Система управления версиями (Version Control System, VCS)   
  
— программное обеспечение для облегчения работы с изменяющейся информацией. Это программа, которая позволяет:

* Упростить создание резервных копий вашего кода (ваших файлов)
* Хранить резервную копию на отдельном компьютере (например на сервере в Интернете)
* Автоматически объединять исходный код разных версий программы, над которыми работают разные программисты
* Показывать чем отличается одна версия исходного кода от другой
* И многое другое ...

Мы будем использовать программу для управления версиями - git.   
[*Ссылка для скачивания.*](https://git-scm.com/)

Репозиторий (хранилище) — место, где хранятся и поддерживаются какие-либо данные. Обычно это просто папка с файлами. Локальный репозиторий – это папка с файлами на вашем компьютере.

* git - не делает ничего в автоматическом режиме (без участия пользователя).
* Для совершения всех действий нужно давать программе команды.
* Git устанавливается вместе со своей оболочкой командной строки (более удобной чем cmd.exe)
* Чтобы запустить эту оболочку нужно в проводнике Windows в контекстном меню выбрать "git Bash here". (В ней работают и обычные команды для работы с файлами. Например ls – показать содержимое текущей папки; cd перейти в другую папку)
* Запустить эту программу лучше прямо из той папки, где у вас хранится исходный код программы.

## Создание репозитория

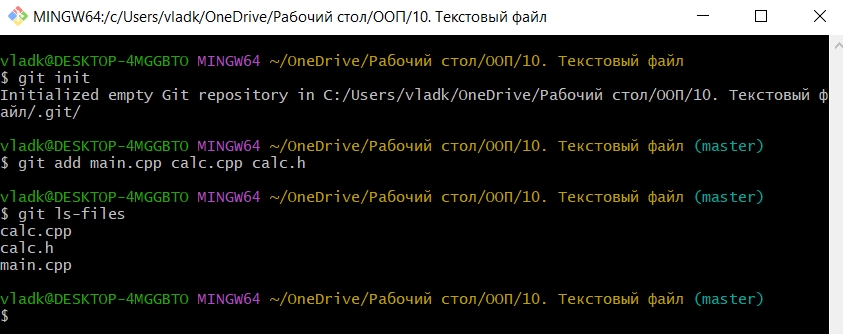
1. Создать репозиторий: git init

Git создаст скрытую папку, где будет хранить служебную информацию. Признак того, что репозиторий создан - в консоли появилась надпись (master).

1. Добавить файлы к отслеживанию: git add файлы

Как правило в отслеживании должны быть файлы исходных кодов и другие файлы, необходимые для компиляции и запуска программы. Исполняемые файлы не отслеживаются. Потому, что их всегда можно получить после компиляции и чтобы не засорять ими репозиторий.

1. Просмотреть список отслеживаемых файлов: git ls-files



## Фиксация изменений

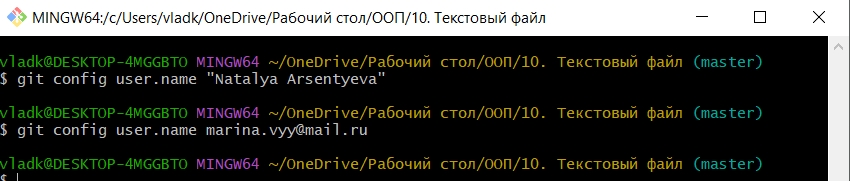
Git не запоминает изменения в реальном времени. Это происходит потому, что каждое изменение должно быть логически завершённым. А это решает разработчик. Чтобы зафиксировать или записать текущее состояние файлов используется команда commit.

1. Нужно указать ваше имя и адрес электронной почты. (Это важно, потому что каждый коммит в Git содержит эту информацию):

git config --global user.name "John Doe"

git config --global user.email johndoe@example.com

Если указана опция --global, то эти настройки достаточно сделать только один раз, поскольку в этом случае Git будет использовать эти данные для всего, что вы делаете в этой системе. Если для каких-то отдельных проектов вы хотите указать другое имя или электронную почту, можно выполнить эту же команду без параметра --global в каталоге с нужным проектом.

