# **Методическое пособие для работы с Git**

Git - это консольная утилита, для отслеживания и ведения истории изменения файлов, в вашем проекте. Чаще всего его используют для кода, но можно и для других файлов. Например, для картинок - полезно для дизайнеров.

С помощью Git-a вы можете откатить свой проект до более старой версии, сравнивать, анализировать или сливать свои изменения в репозиторий.

Репозиторием называют хранилище вашего кода и историю его изменений. Git работает локально и все ваши репозитории хранятся в определенных папках на жестком диске.

Так же ваши репозитории можно хранить и в интернете. В данном руководстве будут приведены примеры с использованием сервиса GitHub.

Вся теория понятным языком объясняется на Хабре[[1]](#footnote-1), поэтому, ~~нагло списав~~ цитировав оттуда небольшое введение перейдем к установке Git-а и освоению его базовых (необходимых) команд.

**Установка**

Для системы Windows

Windows почти полностью отказались от использования консоли и поэтому, вытирая слёзы, переходим на официальный сайт Git-а (достаточно ввести в поиск «Git») и скачиваем его оттуда.

Дабы понять, установился ли Git, и видит ли его устройство, откройте консоль и введете команду

git --version

# 

Если вы получили такой же результат (версия может отличаться), то смело спешу вас порадовать, вы установили Git, программу которая может спрятать весь ваш 18+ архив используя папку на рабочем столе и ваши прямые руки на клавиатуре.

По мнению автора консоль в Windows не очень удобная, поэтому он использует Git bash, который является консолью Git-а с более приятным интерфейсом.

Для системы Linux

В зависимости от установленного дистрибутива команда может немного отличаться, в данном случае на Mint, Debian, Ubuntu используется данная консольная команда

sudo apt install git

В Linux консоль всемогущая и для нее не существует преград, так что ничего дополнительного по типу Git bash ставить не придется.

Пусть все узнают кто накосячил в коде

Для того что бы коммит (комментарий, об этом будет описано позже) не был безымянным, нужно дать Git-у понять кто им пользуется, для этого вводим в терминал (консоль) данные команды

git config --global user.name “*ваше имя*”

git config --global user.email “*ваша почта@gg.ru*”

Курсив заменить на нужное, !кавычки оставить!

Настало время перейти к работе

И начнём мы с создания нашего первого репозитория.

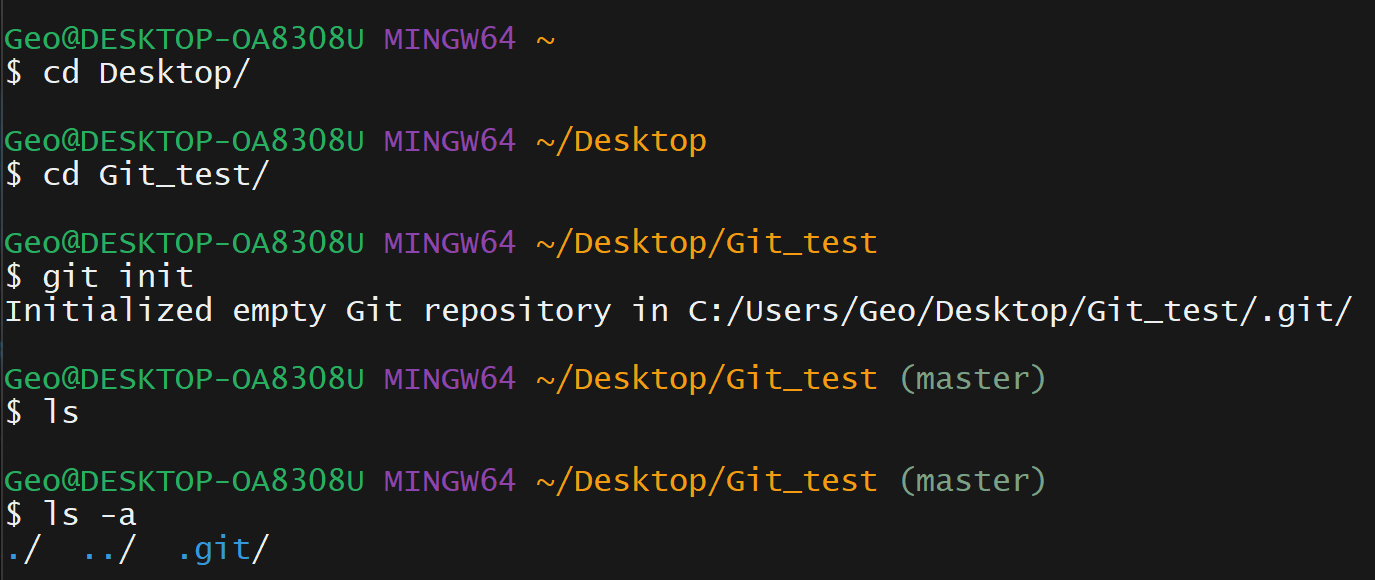
Создайте на рабочем столе папку (или там, где удобно вам), далее при помощи команды

