**Практическая работа №1. Работа с Git. Основы**

Что такое Git?

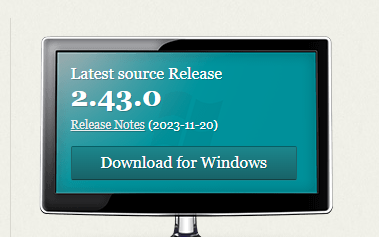
Git — это распределенная система управления версиями, которая используется для отслеживания изменений в исходном коде программного обеспечения и координации работы нескольких разработчиков над одним проектом. Он был создан Линусом Торвальдсом в 2005 году.

Основные цели Git:

1. **Отслеживание изменений:** Git позволяет фиксировать каждое изменение в исходном коде, что обеспечивает полный контроль над историей проекта.
2. **Коллаборация:** Git позволяет нескольким разработчикам одновременно работать над одним и тем же проектом. Каждый разработчик может вносить свои изменения, а затем объединять их с изменениями других.
3. **Ветвление и слияние:** Git позволяет создавать отдельные ветки для работы над конкретными функциональностями или исправлениями ошибок. Затем эти ветки могут быть объединены (слиты) обратно в основную ветку.
4. **Локальная работа:** Git позволяет разработчикам работать локально, не завися от сетевого соединения. Это увеличивает эффективность и удобство работы.
5. **Эффективное хранение данных:** Git использует эффективные алгоритмы хранения данных, что делает его компактным и быстрым.
6. **Откат к предыдущим версиям:** Разработчики могут легко откатываться к предыдущим версиям проекта, если что-то пошло не так или если нужно вернуться к предыдущей версии кода.

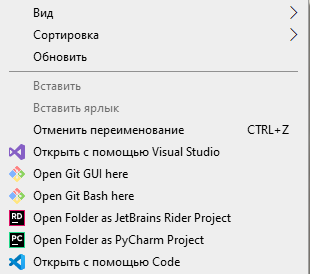
В целом, Git предоставляет мощные инструменты для управления версиями и совместной разработки программного обеспечения, делая процесс более удобным и эффективным для команд разработчиков.

**Шаг 1. Установка GIT**

1. Заходим на официальный сайт - <https://git-scm.com/>
2. Нажимаем на кнопку установки - 
3. Выбираем нужную нам версию



1. После установки в контекстном меню появится Open Git Bash here, открываем его



1. Прописываем

**echo "url=https://github.com" | git credential reject**

**Сначала читаем теорию, без нее у вас в последующем будут трудности**

**И выполняем каждый шаг который тут описан**

**Серым цветом – то что надо писать в консоль**

**Настройка**

Итак, мы установили git, теперь нужно добавить немного настроек. Есть довольно много опций, с которыми можно играть, но мы настроим самые важные: наше имя пользователя и адрес электронной почты. Откройте терминал git bush и запустите команды:

git config --global user.name "Ваше имя"

git config --global user.email [ВашаПочта@example.com](mailto:ВашаПочта@example.com)

Теперь каждое наше действие будет отмечено именем и почтой. Таким образом, пользователи всегда будут в курсе, кто отвечает за какие изменения — это вносит порядок.

Git хранит весь пакет конфигураций в файле .gitconfig, находящемся в вашем локальном каталоге. Чтобы сделать эти настройки глобальными, то есть применимыми ко всем проектам, необходимо добавить флаг –global. Если вы этого не сделаете, они будут распространяться только на текущий репозиторий.

Для того, чтобы посмотреть все настройки системы, используйте команду:

git config --list

Для удобства и легкости зрительного восприятия, некоторые группы команд в Гит можно выделить цветом, для этого нужно прописать в консоли:

git config --global color.ui true

git config --global color.status auto

git config --global color.branch auto

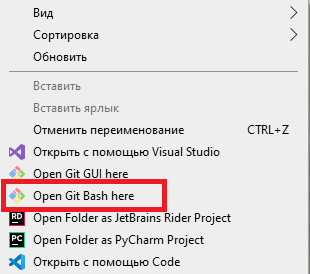
Если вы не до конца настроили систему для работы, в начале своего пути - не беда. Git всегда подскажет разработчику, если тот запутался, например:

