Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский экономический университет имени Г.В. Плеханова»

**МОСКОВСКИЙ ПРИБОРОСТРОИТЕЛЬНЫЙ ТЕХНИКУМ**

специальность 09.02.07 «Информационные системы и программирование»

Квалификация: Программист

ПРАКТИЧЕСКИЕ РАБОТЫ

ПО МДК 04.02 «Обеспечение качества функционирования компьютерных систем»

|  |  |
| --- | --- |
| Выполнил студент  группы П50-5-21  Сбродов Е.Д. | Проверил преподаватель  \_\_\_\_\_\_\_\_\_\_\_ К.С. Образцова  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 года |

Москва 2024

Содержание

[Практическая работа №1 3](#_Toc177338922)

# Практическая работа №1



Waterfall

Каждый этап один за другим – нельзя вернуться назад. Этапы циркулируют только после завершения последнего этапа.



SCRUM

В конце спринта имеется готовый продукт. Гибкое изменение требование. Быстрое реагирование на изменение трендов. Бюджет не фиксирован.

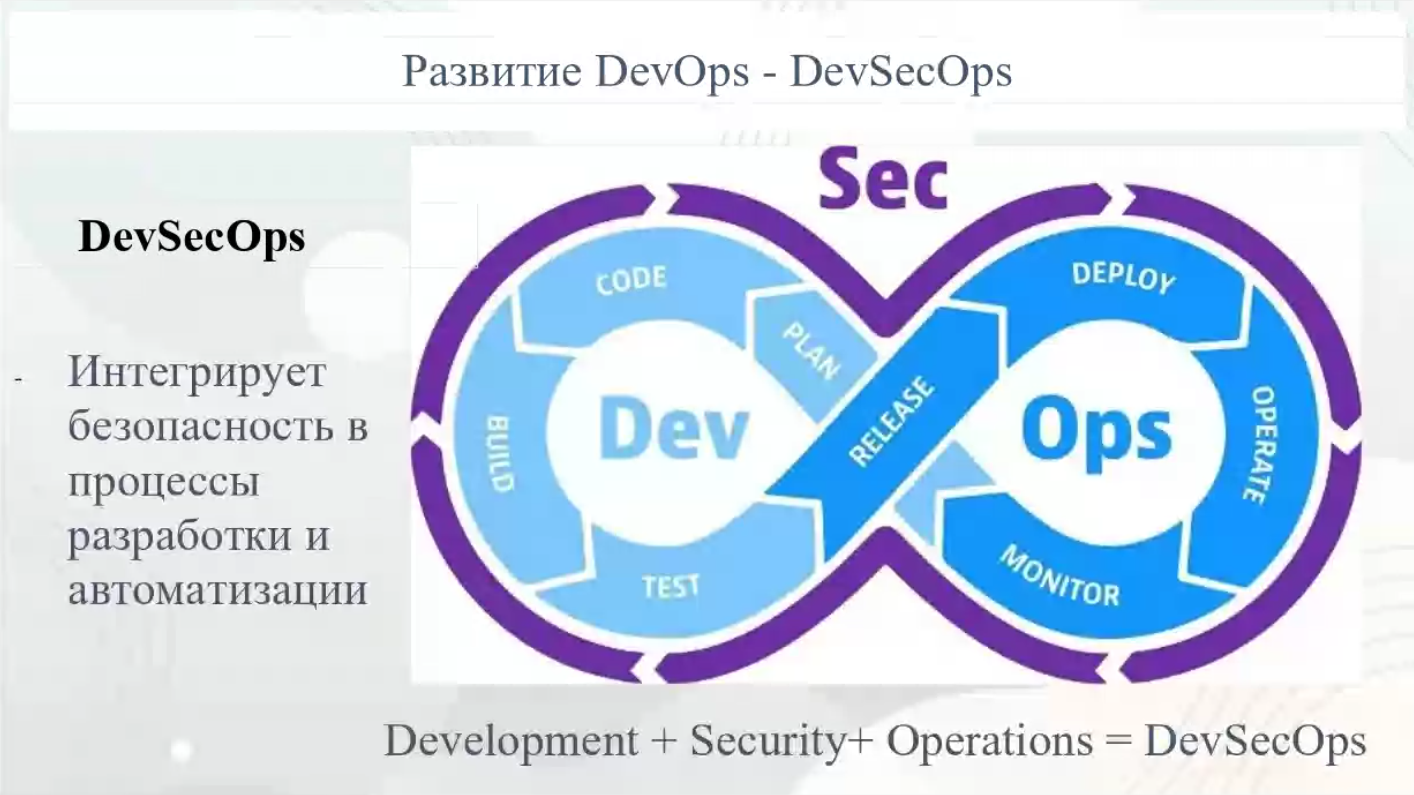
Основные этапы разработки ПО

* Анализ
* Разработка
* Сборка
* Тестирование
* Развертывание
* Релиз (деплой)
* Сопровождение
* Мониторинг