Сd «Путь к папке»

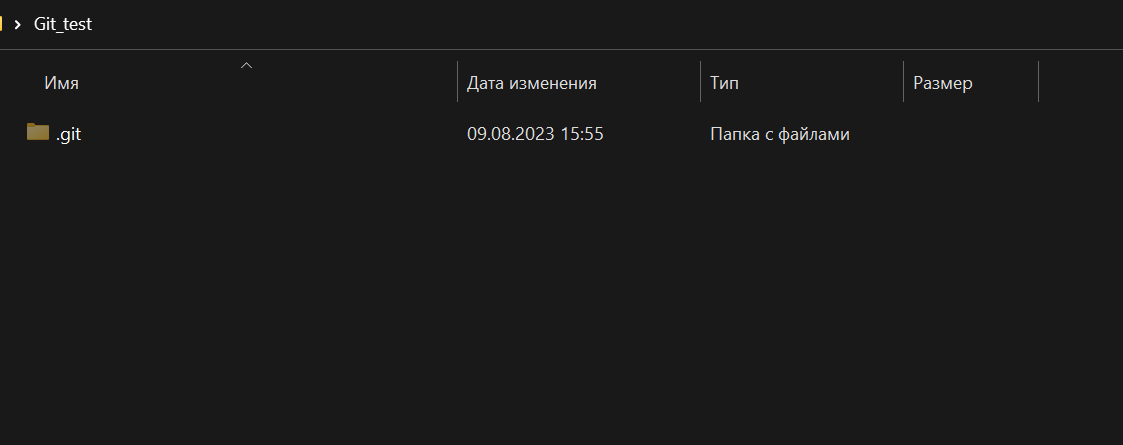
переходим к нашей папке, при помощи команды

