro13 ~ % cd git-demo_for_task
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

Теперь, когда мы внутри папки нашего проекта, самое время превратить её в репозиторий, за которым будет следить Git. Для этого существует команда

```
git init
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git init
Initialized empty Git repository in /Users/fedorcagin/git-demo_for_task/.git/
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

Это значит, что Git создал скрытую служебную папку .git — именно в ней будет храниться вся история изменений, информация о коммитах и настройки именно для этого проекта. Теперь наша папка — не просто папка, а полноценный репозиторий. Можно сразу проверить, что нам об этом говорит Git –

```
git status
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

показывает текущее состояние нашего репозитория: какие файлы изменены, какие готовы к сохранению, а какие ещё не отслеживаются. В данном случае мы находимся на ветке main (или master в некоторых старых настройках), и видим что здесь не было ни одного коммита. А главное — ей просто нечего нам показывать в плане изменений, потому что у нас даже нет ни одного файла. Исправим это и создадим самый главный файл любого проекта — README.md.

```
echo "# какой-то текст" >> README.md
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % echo "# какой-то текст" >> README.md
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

Командой echo создаётся текстовая строка, а стрелочками она направляется прямо в указанный файл. Теперь, если снова проверить статус, то увидим, что Git обнаружил новый файл и сообщает, что он «untracked» — это значит, что за ним ещё не ведётся наблюдение (т.е. Git видит этот файл в папке, но целенаправленно игнорирует его и не сохраняет его историю изменений). Чтобы Git начал его отслеживать, нужно специально добавить его в так называемый «индекс» или «staging area» — промежуточную область, где собираются изменения для следующего сохранения. Делается это командой

```
git add README.md
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git add README.md
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

Снова проверим статус:

```
git status
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git status
On branch main

No commits yet

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

fedorcagin@MacBook-Pro13 git-demo_for_task %
```

Теперь Git сообщает, что есть новый файл, и он «staged» — то есть готов к сохранению. Он уже в индексе и ждёт своего момента. Это отличие между просто файлом в папке и файлом, готовым к коммиту — одна из ключевых идей Git. Теперь, когда изменения готовы и лежат в индексе, самое время сделать их «снимок» — т.е. зафиксировать текущее состояние проекта. Правильно это называется «сделать коммит». Для этого используется команда

```
git commit
```

Но просто сделать коммит — мало. К нему нужно обязательно добавить комментарий, который объясняет, что сохраняется в этом коммите. Это в интересах человека, работающего с Git и в интересах разработчиков, с которыми он работает.

Поэтому припишем флаг -m и добавим комментарий. В результате команда будет выглядеть так:

```
git commit -m "Добавлен файл README"
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git commit -m "Добавлен файл README"
[main (root-commit) 2a105e2] Добавлен файл README
1 file changed, 1 insertion(+)
create mode 100644 README.md
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

После выполнения команды Git сообщит, что он сделал. В данном случае: «[main (root-commit) 2a105e2] Добавлен файл README». Это значит, что был создан самый первый (root) коммит в ветке main с уникальным идентификатором 2a105e2 и введённым сообщением.

Если на этом этапе ввести

```
git status,
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git status
On branch main
nothing to commit, working tree clean
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

то Git снова скажет «нечего коммитить» — потому что все изменения, которые были (новый файл), уже успешно был закоммичен. Рабочая папка «чистая». Для наглядности можно повторить последние несколько действий и сделать ещё один коммит: для примера можно добавить файл example.c (перемещён вручную) в папку проекта и сохраним эти изменения – всё так же в ветку main. Для этого нужно:

```
git add example.c
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git add example.c
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

проверить статус

```
git status
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   example.c

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

fedorcagin@MacBook-Pro13 git-demo_for_task %
```

и закоммитить

```
git commit -m "Добавлен файл .c»
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git commit -m "Добавлен файл .c"
[main 67eff1a] Добавлен файл .c
1 file changed, 7 insertions(+)
create mode 100644 example.c
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