DevOps

Бесперебойная поставка ПО с помощью непрерывной интеграции рабочих процессов. - Ускорение этапов разработки –Снижение ошибок, связанных с человеческим фактором



DevSecOps

Интегрирует безопасность в процессы разработки и автоматизации

Концепции составляющие DevOps

Инфраструктура как код

Основная идея заключается в том, что вся инфраструктура, такая как виртуальные машины, сетевые настройки, хранилища данных и другие ресурсы, описывается в виде кода, который может быть управляем исходными файлами, хранящимися в системе контроля версий.

CI/CD

Цель CI/CD состоит в том, чтобы обеспечить быстрое время от идеи до развертывания новых функций или исправлений ошибок, с учетом меньшей вероятности ошибок и конфликтов, а также поддержания высокого уровня качества кода и процесса разработки.

Состав

-Непрерывная Интеграция

-Непрерывная доставка

-Непрерывное развертывание

Что такое система контроля версий и для чего она нужна

Система контроля версий- это специальное программное обеспечение, которое используется для управления изменениями в файловой системе, отслеживания и контроля версий документов или кода программы. Она позволяет разработчикам и команде проекта работать над файлами вместе, отслеживать изменения, управлять их версиями и восстанавливаться к предыдущим состояниям проекта.

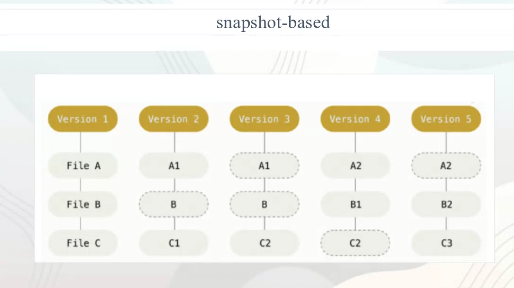
Преимущества применения системы контроля версий

* История изменений
* Контроль версий
* Коллективная работа
* Ветвление и слияние
* Отслеживание ошибок

Распределенная VS Централизованная системы контроля версий

Архитектура: Централизованная система контроля версий имеет единую центральную базу данных, где хранятся все файлы и изменения. В распределенной системе каждый пользователь имеет собственную копию репозитория, включая всю историю изменений.

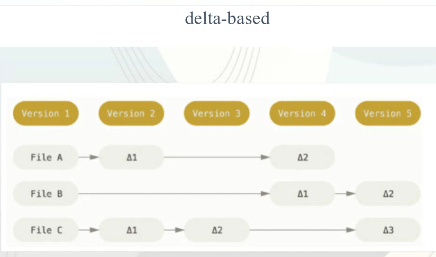
Работа в офлайн-режиме: Распределенная система позволяет работать в офлайн-режиме, так как каждый пользователь имеет полную копию репозитория. В централизованной системе требуется подключение к центральному серверу для выполнения большинства операций.



snapshot-based

Система контроля версий с механизмом снимков

(например, Git, Mercurial) хранят все версии файлов и директорий в форме полных копий (снимков) состояния проекта в определенный момент времени. Когда вы делаете коммит, система фиксирует все измененные файлы и создает новый снимок, который включает в себя полные копии всех измененных файлов. Это позволяет быстро переключаться между разными версиями проекта и осуществлять обход старых коммитов.



delta-based

Система контроля версий со списком изменений (например,

Subversion) хранят только разницу (delta) между последующими версиями файлов. Когда вы деласте коммит, система сохраняет только изменения, сделанные в файлах, файлах, относительно

предыдущего коммита, в виде списка изменений. На самом деле, система часто хранит несколько версий файла, чтобы быстрее обрабатывать запросы, основанные на списке изменений. При переключении на предыдущие версии проекта, СКВ применяет все изменения последовательно, чтобы восстановить запрошенную версию проекта.

Коммит

Коммит (commit) представляет собой операцию, при которой изменения в файловой системе, внесенные программистом, сохраняются в репозитории проекта. Коммит фиксирует изменения в файле или наборе файлов, а также добавляет комментарии или описания к этим изменениям

Система контроля версий Git

Git - это распределенная система управления версиями, которая используется для отслеживания изменений в коде программного обеспечения. Git позволяет разработчикам работать над проектом одновременно и совместно, сохраняя историю изменений и легко управляя версиями

Работа с гитом:

Цель работы: настроить конфигурацию гит, научиться создавать локальные репозитории, научиться работать с файлами в данном репозиториям, а также просматривать и перемещаться по коммитам.