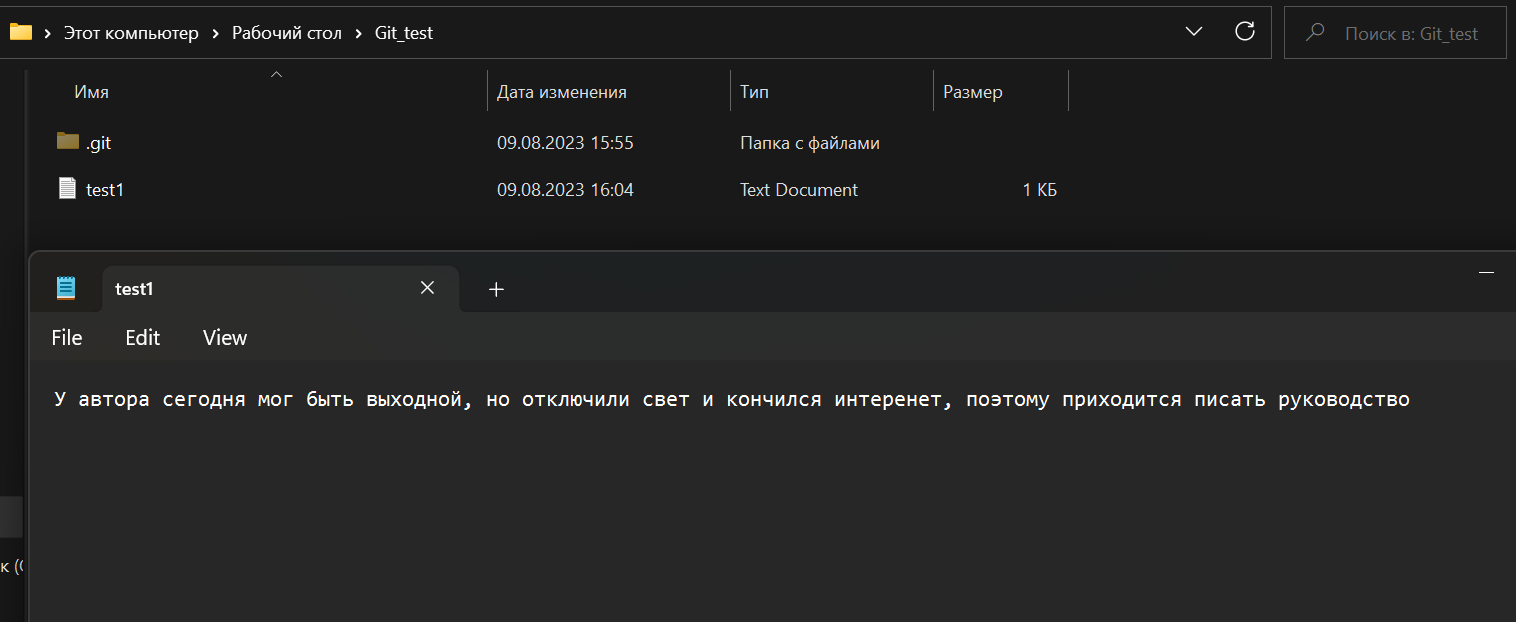
git init

создаём (инициализируем) в данной папке репозиторий, который представляет из себя скрытую папку **.git**, выглядит это все примерно так:



\*Путь можно указывать целиком, а не по частям как на скрине.



Создадим в этой папке текстовый файл и запишем в него что-нибудь

Для того что бы выбрать файлы в папке для его отслеживания (грубо-говоря Git следит за выбранным файлом), используется команда

git add .

Данная команда добавит все файлы в папке в следующий коммит, точка обязательна, аналог этой команды

git add --all

Для того что бы сохранить какой-то определенный файл в папке, используется команда

git add «имя файла»

Далее необходимо проверить, остались ли еще не отслеживаемые файлы, для этого есть команда

git status

Наконец перейдем к созданию коммита, используем команду

git commit -m “*комментарий*”

-m это флаг (параметр), позваляет нам дать некоторые указания для команды, в данном случае –m позволяет оставить комментарий (сообщение)

\*Все что идёт после – и -- является флагами, с которыми желательно ознакомиться отдельно, так как они сильно упростят жизнь опытному пользователю, но в данном руководстве автор постарался использовать минимум флагов, чтобы никого не запутать и дать понять основы.

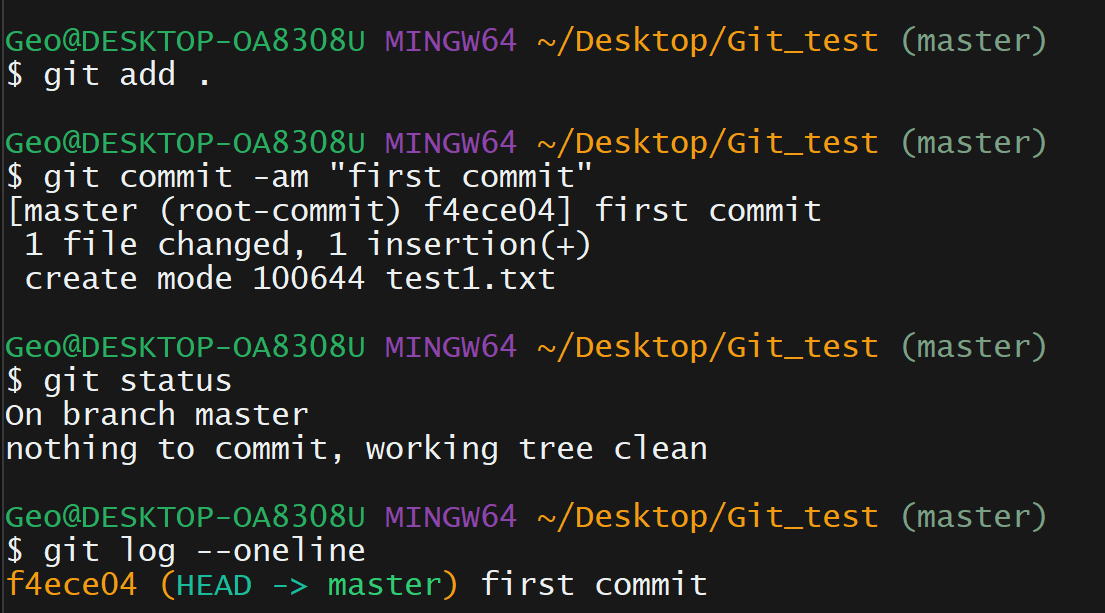
Коммит это комментарий, в котором сохранены изменения

Для того что бы посмотреть имеющиеся коммиты используется команда

git log --oneline

Флаг --oneline нужен, чтобы уменьшить количество информации, выдаваемой на экран. С этим флагом каждый коммит показывается в одну строчку.

