Лабораторная работа №1

Научное программирование

Минов Кирилл | НПМмд-02-23

Содержание

# 1 Цель работы

Изучить средство контроля версий, а также освоить умения по работе с git.

# 2 Теоретическое введение

**Системы контроля версий (Version Control System, VCS)** применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется.

В *классических системах контроля версий* используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. В отличие от классических, в *распределённых системах контроля версий* центральный репозиторий не является обязательным.

Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд.

Более подробно см. в [1]

# Выполнение лабораторной работы

1. Настройка github и установка git-flow(изначально в пакете установки)

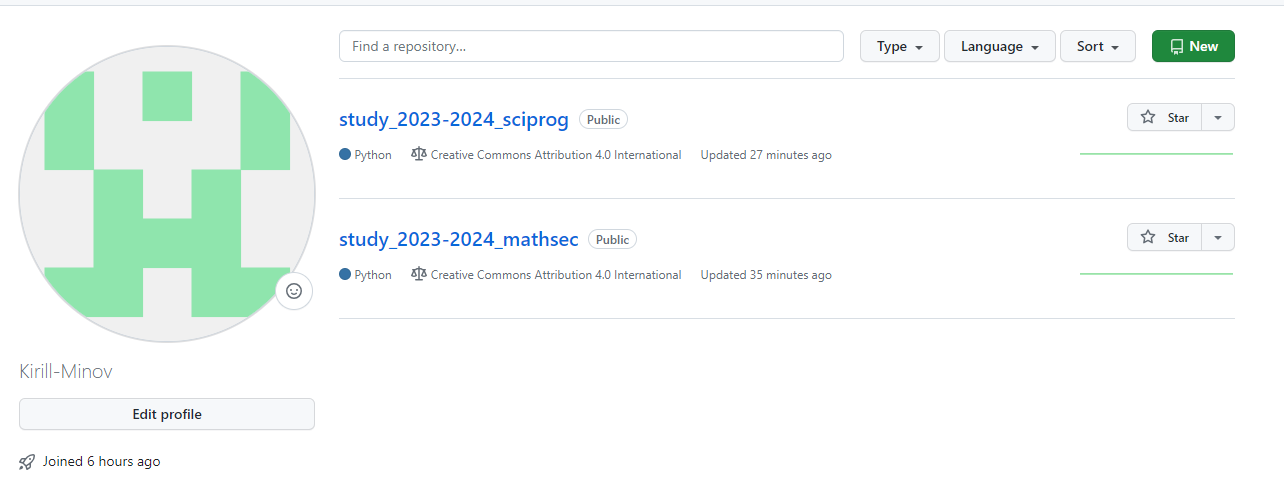


Рисунок 1(регистрация)

1. Базовая настройка git

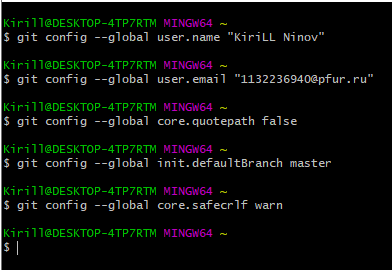


Рисунок 2(базовая настройка)

1. . Создайте ключи ssh + Создайте ключи pgp

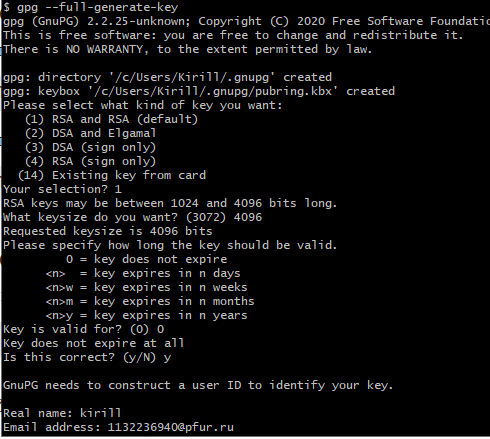


Рисунок3(создание ключей)

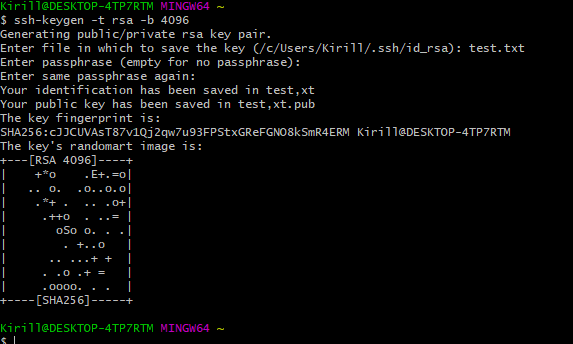


Рисунок4(создание ключей)