1. Команда git --help - выводит общую документацию по git
2. Если введем git log --help - он предоставит нам документацию по какой-то определенной команде (в данном случае это - log)
3. Если вы вдруг сделали опечатку - система подскажет вам нужную команду
4. После выполнения любой команды - отчитается о том, что вы натворили
5. Также Гит прогнозирует дальнейшие варианты развития событий и всегда направит разработчика, не знающего, куда двигаться дальше

**Создание нового репозитория**

Как мы отметили ранее, git хранит свои файлы и историю прямо в папке проекта. Чтобы создать новый репозиторий, нам нужно открыть терминал, зайти в папку нашего проекта и выполнить команду init. Это включит приложение в этой конкретной папке и создаст скрытую директорию. git, где будет храниться история репозитория и настройки.

**Создайте на рабочем столе папку под названием git\_ВашаФамилия. Зайдите туда, и откройте уже в самой папке Git Bush консоль, он автоматически поймет, что находится в вашей папки.**

****

После этого пропишите команду

*git init*

Командная строка должна вернуть что-то вроде:

**Initialized empty Git repository in /home/user/Desktop/git\_exercise/.git/**

Это значит, что наш репозиторий был успешно создан, но пока что пуст. Теперь создайте текстовый файл под названием hello.txt и сохраните его в вашей папке.

**Определение состояния**

status — это еще одна важнейшая команда, которая показывает информацию о текущем состоянии репозитория: актуальна ли информация на нём, нет ли чего-то нового, что поменялось, и так далее. Запуск git status на нашем свежесозданном репозитории должен выдать:

git status

On branch master

Initial commit

Untracked files:

(use "git add ..." to include in what will be committed)

hello.txt

Сообщение говорит о том, что файл hello.txt неотслеживаемый. Это значит, что файл новый и система еще не знает, нужно ли следить за изменениями в файле или его можно просто игнорировать. Для того, чтобы начать отслеживать новый файл, нужно его специальным образом объявить.

**Подготовка файлов**

В git есть концепция области подготовленных файлов. Можно представить ее как холст, на который наносят изменения, которые нужны в коммите. Сперва он пустой, но затем мы добавляем на него файлы (или части файлов, или даже одиночные строчки) командой add и, наконец, коммитим все нужное в репозиторий (создаем слепок нужного нам состояния) командой commit.  
В нашем случае у нас только один файл, так что добавим его:

git add hello.txt

Если нам нужно добавить все, что находится в директории, мы можем использовать (точка обозначает все файлы):

git add .

Проверим статус снова, на этот раз мы должны получить другой ответ:

git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached ..." to unstage)

new file: hello.txt

Файл готов к коммиту. Сообщение о состоянии также говорит нам о том, какие изменения относительно файла были проведены в области подготовки — в данном случае это новый файл, но файлы могут быть модифицированы или удалены.

**Фиксация изменений**

**Как сделать коммит**

Представим, что нам нужно добавить пару новых блоков в html-разметку (index.html) и стилизовать их в файле style.css. Для сохранения изменений, их необходимо закоммитить. Но сначала, мы должны обозначить эти файлы для Гита, при помощи команды git add, добавляющей (или подготавливающей) их к коммиту. Добавлять их можно по отдельности:

git add index.html

git add css/style.css

или вместе - всё сразу:

git add .

Конечно, добавлять всё сразу удобнее, чем прописывать каждую позицию отдельно. Однако, тут надо быть внимательным, чтобы не добавить по ошибке ненужные элементы. Если же такое произошло изъять оттуда ошибочный файл можно при помощи команды

git reset:

git reset css/style.css

Теперь создадим непосредственно сам коммит

git commit -m 'Add some code'

Флажок -m задаст commit message - комментарий разработчика. Он необходим для описания закоммиченных изменений. И здесь работает золотое правило всех комментариев в коде: «Максимально ясно, просто и содержательно обозначь написанное!»