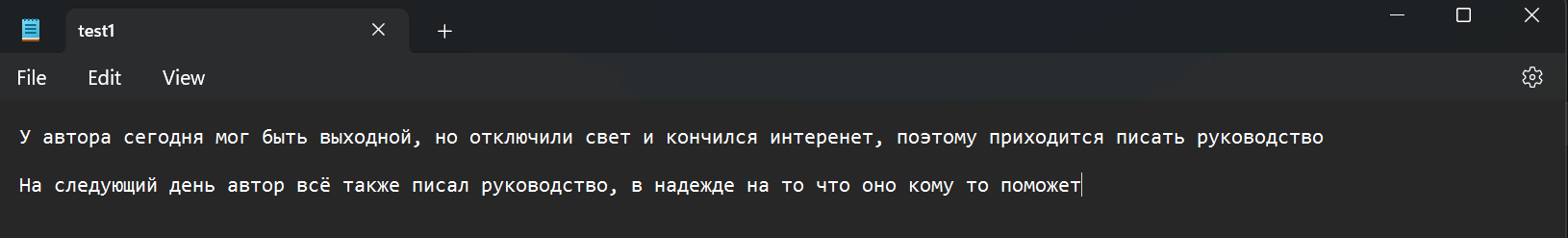
Долгожданный пример



К коммитам стоит относиться серьёзно и не делать их после каждой написанной буквы, следует коммитить исправленные крупные ошибки добавление больших новых блоков и т.д., не стоит забывать, что ваш код (или что то иное) сохраняется в самом текстовом редакторе.

Сохранение изменений и переход по коммитам

Предположим, что в наш текстовый файл нам захотелось добавить еще один “крупный” фрагмент



Теперь нам необходимо дать Git-у понять, что данные изменения должны быть внесены в репозиторий, для этого используются уже известные нам команды

git add

git commit

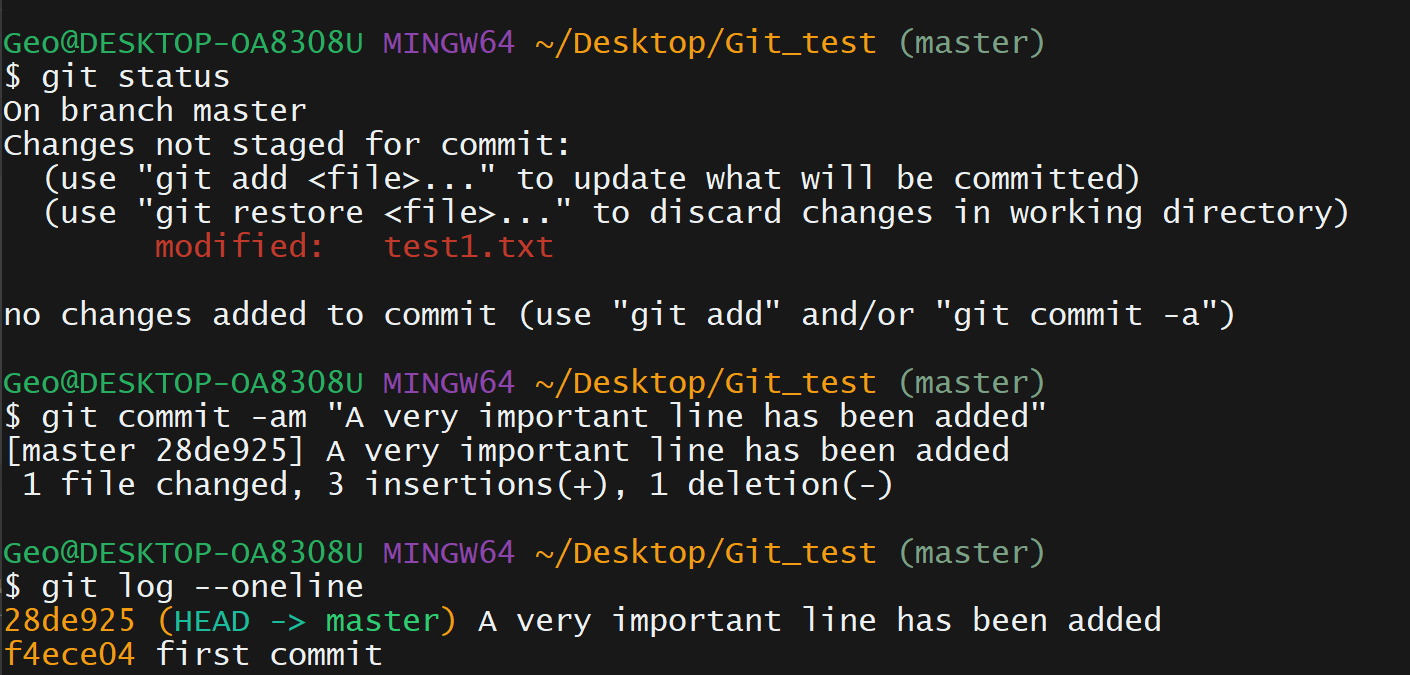
\*В дальнейшем можно пользоваться флагом –am для команды

git commit

-am это –a и –m, два флага записанные вместе, флаг –а заменяет команду

git add --all

тем самым упрощая нам жизнь, если мы хотим сохранить изменения во ВСЕХ файлах.