4) Добавление PGP ключа в GitHub

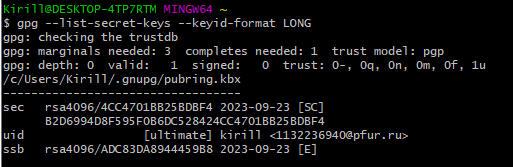


Рисунок5(добавление ключей в гитхаб)

5) Настройка автоматических подписей коммитов git + Настройка gh

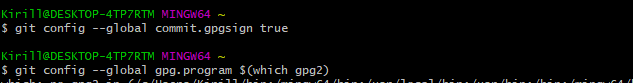


Рисунок6(настройка коммитов)

6) Шаблон для рабочего пространства + Сознание репозитория курса на основе шаблона сделано заранее и видно на первом рисунке

# 4 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? Системы контроля версий (VCS) - это программные инструменты, предназначенные для управления изменениями в коде и других ресурсах проекта во времени. Они позволяют разработчикам отслеживать, контролировать и координировать изменения в проекте, что делает их полезными для разнообразных задач: (Отслеживание изменений в коде, совместная работа, автоматическое резервное копирование, анализ изменений, управление версиями релизов)
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.

Хранилище (Repository): Хранилище в системе контроля версий (VCS) - это центральное хранилище, где хранятся все версии файлов и история изменений проекта. В хранилище сохраняются не только текущие версии файлов, но и информация о том, как эти файлы изменялись с течением времени. Оно может быть централизованным (один сервер, на котором хранятся все версии) или распределенным (каждый участник проекта имеет полную копию хранилища). Commit (Фиксация, Коммит):

Commit - это действие, при котором разработчик сохраняет текущее состояние своей рабочей копии проекта в системе контроля версий. Каждый коммит снабжается комментарием, который описывает, какие изменения были внесены в проект на этом этапе. Commit фиксирует изменения локально, и они становятся частью истории проекта в хранилище после синхронизации. История (History):

История в системе контроля версий представляет собой запись всех коммитов и изменений, сделанных в проекте с момента его создания. Она содержит информацию о том, кто и когда внес изменения, а также комментарии к коммитам, которые помогают понять, какие изменения были сделаны и почему. История позволяет отслеживать развитие проекта, возвращаться к предыдущим версиям и искать источники проблем. Рабочая копия (Working Copy):

Рабочая копия - это локальная копия проекта, доступная для работы над ним на компьютере разработчика. Это текущее состояние проекта, с которым разработчик взаимодействует и вносит изменения. Рабочая копия может быть изменена, и эти изменения можно фиксировать с помощью коммитов для обновления истории и хранилища.

1. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.

Централизованные (Centralized) и децентрализованные (Distributed) системы контроля версий (VCS) - это два основных вида VCS, которые отличаются способом управления и распределения версий проекта. Вот их основные характеристики и примеры:

Централизованные системы контроля версий (Centralized VCS):

Характеристики:

В централизованных VCS существует единственное центральное хранилище (репозиторий), в котором хранятся все версии файлов и история изменений. Разработчики работают с копией файлов из центрального хранилища, называемой "рабочей копией". Чтобы внести изменения, разработчики вынуждены синхронизироваться с центральным хранилищем. Примеры централизованных VCS:

Subversion (SVN): Это один из наиболее известных централизованных VCS. В SVN проект хранится в центральном репозитории, и разработчики получают рабочие копии для работы над проектом. Децентрализованные системы контроля версий (Distributed VCS):

Характеристики:

В децентрализованных VCS каждый участник проекта имеет свою полную копию репозитория, включая всю историю и версии файлов. Разработчики могут работать над проектом независимо, даже оффлайн, и синхронизировать изменения между своими репозиториями и другими участниками по необходимости. Эта архитектура делает децентрализованные VCS более гибкими и устойчивыми к сбоям сети. Примеры децентрализованных VCS:

Git: Git - это наиболее популярная децентрализованная система контроля версий. Каждый разработчик имеет свой собственный локальный репозиторий, и они могут вносить изменения, фиксировать коммиты и обмениваться изменениями между собой. Git также поддерживает хостинг на удаленных серверах, таких как GitHub, GitLab и Bitbucket, где можно обмениваться изменениями с другими разработчиками.