**Как посмотреть коммиты**

Для просмотра все выполненных фиксаций можно воспользоваться историей коммитов. Она содержит сведения о каждом проведенном коммите проекта. Запросить ее можно при помощи команды:

git log

В ней содержится вся информация о каждом отдельном коммите, с указанием его хэша, автора, списка изменений и даты, когда они были сделаны. Отследить интересующие вас операции в списке изменений, можно по хэшу коммита, при помощи команды git show:

git show hash\_commit

Ну а если вдруг нам нужно переделать commit message и внести туда новый комментарий, можно написать следующую конструкцию:

git commit --amend -m 'Новый комментарий'

В данном случае сообщение последнего коммита перезапишется. Но злоупотреблять этим не стоит, поскольку эта операция опасная и лучше ее делать до отправки коммита на сервер.

**Удаленные репозитории**

Сейчас наш коммит является локальным — существует только в директории. git на нашей файловой системе. Несмотря на то, что сам по себе локальный репозиторий полезен, в большинстве случаев мы хотим поделиться нашей работой или доставить код на сервер, где он будет выполняться.

**1. Что такое удаленный репозиторий**

Репозиторий, хранящийся в облаке, на стороннем сервисе, специально созданном для работы с git имеет ряд преимуществ. Во-первых - это своего рода резервная копия вашего проекта, предоставляющая возможность безболезненной работы в команде. А еще в таком репозитории можно пользоваться дополнительными возможностями хостинга. К примеру -визуализацией истории или возможностью разрабатывать вашу программу непосредственно в веб-интерфейсе.

**Клонирование**

Клонирование — это когда вы копируете удаленный репозиторий к себе на локальный ПК. Это то, с чего обычно начинается любой проект. При этом вы переносите себе все файлы и папки проекта, а также всю его историю с момента его создания. Чтобы склонировать проект, сперва, необходимо узнать где он расположен и скопировать ссылку на него. В нашем руководстве мы будем использовать адрес https://github.com/tutorialzine/awesome-project**, но вам необходимо, создать свой репозиторий в GitHub**

**Шаг 2. Регистрация на GitHub**

**GitHub** — это сервис для совместной разработки и хостинга проектов. C помощью GitHub над кодом проекта может работать неограниченное количество программистов из любых точек мира. В GitHub есть система контроля (управления) версий Git: сервис позволяет просматривать и контролировать любые изменения кода любым разработчиком и возвращаться к состоянию до изменений.

В целом GitHub — это социальная сеть для разработчиков, в которой можно найти проекты с открытым кодом от других разработчиков, практиковаться в написании кода и хранить свое портфолио.

**Проекты в GitHub**

Проект в GitHub хранится в репозитории (repository) — коллекции всех изменений создаваемого кода. Если вы будете работать над проектом в одиночку — вам нужно создать новый репозиторий. Если в вашем проекте несколько разработчиков — каждый из них будет клонировать репозиторий первоначального создателя проекта.

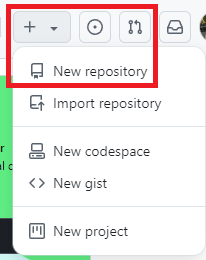
Внутри репозитория изменения кода хранятся в виде веток и коммитов.

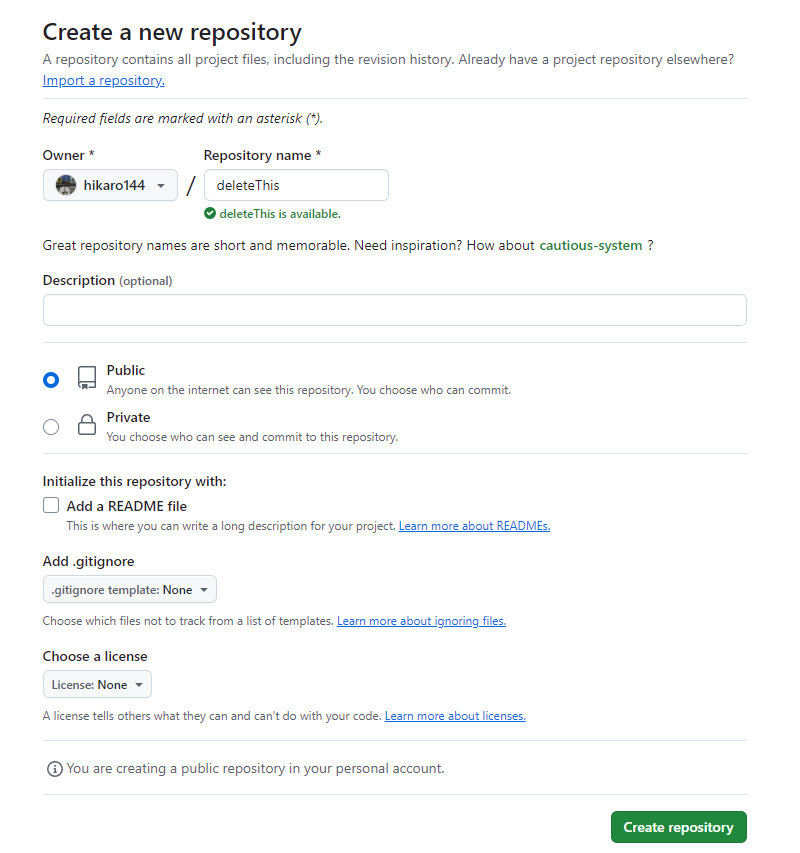
*Коммит (commit)* — основной объект разработки, в котором хранятся все изменения кода за итерацию. По сути, это список со всеми актуальными изменениями и ссылка на предыдущую версию коммита. У каждого коммита есть атрибуты: имя, дата создания, автор и комментарии к текущей версии (например, «Создал страницу courses.html» при разработке сайтов с видеокурсами).