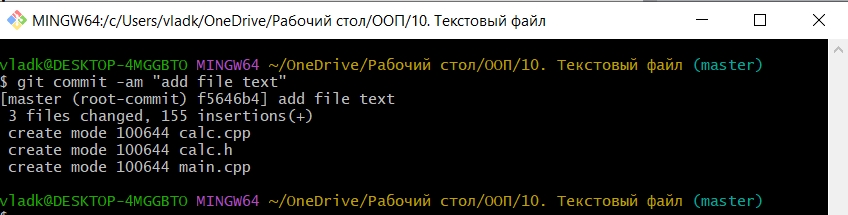


1. Зафиксировать изменения (сделать коммит):

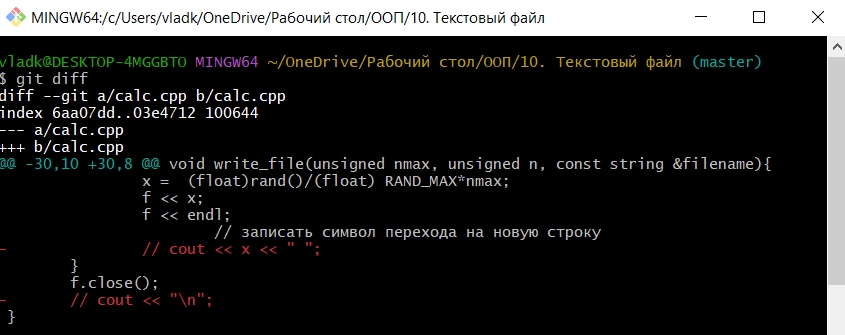
git commit -am "кратко об изменениях"

Ключи команды commit:   
-a - добавить все отслеживаемые файлы в фиксацию   
-m - комментарий к фиксации

В комментариях следует кратко описывать сделанные изменения.

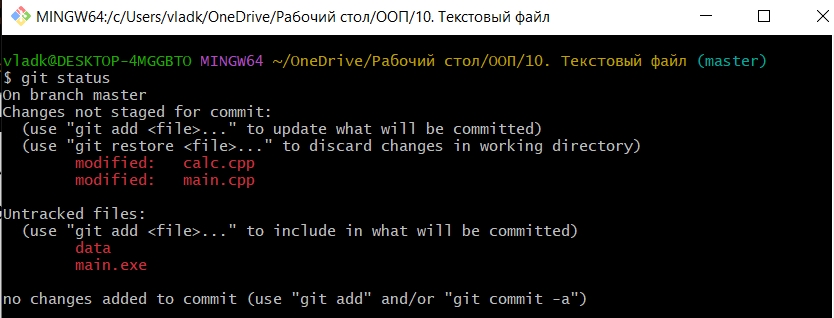


1. Посмотреть изменения: git diff



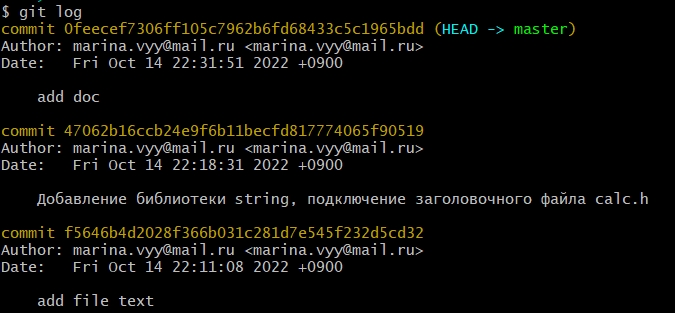
Выйти из просмотра изменений клавиша: Q

1. Статус репозитория (просмотр измененных файлов): git status



git заметил что один из файлов изменился.  
Есть файлы и папки, за которыми git не следит – не отслеживаемые файлы

