

Подробный отчёт по работе с Git через терминал

Введение

Git — это распределённая система контроля версий, которая позволяет отслеживать изменения в проекте, вести историю разработок, создавать ветки для экспериментов и объединять их обратно в основную ветку.

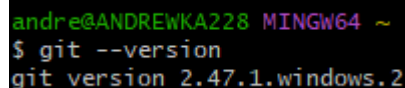
Работа с Git через **терминал** (командную строку) является самым универсальным и гибким способом, так как предоставляет доступ ко всем возможностям системы.

1. Установка и проверка Git

1. Скачиваем Git с официального сайта: <https://git-scm.com>.
2. После установки проверяем наличие Git в системе:

```
1  git --version
```

Пример результата:



```
andre@ANDREWKA228 MINGW64 ~  
$ git --version  
git version 2.47.1.windows.2
```

2. Первоначальная настройка

Перед началом работы нужно указать имя и почту — они будут отображаться в истории коммитов.

```
1  git config --global user.name "Иван Иванов"  
2  git config --global user.email "ivanov@example.com"
```

Чтобы убедиться, что параметры применились:

```
1  git config --list
```

Пример результата:

```
andre@ANDREWKA228 MINGW64 ~  
$ git config --list  
diff.astextplain.textconv=astextplain  
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  
http.sslbackend=openssl  
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt  
core.autocrlf=true  
core.fscache=true  
core.symlinks=false  
pull.rebase=false  
credential.helper=manager  
credential.https://dev.azure.com.usehttppath=true  
init.defaultbranch=master  
user.name=Andrew
```

3. Создание репозитория

Переходим в папку проекта и инициализируем репозиторий:

```
1  mkdir my-project  
2  cd my-project  
3  git init
```

Результат:

```
andre@ANDREWKA228 MINGW64 /d  
$ mkdir test  
  
andre@ANDREWKA228 MINGW64 /d  
$ cd test  
  
andre@ANDREWKA228 MINGW64 /d/test  
$ git init  
Initialized empty Git repository in D:/test/.git/  
  
andre@ANDREWKA228 MINGW64 /d/test (master)  
$
```

Теперь проект находится под управлением Git.

4. Проверка состояния

Команда `git status` показывает текущее состояние репозитория:

```
1  git status
```

Пример результата:

```
andre@ANDREWKA228 MINGW64 /d/test (master)
$ git status
On branch master

No commits yet

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

5. Добавление файлов в индекс

Создаём файл `readme.txt` и добавляем его в Git:

```
1  git add readme.txt
```

Чтобы добавить сразу все файлы:

```
1  git add .
```

6. Создание коммита

Фиксируем изменения в истории проекта:

```
1  git commit -m "Добавлен файл readme.txt"
```

Пример результата:

```
andre@ANDREWKA228 MINGW64 /d/test (master)
$ git add .

andre@ANDREWKA228 MINGW64 /d/test (master)
$ git commit -m "Readme txt added"
[master (root-commit) bcc7009] Readme txt added
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 readme.txt.txt
```

7. История коммитов

Просмотр истории:

```
1  git log
```

Пример результата:

```
andre@ANDREWKA228 MINGW64 /d/test (master)
$ git log
commit bcc70098c7eab86eb3da8c6eb79801f227bd21cc (HEAD -> master)
Author: Andrew <180178858+bluv4nul@users.noreply.github.com>
Date: Thu Sep 25 11:04:37 2025 +0300

    Readme txt added
```

Краткая форма:

```
1 git log --oneline --graph --decorate
```

Пример:

```
andre@ANDREWKA228 MINGW64 /d/test (master)
$ git log --oneline --graph --decorate
* bcc7009 (HEAD -> master) Readme txt added
```

8. Работа с ветками

Создание новой ветки:

```
1 git branch feature
```

Переключение:

```
1 git checkout feature
```

В новых версиях:

```
1 git switch feature
```

Просмотр веток:

```
1 git branch
```

Пример результата:

```
andre@ANDREWKA228 MINGW64 /d/test (master)
$ git branch feature

andre@ANDREWKA228 MINGW64 /d/test (master)
$ git checkout feature
Switched to branch 'feature'

andre@ANDREWKA228 MINGW64 /d/test (feature)
$ git branch
* feature
  master
```

9. Слияние веток

Возвращаемся в `main` и объединяем изменения из `feature` :

```
1  git checkout main
2  git merge feature
```

Если конфликтов нет, изменения объединяются автоматически.

10. Работа с удалёнными репозиториями (GitHub)

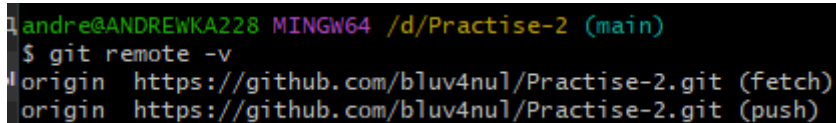
Подключение:

```
1  git remote add origin https://github.com/username/my-project.git
```

Проверка:

```
1  git remote -v
```

Пример результата:

A terminal window with a black background and white text. The prompt is 'andre@ANDREWKA228 MINGW64 /d/Practise-2 (main)'. The command '\$ git remote -v' has been executed, resulting in two lines of output: 'origin https://github.com/bluv4nu1/Practise-2.git (fetch)' and 'origin https://github.com/bluv4nu1/Practise-2.git (push)'.

```
andre@ANDREWKA228 MINGW64 /d/Practise-2 (main)
$ git remote -v
origin https://github.com/bluv4nu1/Practise-2.git (fetch)
origin https://github.com/bluv4nu1/Practise-2.git (push)
```

11. Отправка изменений

Первая отправка (нужно указать ветку):

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

Дальнейшие отправки:

```
1  git push
```

12. Получение изменений

Чтобы загрузить изменения с удалённого репозитория:

```
1 git pull origin main
```

13. Клонирование репозитория

Чтобы скачать проект с GitHub:

```
1 git clone https://github.com/username/my-project.git
```

14. Основные команды Git через терминал (сводная таблица)

Команда	Назначение
git init	Создать новый репозиторий
git status	Проверить состояние репозитория
git add <файл>	Добавить файл в индекс
git commit -m "msg"	Зафиксировать изменения
git log	Показать историю коммитов
git branch	Список веток
git checkout <ветка>	Переключиться на ветку
git merge <ветка>	Объединить изменения
git remote add origin <url>	Подключить удалённый репозиторий
git push	Отправить изменения
git pull	Получить изменения
git clone <url>	Склонировать репозиторий

15. Вывод

Работа через **терминал** даёт полный контроль над Git:

- можно выполнять **все команды без ограничений**;
- удобно использовать на разных ОС (Windows, Linux, macOS);
- легко автоматизировать задачи с помощью скриптов.

Недостаток — нужно запоминать синтаксис команд, но со временем это становится привычным.