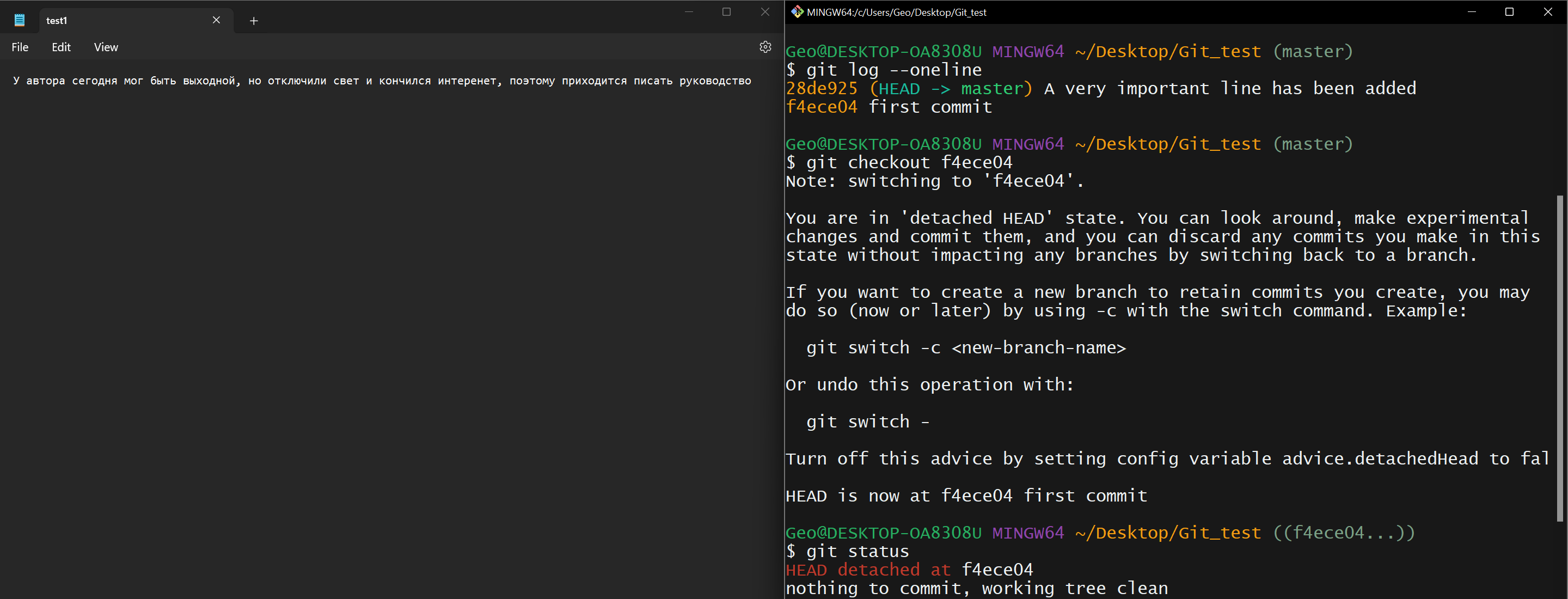
\*В данном случае дополнительно проверяем командой git status наличие изменений и добавляем их в репозиторий.

Мы сохранили изменения, теперь по какой-то неизвестной причине нам нужно вернуться на первый коммит, для этого используется команда

git checkout «хеш нужного коммита»

Получаем откат изменений

Команда git status также указывает нам на то что “голова” переехала на другое место