1. Опишите действия с VCS при единоличной работе с хранилищем. При единоличной работе с системой контроля версий (VCS) действия разработчика остаются простыми и не требуют множественных взаимодействий или синхронизации с другими участниками. Ниже описаны основные действия при единоличной работе с VCS:

Настройка репозитория:

Создайте новый репозиторий в системе контроля версий. Это может быть централизованный или децентрализованный репозиторий, в зависимости от выбранной системы (например, Git, SVN и т. д.). Инициализируйте локальный репозиторий на своем компьютере, связав его с удаленным репозиторием, если это необходимо. Работа с проектом:

Работайте над вашим проектом, внося изменения в исходный код, создавая новые файлы и папки, удаляя или изменяя существующие элементы. Добавление файлов:

Перед сохранением изменений в систему контроля версий, добавьте измененные файлы в "индекс" (также называемый "стэйджем") с помощью команды git add (для Git) или аналогичной команды в другой VCS. Это позволяет выбрать, какие изменения будут включены в следующий коммит. Коммит изменений:

Создайте коммит, фиксируя изменения, которые вы хотите сохранить в истории проекта. Каждый коммит снабжается описанием, которое объясняет, какие изменения были внесены. Используйте команду git commit (для Git) или аналогичную команду в другой VCS для выполнения коммита. Сохранение изменений в хранилище:

Синхронизируйте ваш локальный репозиторий с удаленным репозиторием (если это децентрализованная система) с помощью команды git push (для Git) или аналогичной команды в другой VCS. Это отправит ваши коммиты на сервер и обновит историю изменений. Обновление до актуальной версии (только для децентрализованных систем):

Периодически получайте обновления из удаленного репозитория с помощью команды git pull (для Git) или аналогичной команды в другой VCS, чтобы убедиться, что ваша локальная копия актуальна.

1. Опишите порядок работы с общим хранилищем VCS.

Работа с общим хранилищем (репозиторием) в системе контроля версий (VCS) предполагает совместную работу нескольких разработчиков над проектом. Вот основные шаги и порядок работы с общим хранилищем в VCS:

Настройка репозитория:

Создание репозитория: Один из разработчиков создает центральное хранилище (репозиторий) в VCS. Это может быть удаленный сервер (например, GitHub, GitLab, Bitbucket) или локальный сервер, доступный для всех участников проекта.

Установка прав доступа: Устанавливаются права доступа и разрешения на репозиторий, чтобы определить, кто имеет право читать, записывать и управлять кодом в хранилище.

Клонирование репозитория:

Остальные разработчики клонируют (копируют) репозиторий на свои локальные компьютеры с использованием команды git clone (для Git) или аналогичной команды в другой VCS. Клонирование создает локальную копию репозитория, которую разработчик может изменять и синхронизировать с центральным хранилищем. Работа над проектом:

Каждый разработчик работает над проектом, вносит изменения в код, добавляет новые функции и решает задачи. Добавление и коммит изменений:

После внесения изменений, разработчики добавляют файлы с изменениями в "индекс" (стэйдж) с помощью команды git add (для Git) или аналогичной команды в другой VCS.

Затем они создают коммиты, фиксируя изменения с комментариями, описывающими, что было сделано, с помощью команды git commit (для Git) или аналогичной команды в другой VCS. Коммиты сохраняются в истории изменений.

Обновление до актуальной версии:

Перед началом работы и перед завершением работы над проектом, разработчики обновляют свои локальные копии репозитория до актуальной версии с помощью команды git pull (для Git) или аналогичной команды в другой VCS. Это позволяет синхронизировать свои изменения с изменениями, внесенными другими участниками проекта. Отправка изменений в репозиторий:

Разработчики отправляют свои локальные изменения (коммиты) в центральный репозиторий с помощью команды git push (для Git) или аналогичной команды в другой VCS. Это обновляет центральное хранилище и делает изменения доступными для других участников проекта. Работа с ветками:

Разработчики могут создавать ветки (branches) для изоляции новых функций и задач. Ветки позволяют разрабатывать функциональность независимо и объединять её обратно в основную ветку после завершения работы. Решение конфликтов :

Если несколько разработчиков внесли изменения в одни и те же части кода и есть конфликты, разработчики должны решить эти конфликты, прежде чем объединить изменения в центральном репозитории.

1. Каковы основные задачи, решаемые инструментальным средством git?

Отслеживание изменений: Git позволяет отслеживать изменения в исходном коде и других файлах, включая добавление, удаление и изменение файлов.