А чтобы полюбоваться на свежую историю, можно использовать команду:

```
git log --oneline
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git log --oneline
67eff1a (HEAD -> main) Добавлен файл .c
2a105e2 Добавлен файл README
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

будет показана краткая история коммитов в обратном порядке (т.е. сверху, как видно — самый свежий).

Таким образом был создан готовый локальный репозиторий с историей коммитов: но вся эта история пока что существует только на конкретном компьютере, где выполнялись эти команды. Чтобы поделиться ей с коллегами или просто иметь backup в облаке, нужно запустить эти изменения – эта возможность предусмотрена в Git. Вместе с Git часто используется GitHub, ресурсами которого и будет продемонстрирована и возможность отправки таких изменений на сервера. Для этого понадобится пустой репозиторий на GitHub:

Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

1 General

Owner *



Repository name *

demo for task

✓ Your new repository will be created as demo-for-task.

The repository name can only contain ASCII letters, digits, and the characters ., -, and _.

Great repository names are short and memorable. How about **glowing-octo-meme**?

Description

0 / 350 characters

2 Configuration

Choose visibility *

Choose who can see and commit to this repository

Private

Add README

READMEs can be used as longer descriptions. [About READMEs](#)

Off

Add .gitignore

.gitignore tells git which files not to track. [About ignoring files](#)

No .gitignore

Add license

Licenses explain how others can use your code. [About licenses](#)

No license

Create repository

The screenshot shows the GitHub repository page for 'demo-for-task'. The repository is private and owned by 'Fedor-Chagin'. The page displays the repository name, visibility (Private), and a 'Create repository' button. Below the repository name, there are sections for 'Set up GitHub Copilot', 'Add collaborators to this repository', and 'Quick setup — if you've done this kind of thing before'. The 'Quick setup' section provides instructions for creating a new repository on the command line and pushing an existing repository from the command line. The repository is currently empty, and the page shows the repository name, visibility, and a 'Create repository' button.

demo-for-task Private

Watch 0 Fork 0 Star 0

Set up GitHub Copilot
Use GitHub's AI pair programmer to autocomplete suggestions as you code.
Get started with GitHub Copilot

Add collaborators to this repository
Search for people using their GitHub username or email address.
Invite collaborators

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH git@github.com: Fedor-Chagin/demo-for-task.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# demo-for-task" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:Fedor-Chagin/demo-for-task.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:Fedor-Chagin/demo-for-task.git
git branch -M main
git push -u origin main
```

ProTip! Use the URL for this page when adding GitHub as a remote.

После создания репозитория на выбранном сервисе можно добавить туда все изменения. Удобнее и безопаснее всего использовать вариант по SSH: для этого нужно выполнить 2 команды. Первая команда

```
git remote add origin git@github.com:ваш_логин/название_репозитория.git
```

связывает нужный локальный репозиторий с удаленным на GitHub. Origin — это просто стандартное короткое имя для удаленного репозитория. В данном случае целиком она выглядит вот так:

```
git remote add origin git@github.com:Fedor-Chagin/demo-for-task.git
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git remote add origin git@github.com:Fedor-Chagin/demo-for-task.git
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

Вторая команда

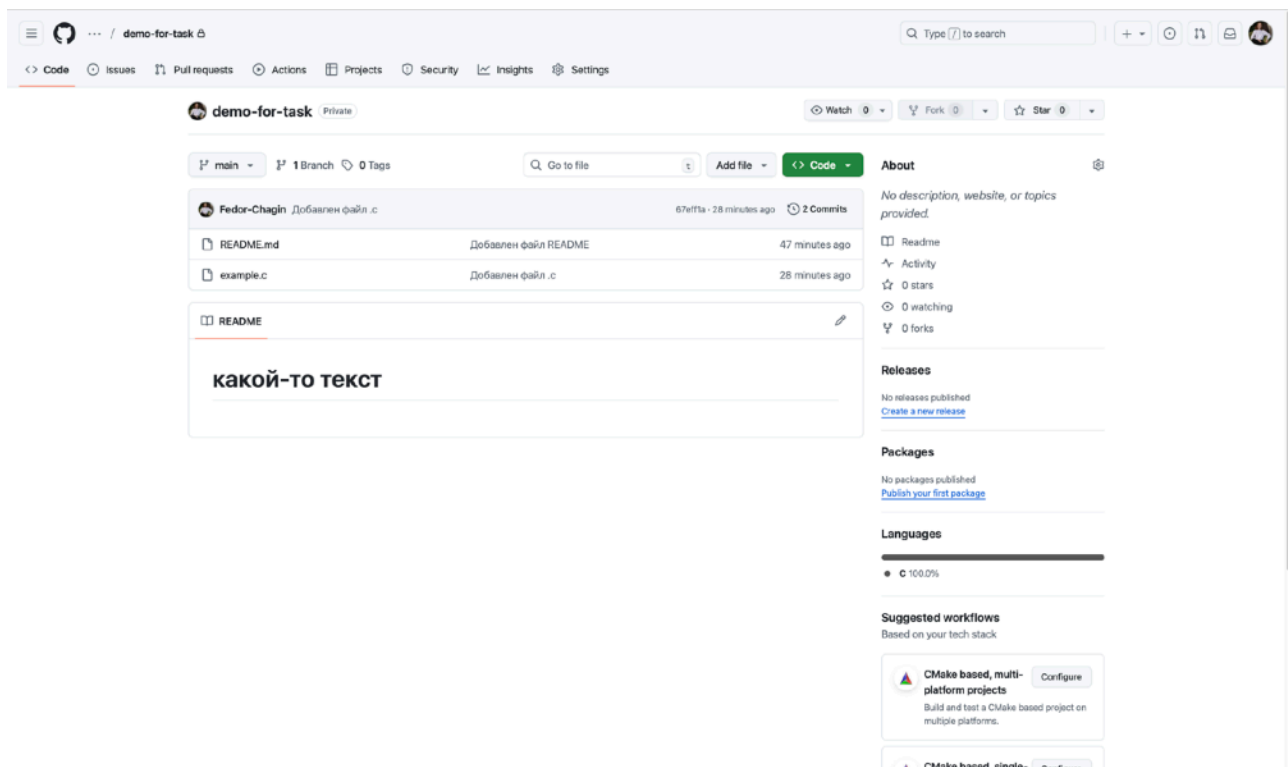
```
git push -u origin main
```

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git push -u origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 634 bytes | 634.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:Fedor-Chagin/demo-for-task.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

отправляет всю локальную историю коммитов (ветку main) на сервер GitHub. Ключ -u нужен только один раз для каждой ветки. Он запоминает, куда были отправлены изменения, и в дальнейшем для отправки новых коммитов из этой же ветки будет достаточно просто набрать git push. Изменить репозиторий (например если изначально был указан не верный), можно удалив старый выбор командой