*Ветка (branch)* — указатель на коммит с определенными изменениями. Например, два разработчика взяли коммит, и каждый из них сделал свои изменения в коде, создав по новому коммиту («Создал страницу coursеs.html c личным кабинетом» и «Создал страницу courses.html со свободным доступом на курсы»). Так в проекте появились две ветки с разным кодом: разработчик может выбрать, над каким коммитом ему работать дальше.

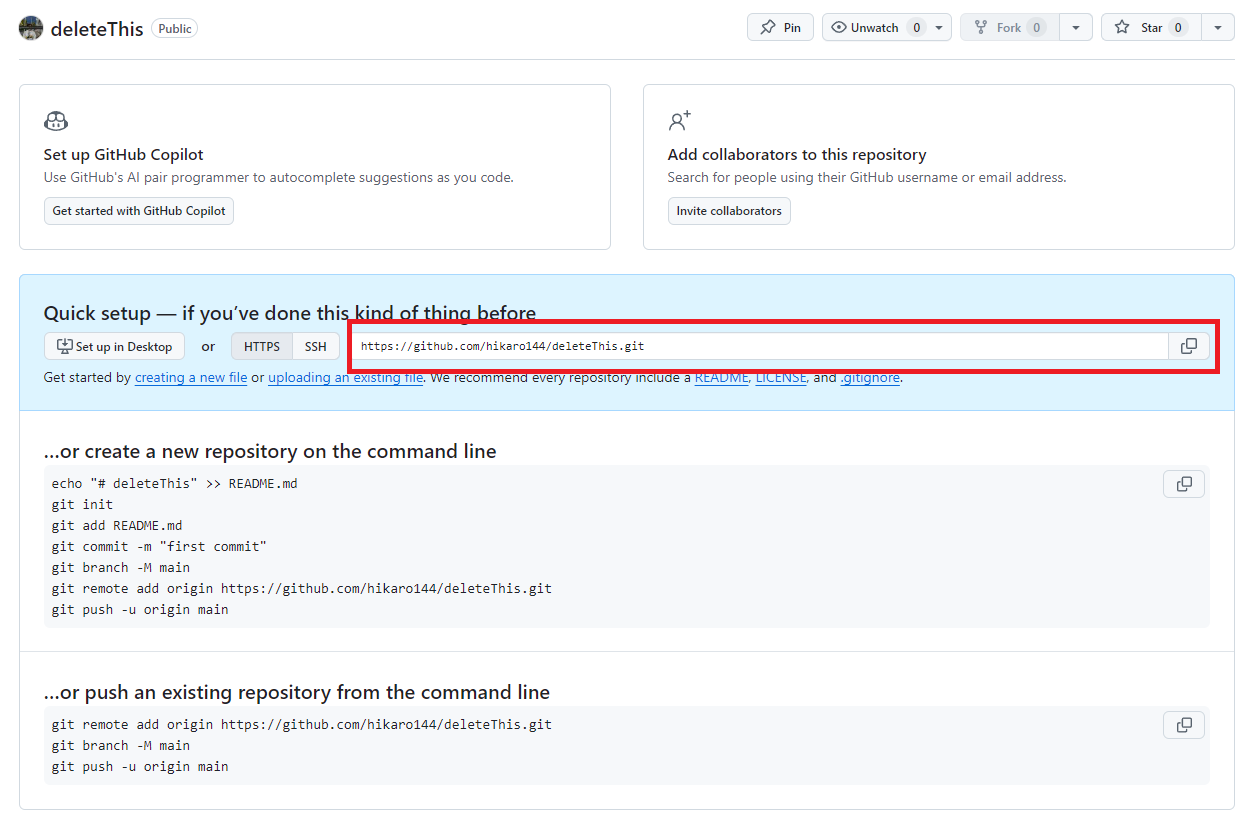
Основной веткой проекта, как правило, считается ветка main или master — разработчики создают новые ветки на ее основе. Также можно создать неограниченное количество веток, чтобы вносить новые изменения, не мешая основному проекту.