История изменений: Git сохраняет историю всех изменений в проекте, включая информацию о том, кто и когда внес изменения, и комментарии к каждому коммиту.

Создание веток (Branches): Разработчики могут создавать ветки для изоляции новых функций или задач. Это позволяет разрабатывать функциональность независимо и объединять её обратно в основную ветку после завершения работы.

Слияние (Merge): Git обеспечивает механизм слияния изменений из одной ветки в другую. Это позволяет объединить код, разработанный в разных ветках, в одну общую версию проекта.

Резервное копирование (Backup): Git сохраняет копии всех версий файлов и историю изменений, что делает его надежным инструментом для создания резервных копий проекта.

Управление конфликтами (Conflict Resolution): Когда несколько разработчиков вносят изменения в одни и те же части кода, могут возникать конфликты. Git предоставляет механизмы для разрешения этих конфликтов.

Сотрудничество (Collaboration): Git поддерживает совместную работу нескольких разработчиков над одним проектом. Он позволяет разработчикам синхронизировать изменения, обмениваться кодом и обсуждать изменения в репозитории.

Версионирование релизов (Versioning): Git позволяет управлять версиями проекта и выпусками программного обеспечения, что полезно при создании стабильных релизов.

Откат к предыдущим версиям (Rollback): Git позволяет вернуться к предыдущим версиям проекта, если что-то пошло не так, или если необходимо откатиться к более стабильной версии.

Отслеживание изменений в удаленных репозиториях (Remote Tracking): Git позволяет отслеживать изменения в удаленных репозиториях и интегрировать их в локальную копию проекта.

1. Назовите и дайте краткую характеристику командам git. git init:

Команда git init создает новый репозиторий Git в текущем каталоге. Эта команда инициализирует Git и позволяет начать отслеживать изменения в проекте. git clone:

Команда git clone создает копию удаленного репозитория на локальном компьютере. Это позволяет разработчикам получить доступ к проекту и начать работу с ним. git add:

Команда git add добавляет измененные файлы в индекс (стэйдж) для подготовки к коммиту. Она позволяет выбирать, какие изменения будут включены в следующий коммит. git commit:

Команда git commit фиксирует изменения в индексе в истории проекта. При выполнении коммита необходимо добавить комментарий, описывающий изменения. git status:

Команда git status показывает текущее состояние рабочей копии и индекса. Она позволяет узнать, какие файлы были изменены и готовы к коммиту. git log:

Команда git log выводит историю коммитов проекта. Она отображает информацию о каждом коммите, включая автора, дату и комментарий. git branch:

Команда git branch позволяет управлять ветками проекта. Она позволяет создавать, переключаться между ветками и удалять их. git merge:

Команда git merge выполняет слияние изменений из одной ветки в другую. Она используется для объединения разных веток в одну.

1. Приведите примеры использования при работе с локальным и удалённым репозиториями. Работа с локальным репозиторием:

Инициализация нового локального репозитория(git init) Добавление и фиксация изменений(git add) Просмотр истории коммитов(git log)

Работа с удаленным репозиторием Клонирование удаленного репозитория(git clone) Добавление удаленного репозитория(git remote add) Получение и отправка изменений(git pull) 15. Что такое и зачем могут быть нужны ветви (branches)? Ветви (branches) в системах контроля версий, таких как Git, представляют собой параллельные линии разработки, которые позволяют разработчикам работать над различными функциями, исправлениями или задачами независимо друг от друга. Ветви очень полезны и играют важную роль в управлении и организации разработки программного обеспечения

1. Как и зачем можно игнорировать некоторые файлы при commit?

Секретные данные: Вы можете хранить конфиденциальные данные, такие как пароли, ключи API или файлы с настройками, в файлах, которые должны быть исключены из контроля версий.

Временные файлы: Временные файлы, такие как файлы журналов, файлы созданные средствами отладки или файлы, создаваемые в процессе сборки проекта, могут быть исключены, чтобы не загромождать репозиторий.

Бинарные файлы: Бинарные файлы, такие как изображения, видео и бинарные исполняемые файлы, часто не должны храниться в репозитории, так как они могут увеличить размер репозитория и замедлить операции с ним.

# 5 Вывод

В ходе выполнения данной лабораторной работы я изучил средство контроля версий, а также освоил умения по работе с git.

# Список литературы

1. Cистемы контроля версий [Электронный ресурс]. 2023. URL: <https://apptractor.ru/info/articles/chto-takoe-sistema-kontrolya-versiy.html>.