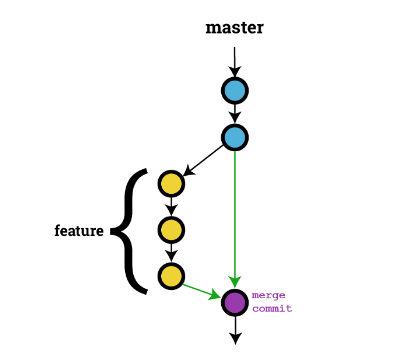
1. История коммитов: git log



## Ветки

Если планируются обширные изменения, то стоит подумать над созданием отдельной ветви, чтобы параллельно существовала исходная версия программы и версия, в которую вносятся изменения - *рабочая версия*. Допускается, что рабочая версия может не транслироваться, работать с ошибками. Однако в любой момент должна быть возможность вернутся к исправной версии программы.

После того как изменения будет закончены, программа станет синтаксический правильной и протестирована, изменения рабочей версии добавляются к основной версии. При внесении новых изменений, снова создаётся рабочая версия и всё повторяется заново.



В git отдельные версии программы хранятся в ветках (branches). Одновременно с созданием репозитория создаётся основная ветка – master

Команды для работы с ветками:   
branch <имя\_ветки> - создание ветки  
checkout <имя\_ветки> - переключение на ветку  
checkout -b <имя\_ветки> - создание ветки и переключение на неё  
merge <имя\_ветки> - объединение текущей ветки с другой

Переключится с ветки на ветку можно только если в текущей ветке все изменения зафиксированы.

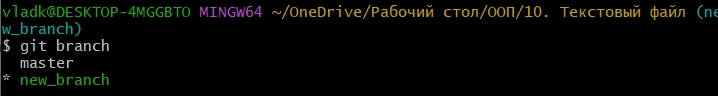
1. Создание рабочей ветки, переключение и просмотр списка всех веток:

git branch -b new\_branch

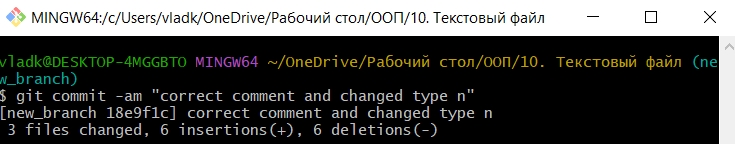
git checkout new\_branch

(или все сразу git checkout -b new\_branch)

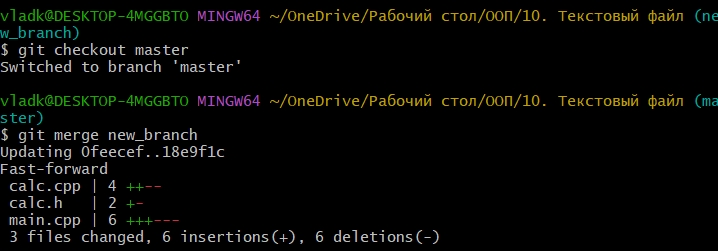
git branch



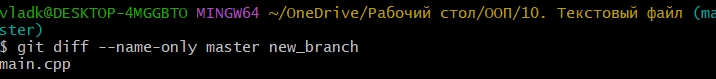
1. Вносим коммит на новой ветке:



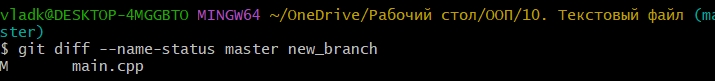
1. Возвращение на основную ветку: git checkout <имя\_ветки>
2. Объединение основной ветки с рабочей: git merge <имя\_ветки>