```
git remote remove origin.
```

После выполнения команды можно обновить страницу репозитория на GitHub и увидеть, что файлы успешно загрузились – теперь наш код не только на компьютере, но и в безопасном месте на GitHub.



Однако может появиться необходимость что-то изменить в проекте – для этого нужно создать новую ветку.

`git switch -c fix-readme`

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git switch -c fix-readme
Switched to a new branch 'fix-readme'
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

Для примера можно зайти в проект и изменить файл `readme`, а так же (вручную) добавить папку `example_№1`.

```
1 # какой-то текст
2 # изменение: ещё какой-то текст
```

И сохранить + запустить эти изменения:

`git add .`

`git commit -m "Неудачные эксперименты"`

`git push -u origin fix-readme`

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git add .
fedorcagin@MacBook-Pro13 git-demo_for_task % git commit -m "Неудачные эксперименты"
[fix-readme f729554] Неудачные эксперименты
2 files changed, 1 insertion(+)
 create mode 100644 .DS_Store
fedorcagin@MacBook-Pro13 git-demo_for_task % git push -u origin fix-readme
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 759 bytes | 759.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'fix-readme' on GitHub by visiting:
remote:   https://github.com/Fedor-Chagin/demo-for-task/pull/new/fix-readme
remote:
To github.com:Fedor-Chagin/demo-for-task.git
 * [new branch]      fix-readme -> fix-readme
branch 'fix-readme' set up to track 'origin/fix-readme'.
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

Branches

New branch

Overview Yours Active Stale All

Q Search branches...

Default


Branch	Updated	Check status	Behind	Ahead	Pull request
main	21 minutes ago			Default	...


Your branches


Branch	Updated	Check status	Behind	Ahead	Pull request
fix-readme	1 minute ago		0	1	...


Active branches


Branch	Updated	Check status	Behind	Ahead	Pull request
fix-readme	1 minute ago		0	1	...


 demo-for-task Private Watch 0


 fix-readme had recent pushes 5 minutes ago Compare & pull request

 fix-readme


 2 Branches




 0 Tags

 Add file

 Code

This branch is 1 commit ahead of main Contribute

 Fedor-Chagin Неудачные эксперименты f729554 · 6 minutes ago 3 Commits

 .DS_Store	Неудачные эксперименты	6 minutes ago
 README.md	Неудачные эксперименты	6 minutes ago
 example.c	Добавлен файл .c	52 minutes ago

Но если изменения не оправдали результата, появились новые баги или просто было принято решение не вносить правки – то можно откатиться к прошлой версии. В данном случае можно просто откатиться на 1 версию назад:
`git reset --soft HEAD~1`

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git reset --hard HEAD~1
HEAD is now at 2a105e2 Добавлен файл README
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

После такого мы увидим старую версию проекта: с 1 строчкой в readme и без добавленной папки. Но если изменения не устраивают до такой степени, что их не хочется оставлять как неудачные версии – поможет полное удаление ветки: для этого также нужно переключиться на main

`git switch main`

Удалить ветку fix-readme локально

`git branch -D fix-readme`

а затем и в GitHub

`git push origin --delete fix-readme`

```
fedorcagin@MacBook-Pro13 git-demo_for_task % git switch main
Switched to branch 'main'
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)
fedorcagin@MacBook-Pro13 git-demo_for_task % git branch -D fix-readme
Deleted branch fix-readme (was 2a105e2).
fedorcagin@MacBook-Pro13 git-demo_for_task % git push origin --delete fix-readme
To github.com:Fedor-Chagin/demo-for-task.git
- [deleted]          fix-readme
fedorcagin@MacBook-Pro13 git-demo_for_task %
```

После выполнения ветка исчезнет даже на GitHub и не будут засорять процесс разработки.