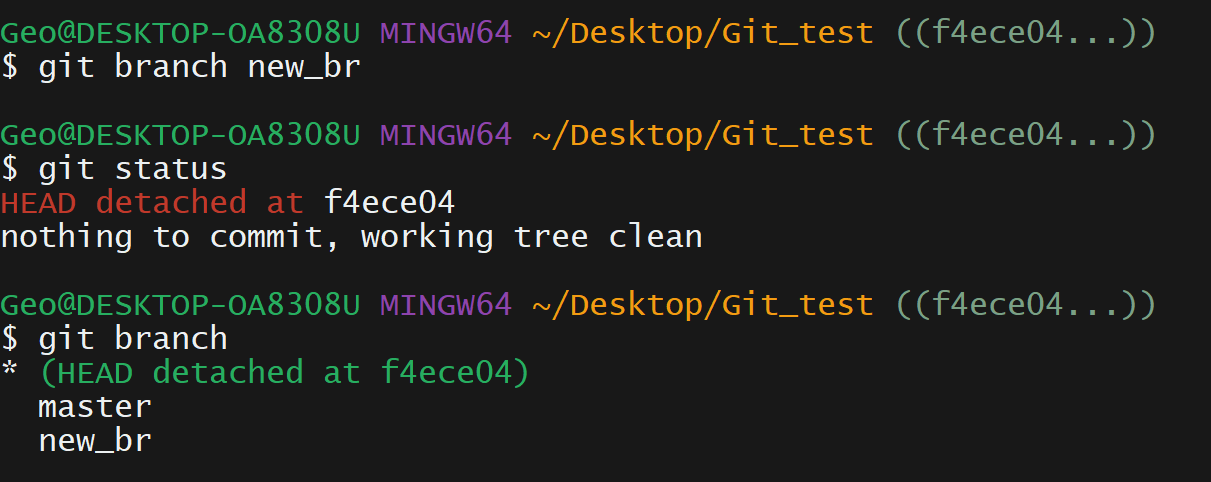


Делать коммиты не с головы не рекомендуется (пока полностью не освоен принцип работы, лучше не рисковать, так можно потерять последний коммит и придется делать всю работу заново)

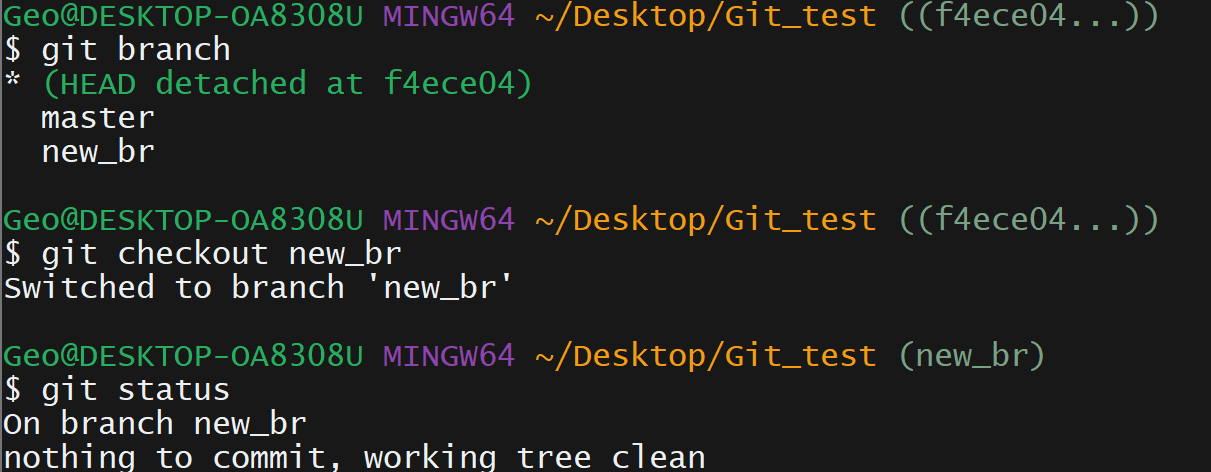
Если все же хочется внести какие-либо изменения на данном коммите, то следует создать новую ветку при помощи команды

git branch –b “Имя новой ветки”