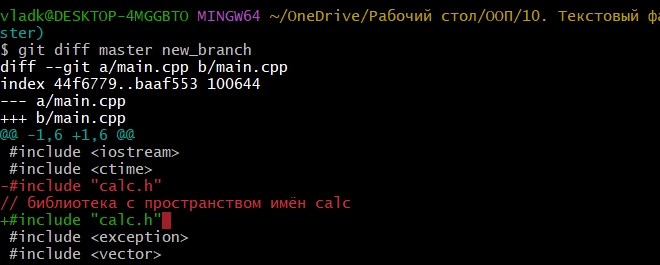
1. Просмотр списка отличающихся файлов и статуса файлов:   
   git diff --name-only ветка1 ветка2



git diff --name-status ветка1 ветка2



1. Просмотр различий между ветками: git diff ветка1 ветка2



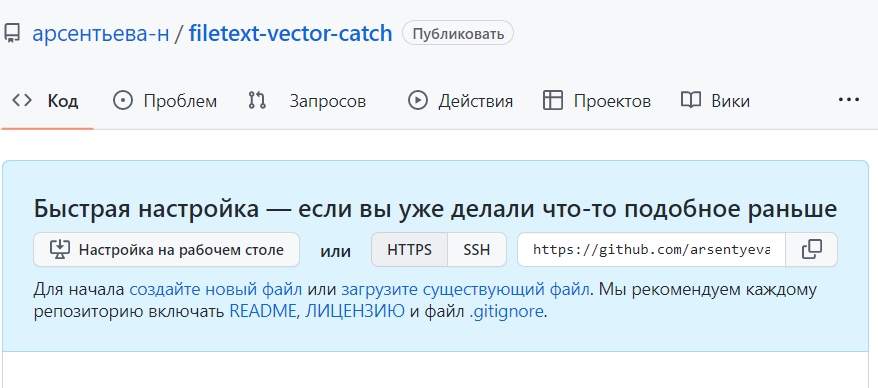
## Удалённый репозиторий

*Локальный репозиторий* позволяет легко отслеживать изменения, хранить несколько версий программы. Всегда есть возможность вернутся к предыдущей версии.

*Удалённый репозиторий* помимо этого делает удобной групповую разработку: разработчик отправляет свои изменения в общий удалённый репозиторий и забирают из него изменения сделанные другими разработчиками.   
Кроме того, удалённый репозиторий можно рассматривать как резервную копию локального.

Веб хостинги использующие git: GitHub, GitLub,Bitbicket.

1. Создать удалённый репозиторий на сайте (например github). Запомнить адрес репозиториия.



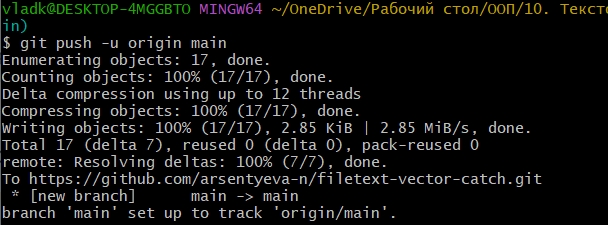
1. Настройка локального репозитория. Он должен знать об удалённом:

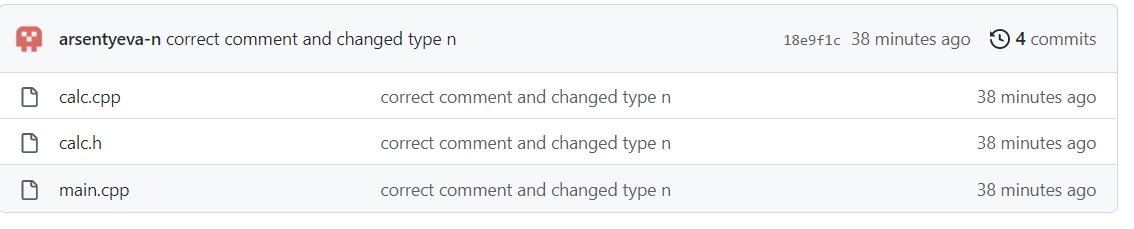
git remote add origin https://github.com/Usernsme/Reponame.git