1. Перейти на сайт GitHub (<https://github.com/>)
2. Зарегистрироваться
3. Создать репозиторий





1. Скопировать на него ссылку



**Продолжаем**

git clone Ссылка на ваш репозиторий

**При клонировании в текущий каталог, там будет создана папка, в которую поместятся все проектные файлы** и скрытая директория .git, с самим репозиторием, или с необходимой информацией о нем. В такой ситуации, для клонируемого репозитория, по умолчанию, будет создана папка с одноименным названием, но его можно залить и в другую директорию, **например**:

git clone https://github.com/tutorialzine/awesome-project new-folder

**Подключение к удаленному репозиторию**

Чтобы загрузить что-нибудь в удаленный репозиторий, сначала нужно к нему подключиться. Регистрация и установка может занять время, но все подобные сервисы предоставляют хорошую документацию.

Чтобы связать наш локальный репозиторий с репозиторием на GitHub, выполним следующую команду в терминале. **Обратите внимание, что нужно обязательно изменить URI репозитория на свой.**

git remote add origin [[здесь ваша ссылки с github]](https://github.com/tutorialzine/awesome-project.git)

Проект может иметь несколько удаленных репозиториев одновременно. Чтобы их различать, мы дадим им разные имена. Обычно главный репозиторий называется origin.

**Отправка изменений на сервер**

Сейчас самое время переслать наш локальный коммит на сервер. Этот процесс происходит каждый раз, когда мы хотим обновить данные в удаленном репозитории.

Команда, предназначенная для этого - push. Она принимает два параметра: имя удаленного репозитория (мы назвали наш origin) и ветку, в которую необходимо внести изменения (master — это ветка по умолчанию для всех репозиториев).

git push origin master

Должно вернуться что-то вроде:

Counting objects: 3, done.

Writing objects: 100% (3/3), 212 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/tutorialzine/awesome-project.git

\* [new branch] master -> master

Эта команда немного похожа на git fetch, с той лишь разницей, что при помощи fetch мы импортируем коммиты в локальную ветку, а применив push, мы экспортируем их из локальной в удаленную. Если вам необходимо настроить удаленную ветку используйте git remote. Однако пушить надо осторожно, ведь рассматриваемая команда перезаписывает безвозвратно все изменения. В большинстве случаев, ее используют, чтобы опубликовать выгружаемые локальные изменения в центральный репозиторий. А еще ее применяют для того, чтобы поделиться, внесенными в локальный репозиторий, нововведениями, с коллегами или другими удаленными участниками разработки проекта. Подытожив сказанное, можно назвать git push - командой выгрузки, а git pull и git fetch - командами загрузки или скачивания. После того как вы успешно запушили измененные данные, их необходимо внедрить или интегрировать, при помощи команды слияния git merge.

В зависимости от сервиса, который вы используете, вам может потребоваться аутентифицироваться, чтобы изменения отправились. **Если все сделано правильно, то, когда вы посмотрите в удаленный репозиторий при помощи браузера, вы увидите файл hello.txt**

**Запрос изменений с сервера**

Если вы сделали изменения в вашем удаленном репозитории, другие пользователи могут скачать изменения при помощи команды pull.

git pull origin master

Вернется что-то вроде:

From https://github.com/tutorialzine/awesome-project

\* branch master -> FETCH\_HEAD

Already up-to-date.