Для начала чтобы узнать установлен ли у нас git и узнать его версию вписываем git –version



Рисунок 1 - версия гита

Потом для того чтобы гит адекватно выводил латиницу вписываем git config --global core.quotePath false



Рисунок 2 - перевод на латиницу

Чтобы узнать конфигурацию гита вписываем следующую команду git config --list

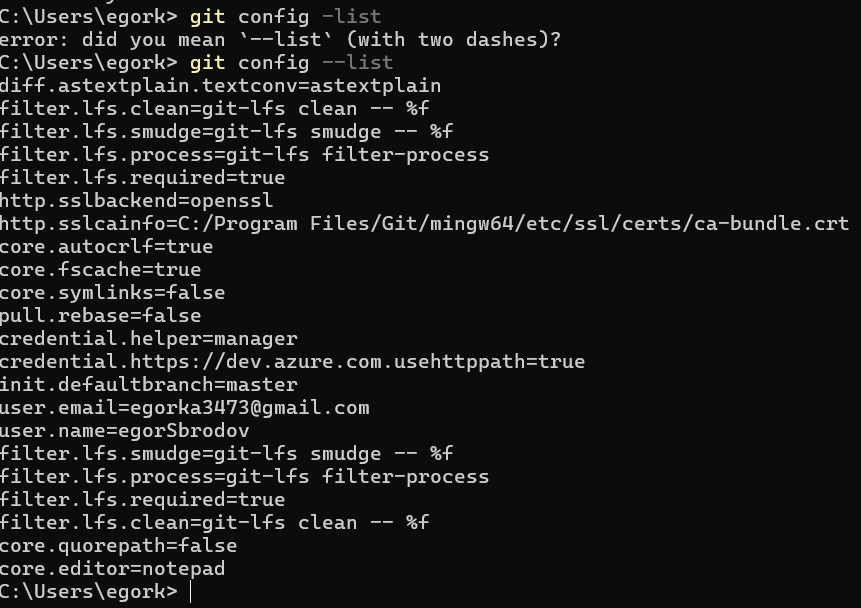


Рисунок 3 - конфигурация гита

Создаем папку для наших экспериментов над гитом и переходим в нее



Рисунок 4 - папка для обучения

После создаем локальный репозиторий командой git init

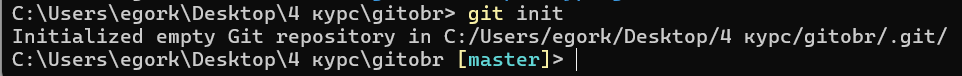


Рисунок 5 - создание локального репозитория

Узнаем состояние репозитория с помощью git status

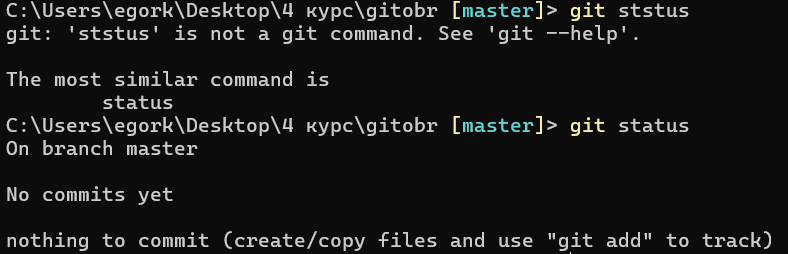


Рисунок 6 - статус гита

Создаем файлы для коммита

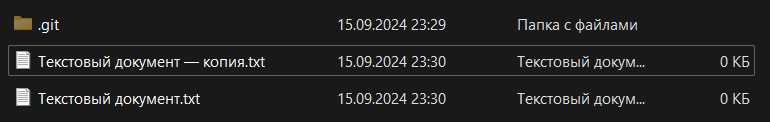


Рисунок 7 - создание файлов

Добавляем их в коммит git add



Рисунок 8 - добавление файла

Команда git add . добавляет все файлы в папке в коммит



Рисунок 9 - добавление всех файлов

И наконец создаем коммит с помощью команды git commit -m "описание коммита"