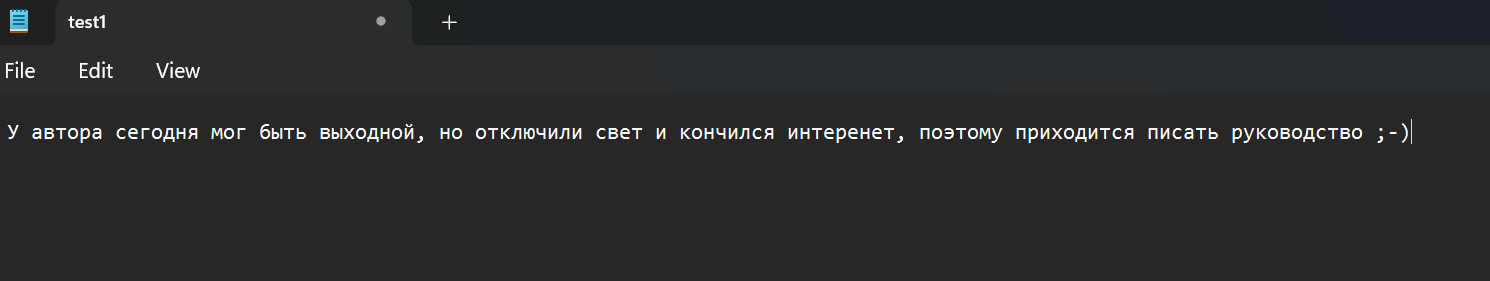
Флаг –b (у автора склероз, он его забыл и теперь видно, что голова находится ни пойми где, исправим это командой git checkout new\_br) сразу переключит нас на новую ветку (в противном случае все изменения останутся в старой ветке и коммиты могут быть утеряны)



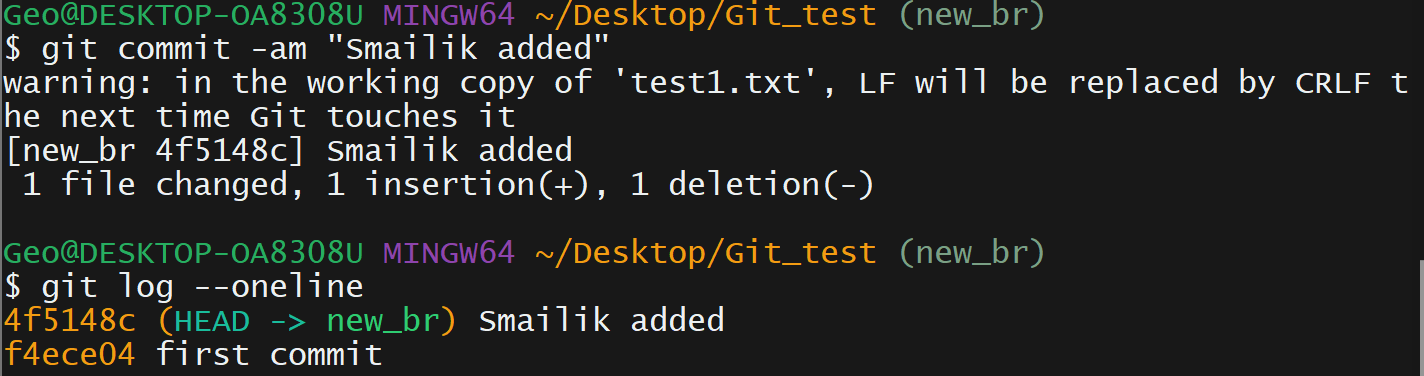
При помощи команды git branch смотрим имеющиеся у нас ветки



Можем внести изменения в ветке “new\_br”, например, добавим смайлик в конце строки))