Так как новых коммитов с тех пор, как мы склонировали себе проект, не было, никаких изменений доступных для скачивания нет.

**Создание новой ветки**

Основная ветка в каждом репозитории называется master. Чтобы создать еще одну ветку, используем команду branch <name>

git branch newBranchNewPractice

Это создаст новую ветку, пока что точную копию ветки master.

**Переключение между ветками**

Сейчас, если мы запустим branch, мы увидим две доступные опции:

git branch

Вернется:

newBranchNewPractice

\* master

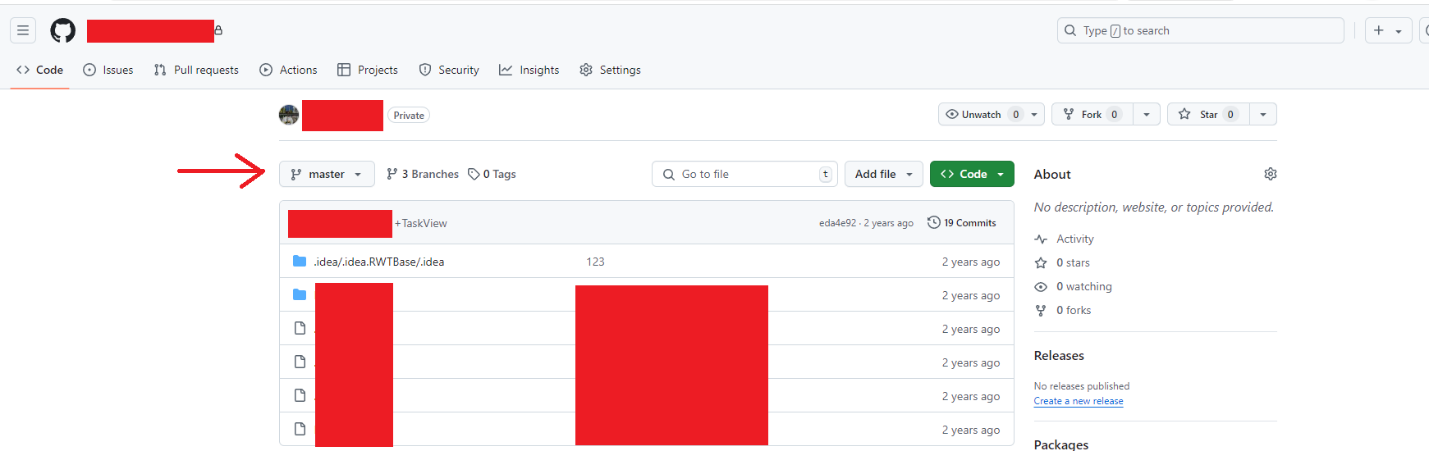
master — это активная ветка, она помечена звездочкой. Но мы хотим работать с нашей «newBranchNewPractice», так что нам понадобится переключиться на другую ветку. Для этого воспользуемся командой **checkout**, она принимает один параметр — имя ветки, на которую необходимо переключиться.

git checkout newBranchNewPractice

В Git ветка — это отдельная линия разработки. Git checkout позволяет нам переключаться как между удаленными, так и меду локальными ветками. Это один из способов получить доступ к работе коллеги или соавтора, обеспечивающий более высокую продуктивность совместной работы. Однако тут надо помнить, что пока вы не закомитили изменения, вы не сможете переключиться на другую ветку. В такой ситуации нужно либо сделать коммит, либо отложить его, при помощи команды git stash, добавляющей текущие незакоммиченные изменения в стек изменений и сбрасывающей рабочую копию до HEAD'а репозитория.

**Создайте новый текстовый файл и по командам выше занесите его в новую ветку, проверьте ее на гитхаб.**

**(Ветки меняются вот тут)**

****

**Самостоятельное задание**

1. Разметить красиво и грамотно readme

Подсказка на шаблоны - <https://github.com/blademoon/Markdown>

1. Сделать слияние веток и описать зачем это нужно