1. Отправим основную ветку в удалённый репозиторий:

git push -u origin master (или main)

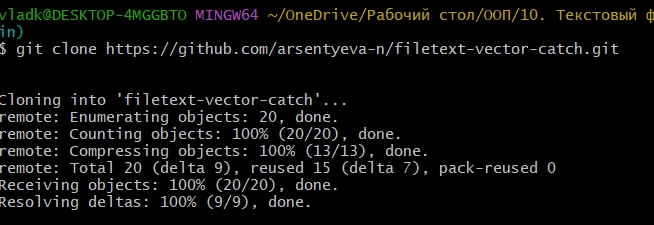
(origin - псевдоним для удалённого репозитория)





В дальнейшем для копирования локального репозитория в удалённый, достаточно ввести git push.

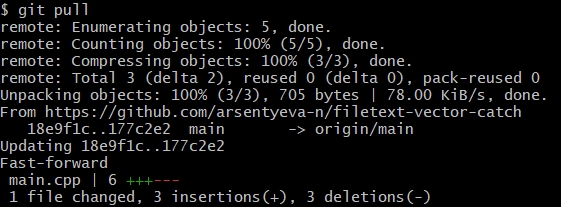
1. Файлы можно изменять прямо на гитхабе и там же делать коммит.
2. Создание локальной копии удаленного репозитория:

git clone https://github.com/Username/Reposname.git

При этом на компьютере создастся отдельная папка (с названием удалённого репозитория), в которую скопируется содержимое удалённого репозитория.



1. Добавление изменений с удаленного репозитория и слияние с активной веткой локального репозитория: git pull



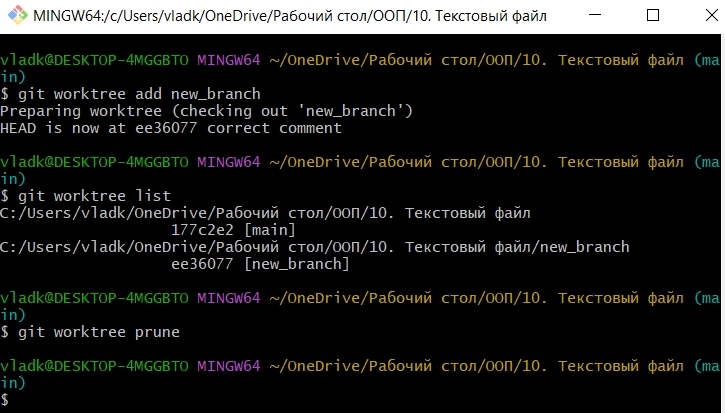
# Как работать с Git worktree: краткая инструкция

C версии 2.5 в **Git** появились рабочие деревья – **worktree**. Это новшество позволяет выгружать в отдельную папку единичные ветки и работать с ними параллельно работе с текущей рабочей веткой.  
Рабочие деревья позволяют держать запущенными две версии вашего софта, что может быть полезно для сравнения поведения после внесённых правок.  
Подробную документацию по работе можно найти [здесь](http://git-scm.com/docs/git-worktree).

## Пример работы с worktree

1. Добавить рабочую ветку  
   git worktree add ../folder name-of-branch
2. Посмотреть список рабочих деревьев  
   git worktree list
3. Когда все изменения сделаны и отправлены на сервер можно просто удалить эту папку, а затем выполнить команду которая всё подчистит.  
   git worktree prune





Источники:

[Как работать с Git worktree](https://inoyakaigor.ru/blog/85)  
[Основы работы с Git](https://gist.github.com/rdnvndr/cb21a06c5a71fd71213aed1619380b8e)  
[Слайды с Git](https://raw.githubusercontent.com/VetrovSV/Programming/master/git_lec.pdf)