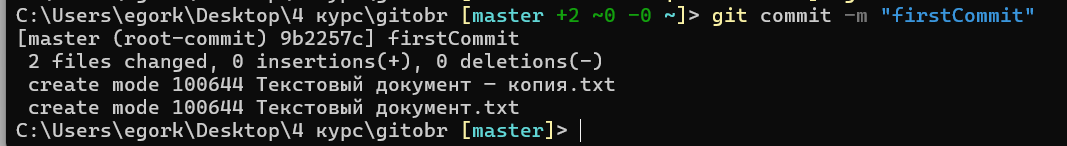


Рисунок 10 - создание коммита

Создаем еще файл в папке

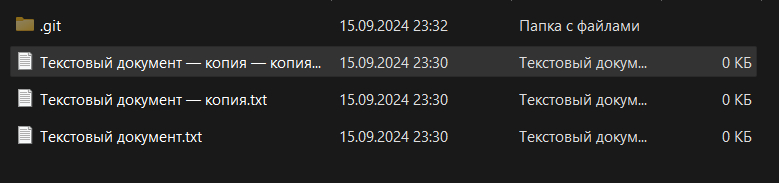


Рисунок 11 - создание файла

С помощью команды git diff можно узнать разницу между зафиксированными и не зафиксированными изменениями



Рисунок 12 - просмотр разницы

Добавляем файл в коммит и создаем его

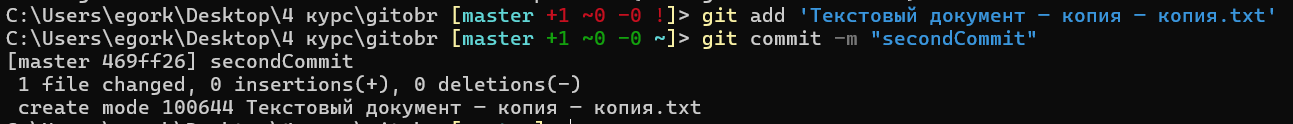


Рисунок 13 - создание нового коммита

С помощью команды git log узнаем нашу историю коммитов

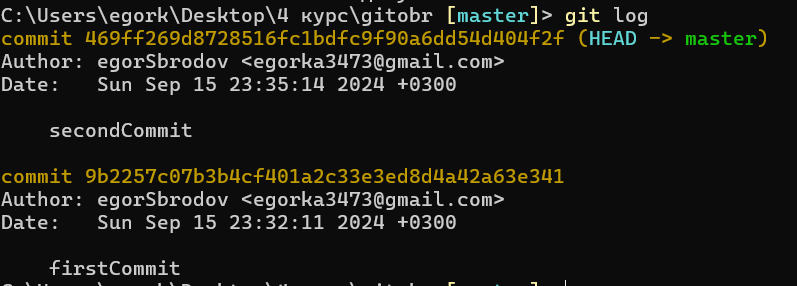


Рисунок 14 - просмотр коммитов

С помощью команды git reset --mixed 9b2257c07b3b4cf401a2c33e3ed8d4a42a63e341 переходим к коммиту с идентификатором 9b2257c07b3b4cf401a2c33e3ed8d4a42a63e341, при этом переводя все остальные коммиты в не отслеживаемую зону



Рисунок 15 - переход к другому коммиту

Видим что в логе у нас остался один коммит

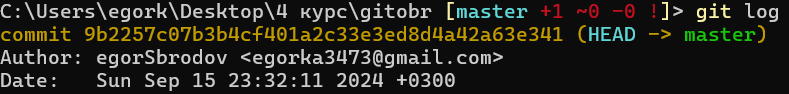


Рисунок 16 - просмотр коммитов

Добавляем незафиксированные файлы в коммит и создаем его. При этом номер нового коммита отличается.

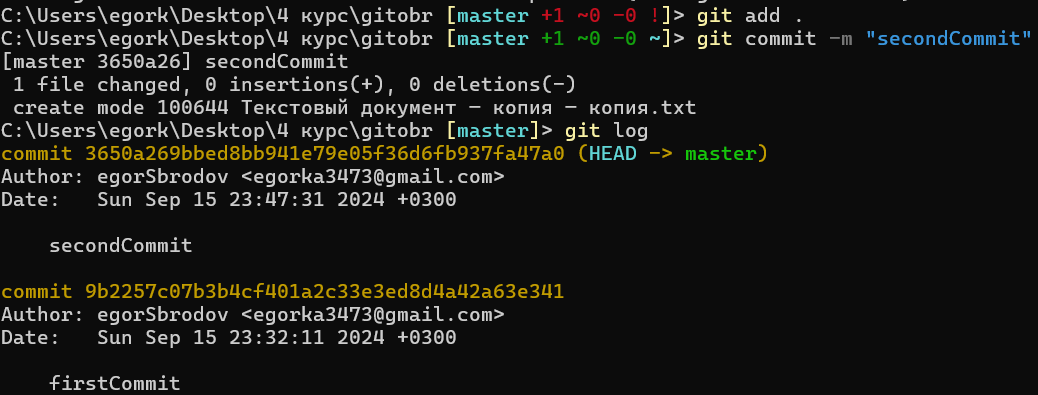


Рисунок 17 создание и просмотр коммитов

С помощью команды git reset --soft 9b2257c07b3b4cf401a2c33e3ed8d4a42a63e341 переходим к коммиту с идентификатором 9b2257c07b3b4cf401a2c33e3ed8d4a42a63e341, при этом переводя все остальные коммиты в отслеживаемую зону