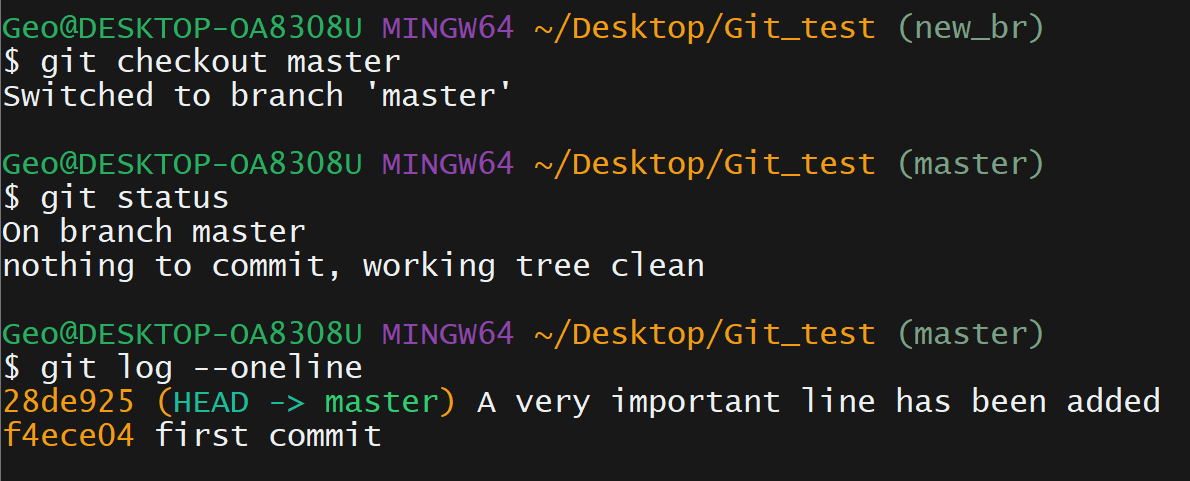
Сохраним изменения

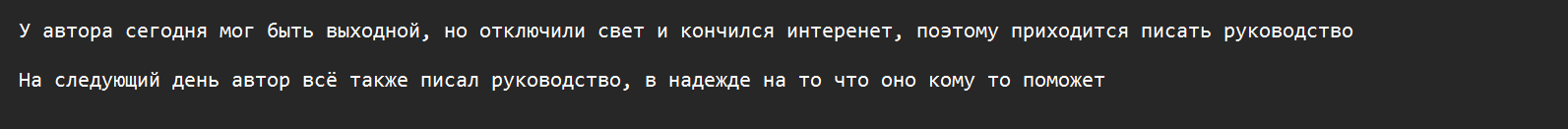


Видим, что коммит с “очень важной строкой” отсутствует, так как мы находимся на другой ветке

Что бы вернуться на основную ветку (основная ветка по умолчанию называется master) введем

git checkout master





Видим, что текст в файле стал таким, каким мы его “закоммитили” в последнем коммите ветки master

Теперь предположим, что нам больше не нужна ветка со смайликом ☹

Удалим ее при помощи команды

git branch –d “имя ветки”

\*Прежде чем удалить ветку, нужно перейти на другую.

Также при помощи таких действий можно исправлять ошибки в коде, только не откатываясь назад на некоторый коммит, а создавая новую ветку с головы, после внесения необходимых изменений используется команда

git merge “имя ветки”

**Перед объединение необходимо сначала перейти на основную ветку и только потом “влить” в нее ветку с исправлением ошибок.**

Работа с удалённым репозиторием на GitHub

Добрались до финала, тут обсудим как нам загрузить наши файлы на GitHub или получить их оттуда.

До текущего момента мы работали с локальным репозиторием, который сохранялся в папке на компьютере. Если мы хотим иметь возможность сохранения проекта в интернете, создадим репозиторий на GitHub. Для начала нужно зарегистрироваться на сайте github.com

После регистрации нажимаем кнопочку *“+”* и вводим название репозитория (уже не локального, а ноходящего в интеренете).

Далее GitHub сам даст нужные инструкции, но на всякий случай они будут продублированы тут.

Добавляем удаленный репозиторий при помощи команды

git remote add origin “ссылка на репозиторий github”

\*origin является именем нашего удаленного репозитория, можете назвать его как вам угодно

Далее отправляем (пушим) наш репозиторий (выбранную ветку) на удаленный репозиторий (с именем origin)

git push –u origin master

Флаг –u используется только в первый раз, дабы связать нашу ветку master локального репозитория c удаленным репозиторием, далее требуется только

git push

Теперь разберемся, как же нам получить файлы из удаленного репозитория.

Тут все просто

git clone “ссылка на репозиторий github”

Также, после использования git clone, можно пользоваться командой

git pull

которая обновит ваши локальные файлы до последной версии (то есть добавит все коммиты, которых у вас нет)

\*Если локальные файлы будут изменены, то git pull выдаст ошибку

\*Место для ваших добавлений

GitHub позволяет редактировать репозиторий не только одному пользователю, грубо говоря для изменения репозитория клонируйте его к себе, добавьте что-то сделайте комммит и после запуште его обратно, поэтому буду рад, если кто то проявит желание впистаь сюда дополнительные блоки с функциями и командами.

\*Ниже свободное место (многоточие)\*

...

На этом всё, данное руководство будет находиться на GitHub в открытом доступе для всех страждущих.

Эксперементируйте, создавайте свои репозитории, пробуйте строить деревья из веток и много другой, ограничений нет (только не удалите папку system32)

Для тех, кто дошел до конца и смог стерепеть всю душность автора, небольшой бонус: сайт для тренировки работы с гитом, на нем приведены графические иллюстрации того, что вы делаете <https://learngitbranching.js.org/?locale=ru_RU>

Автор всегда будет рад получить какие-либо замечание по этому руководству, так как его цель помочь вам в изучении работы с Git.

В данном руководстве далеко не весь функционал гита, поэтому автор ждет ваши дополнения.

1. \*Хабр – интернет ресурс, находящийся в открытом доступе для всех страждущих и не очень.

   P.S. Для общего развития и успешной работы в сфере программирования желательно изучить статью «Makefile для самых маленьких», которая как раз находится на Хабре. [↑](#footnote-ref-1)