**Лабораторная работа №1**

**Работа с Git и GitHub**

**Цели и задачи работы**: изучение платформы GitHub и практическое использование Git для хранения, обновления и распространения исходного кода проекта.

**Задание к работе**: Разработать алгоритм решения задачи по индивидуальному заданию.

**Методика выполнения работы**:

1. Разработать алгоритм решения задачи по индивидуальному заданию.
2. Написать и отладить программу решения задачи.
3. Протестировать работу программы на различных исходных данных.
4. Продемонстрировать работу с Git и GitHub на примере написанной программы.
5. По запросу преподавателя быть готовым модифицировать/добавить функционал программы.
6. Ответить на теоретические вопросы к лабораторной работе на выбор преподавателя.

**Состав отчета**:

1. Титульный лист.
2. Цель работы, задание по варианту.
3. Листинг программы.
4. Результат работы программы.
5. Полное описание всех этапов процесса работы с Git и GitHub на примере написанной программы (клонирование, загрузка в репозиторий и т.д.). В отчет обязательно нужно приложить ссылку на репозиторий в GitHub.

**Подготовить ответы на вопросы**:

1. Что такое Git и Github?

* Git — это распределённая система управления версиями: есть один сервер, через который разработчики обмениваются кодом. Разработчик копирует (клонирует) проект к себе на локальную машину, делает изменения и сохраняет их на удалённый сервер. При необходимости другие разработчики могут скопировать эти изменения к себе.
* GitHub — крупнейший веб-сервис, который позволяет заниматься совместной разработкой с использованием Git и сохранять изменения на своих серверах. На самом деле [функциональность](https://github.com/about) GitHub намного больше, но сейчас нас интересует только совместная разработка и история изменений. Ещё есть [Gitlab](https://about.gitlab.com/" \t "_blank), [Bitbucket](https://bitbucket.org/product/ru/guides/basics/bitbucket-interface" \t "_blank) и другие, но мы будем использовать GitHub как самый популярный в настоящее время.

1. В чем состоит разница между Git и Github?

* Git — это инструмент, который разработчики используют для управления исходным кодом. Он работает локально на компьютере разработчика и не требует подключения к интернету.
* GitHub, с другой стороны, — это облачный сервис, который использует Git в качестве основы. GitHub предоставляет графический интерфейс для работы с Git, а также облачное хранилище для проектов. GitHub также предлагает функции для совместной работы и управления проектами.

Итак, вкратце, Git — это инструмент, а GitHub — это сервис, который использует этот инструмент.

1. Какими основными особенностями обладает Git?

* Ключевая особенность Git — ветвление, позволяющее работать над разными версиями проекта.

1. Что такое репозиторий?

* Репозиторий в контексте Git — это место, в котором хранится весь код и вся история его изменений.
* **Репозиторий (repository)** — директория проекта, который отслеживается Git. В директории хранится проект, история изменений и мета-информация проекта (в скрытой директории .git).

1. Как создать пустой репозиторий?

* **git init** — создание репозитория. Команда **git init** создает в директории пустой репозиторий в виде директории **.git**, где и будет в дальнейшем храниться вся информация об истории коммитов, тегах — о ходе разработки проекта.

1. Что такое ветка репозитория?

* Ветка репозитория — это версия репозитория, или независимое направление разработки.
* **Ветка (branch)** — последовательность коммитов. По сути — ссылка на последний коммит в этой ветке. Ветки не зависят друг от друга — можно вносить изменения в одну, и они не повлияют на другую (если вы явно этого не попросите).

1. Какова цель ветвления в Git?

* Ветки не зависят друг от друга — можно вносить изменения в одну, и они не повлияют на другую. Позволяет проверить изменение без затрагивания основной ветки.
* Основной идеей ветвления является отклонение от основного кода и продолжение работы независимо от него. Также это удобно в тестировании отдельного функционала, потому что позволяет работать над новой частью кода, не беспокоясь о поломке чего-то в рабочей версии.

1. Как переключаться между ветками?

* Чтобы переключаться между ветками в Git, используется команда **git checkout**, после которой указывается имя ветки.

1. Что делает команда git config?

* При установке Git была добавлена утилита **git config**, которая позволяет просматривать и изменять большинство параметров работы Git’а. Если речь о данных пользователя или способе работы репозитория — **git config** будет самым удобным способом настройки.

1. Что делает команда git status?

* **git status** — показывает текущее состояние файлов в репозитории (какие файлы изменились, удалились, добавились).

1. Как добавить файлы в репозиторий с помощью Git и Github?
2. Что такое 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.