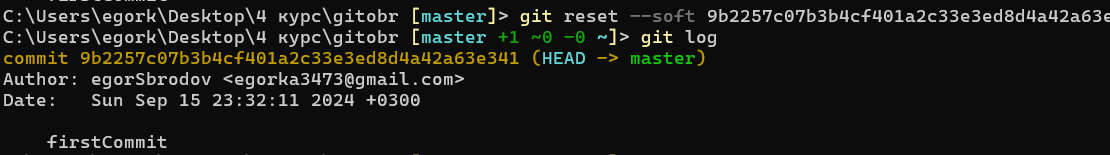


Рисунок 18 - переход к другому коммиту

После этого создаем новый файл, добавляем его в коммит и делаем его. С помощью команды git reset 9b2257c07b3b4cf401a2c33e3ed8d4a42a63e341 переходим к коммиту с идентификатором 9b2257c07b3b4cf401a2c33e3ed8d4a42a63e341, при этом удаляя все остальные коммиты.

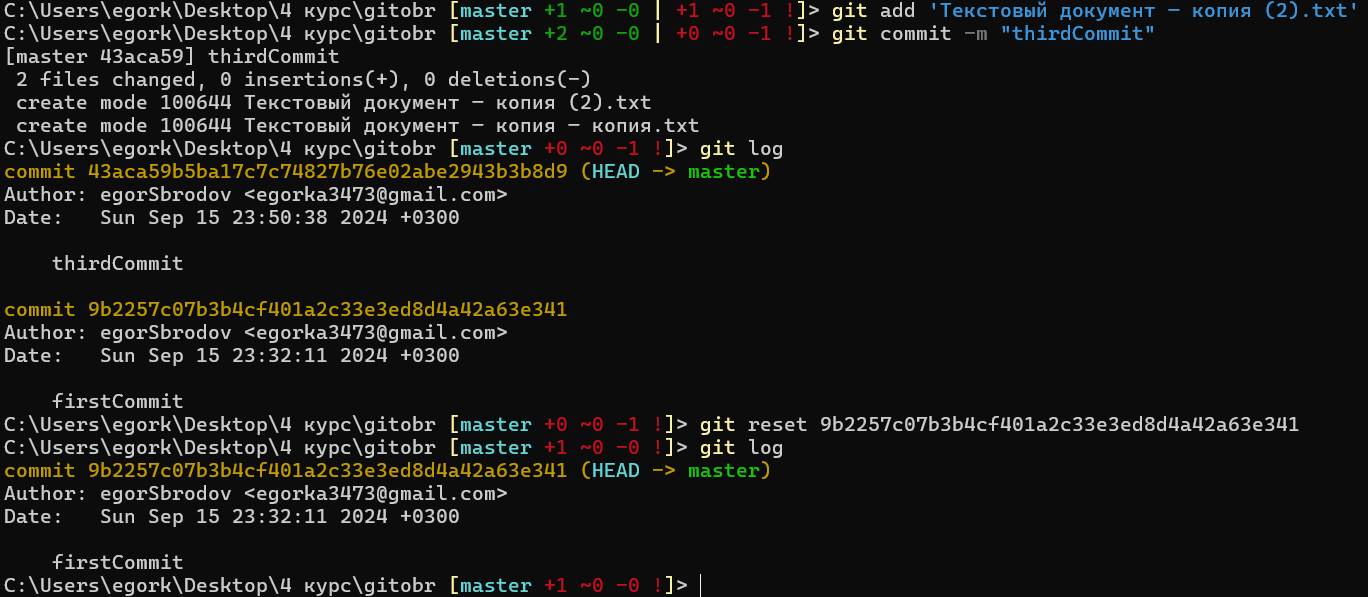


Рисунок 19 - создание коммита, просмотр и переход к другому коммиту с удалеением остальных

Вывод: настроил конфигурацию гит, научился создавать локальные репозитории, научился работать с файлами в данном репозиториям, а также научился просматривать и перемещаться по коммитам.

# Практическая работа №2