1. Какими способами можно осуществить клонирование? https://thecode.media/git-clone/

|  |  |
| --- | --- |
| Клонируем репозиторий, используя протокол http | git clone http://user@somehost:port/~user/repository/project.git |
| Клонируем репозиторий с той же машины в директорию myrepo: | git clone /home/username/project myrepo |
| Клонируем репозиторий, используя безопасный протокол ssh: | git clone ssh://user@somehost:port/~user/repository |
| У git имеется и собственный протокол: | git clone git://user@somehost:port/~user/repository/project.git/ |
| Импортируем svn репозиторий, используя протокол http: | git svn clone -s http://repo/location |

* Командная строка — самый гибкий и универсальный способ, особенно если часто работаем с Git.
* Git GUI — подойдёт, если хочется визуально управлять репозиториями без команд.
* Visual Studio — удобный вариант, если используете VS для разработки и хотите интегрировать работу с Git прямо в IDE.

1. Что такое коммит?

* Коммит (commit) — это фиксация изменений в истории проекта (изменения, которые внесены в индекс). Коммит хранит изменённые файлы, имя автора коммита и время, в которое был сделан коммит. Кроме того, каждый коммит имеет уникальный идентификатор, который позволяет в любое время к нему откатиться. Можете считать коммит этакой точкой сохранения.
* **git log** — показывает историю изменений (это про зафиксированные изменения, то есть коммиты).
* Зафиксировать (закоммитить) изменения — git commit.

1. Что такое слияние веток? С помощью какой команды оно осуществляется?

* Слияние — обычная практика для разработчиков, использующих системы контроля версий. Независимо от того, созданы ли ветки для тестирования, исправления ошибок или по другим причинам, слияние фиксирует изменения в другом месте. Слияние принимает содержимое ветки источника и объединяет их с целевой веткой. В этом процессе изменяется только целевая ветка. История исходных веток остается неизменной.

1. Что такое конфликт в Git? Как его разрешить?

* https://docs.github.com/ru/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/about-merge-conflicts

1. Что делает команда git push?

* Синхронизировать изменения с сервером — git push.

1. Что делает команда git stash?

* Команда **git stash** позволяет временно сохранить изменения, сделанные в рабочей копии, чтобы можно было применить их позже.

1. Как посмотреть историю изменений в GitHub?

* Каталог logs хранит в себе историю проекта для всех веток в вашем проекте.
* Просматривать изменения, внесённые в репозиторий, можно с помощью параметра **log**. Он отображает список последних коммитов в порядке выполнения. Кроме того, добавив флаг **-p**, вы можете подробно изучить изменения, внесённые в каждый файл.

1. Что делает команда git pull?

* **git pull** и *git fetch* — команды загрузки или скачивания.

1. Как работать с Git через Visual Studio?

* https://habr.com/ru/companies/microsoft/articles/216037/

**Литература**:

1. Официальная документация по GitHub - <https://docs.github.com/ru/get-started>
2. Статья на Хабре «Работаем с Git: первые шаги в GitHub» - <https://habr.com/ru/companies/yandex_praktikum/articles/700708/>
3. Видеоуроки по Git и GitHub - <https://www.youtube.com/playlist?list=PLuY6eeDuleIOMB2R_Kky05ZfiAx2_pbAH>
4. Статья «Git за полчаса: руководство для начинающих» - <https://proglib.io/p/git-for-half-an-houR?ysclid=lt5ifup6iu748174111>

**Процесс выполнения задания**:

1. Разбиться на группы по 2-3 человека.
2. Выбрать одно задание из предложенных (варианты заданий см. ниже).
3. Распределить работу над задачей между участниками группы.
4. Написать и отладить программу решения задачи. Язык программирования выбирается группой самостоятельно.
5. Создать репозиторий для проекта (наличие файла README обязательно). В README нужно отразить полное описание проекта.
6. Загрузить в репозиторий свою часть проекта в отдельную ветку.
7. Выполнить клонирование проекта.
8. Произвести изменения в своей части проекта и загрузить их в репозиторий в соответствующую ей ветку.
9. Выполнить слияние отдельной ветки с основной.
10. Выполнить обновление файлов проекта в случае внесения изменений данных другими участниками группы.
11. Показать историю изменений файлов в репозитории.

**Варианты заданий**:

1. Геометрические фигуры. Реализовать функции вычисления параметров для двух-трех следующих геометрических фигур (по одной фигуре на студента):
   1. Прямоугольник – периметр, площадь, длина диагонали;
   2. Треугольник – периметр, площадь по формуле Герона, проверка на равнобедренность;
   3. Трапеция – периметр, площадь, длина средней линии;
   4. Окружность – длина окружности, площадь круга, площадь кругового сектора.
2. Строки. Реализовать функции, связанные со строками (по одной функции на студента):
   1. Подсчет количества гласных букв в строке;
   2. Привидение всех букв к верхнему и нижнему регистру;
   3. Удаление всех знаков препинания и служебных символов (кроме пробела) из строки;
   4. Проверка, является ли строка палиндромом.