

Лабораторная работа 2

Первоначальная настройка git

Головина Мария Игоревна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	16
	Список литературы	17

Список иллюстраций

4.1	Создание папки	9
4.2	Переход в папку	10
4.3	Создание репозитория	10
4.4	Клонирование репозитория	11
4.5	Переход в каталог и удаление лишних файлов	11
4.6	Создание необходимых каталогов	12
4.7	Отправка файлов на сервер	12

Список таблиц

1 Цель работы

Изучить идеологию и применение средств контроля версий. Освоить умения по работе с git.

2 Задание

1. Установка программного обеспечения.
2. Базовая настройка git.
3. Создайте ключи ssh.
4. Создайте ключи pgr.
5. Настройка github.
6. Добавление PGP ключа в GitHub.
7. Настройка автоматических подписей коммитов git.
8. Настройка gh.
9. Шаблон для рабочего пространства.
10. Создание репозитория курса на основе шаблона.
11. Настройка каталога курса.

3 Теоретическое введение

Системы контроля версий. Общие понятия

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. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию изменённых файлов, а производить так называемую дельта-компрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными

участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом.

Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить.

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 о Linux см. в [1-7]

4 Выполнение лабораторной работы

1. Так как я уже ранее подключала GitHub и создавала необходимые ключи для работы с виртуальной машиной. Начинаю выполнение заданий не с начала. Создаю папку “Операционные системы”. (рис. 4.1).

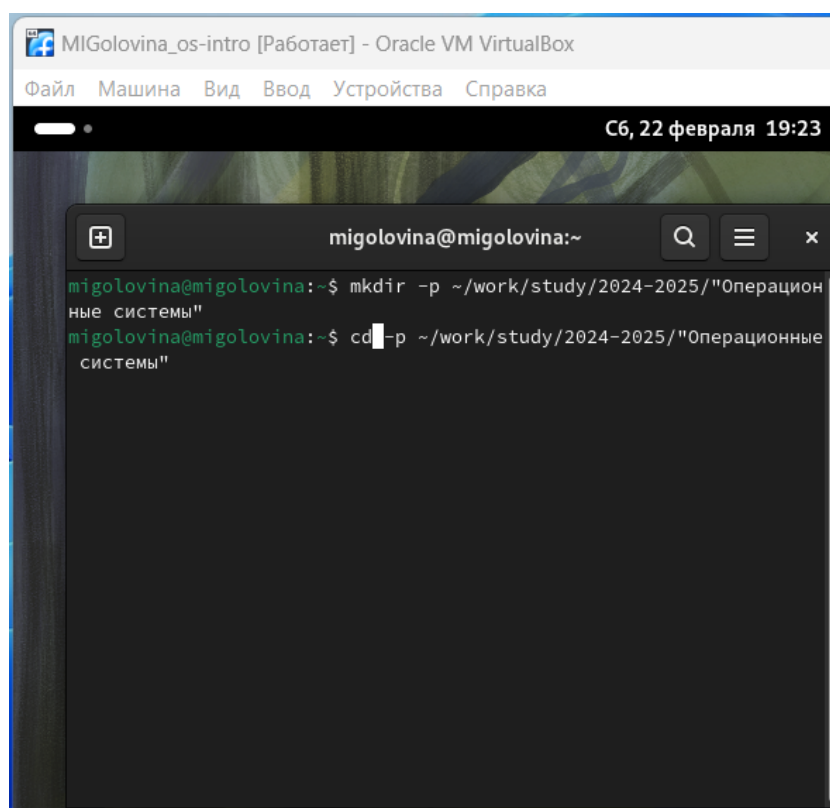


Рис. 4.1: Создание папки

2. Переход в папку “Операционные системы”. (рис. 4.2).

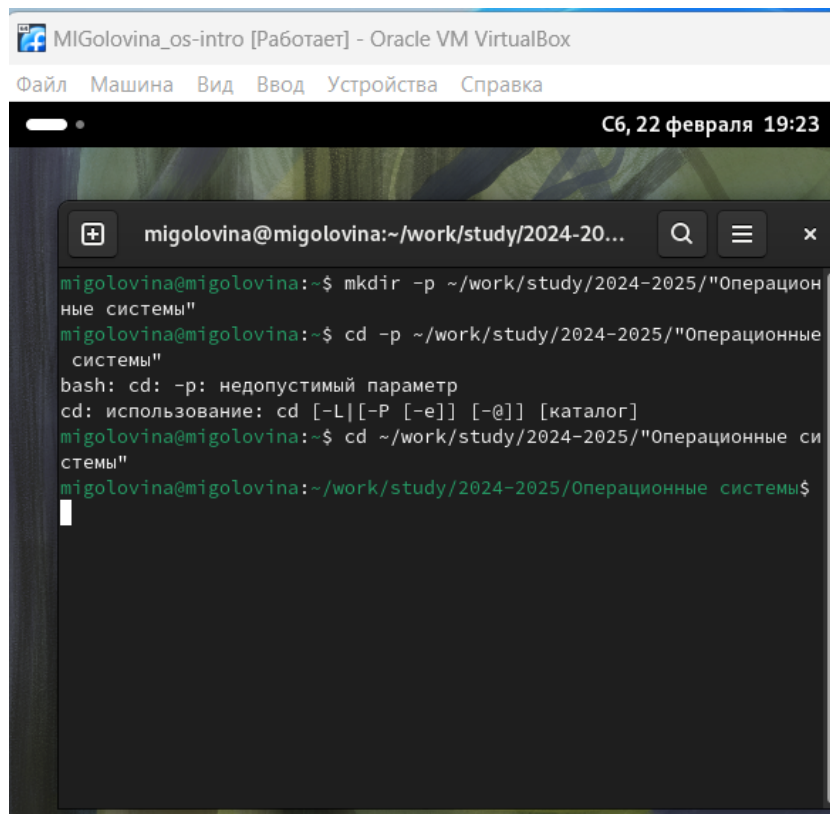


Рис. 4.2: Переход в папку

3. Создание и клонирование репозитория. (рис. 4.3-4.4).

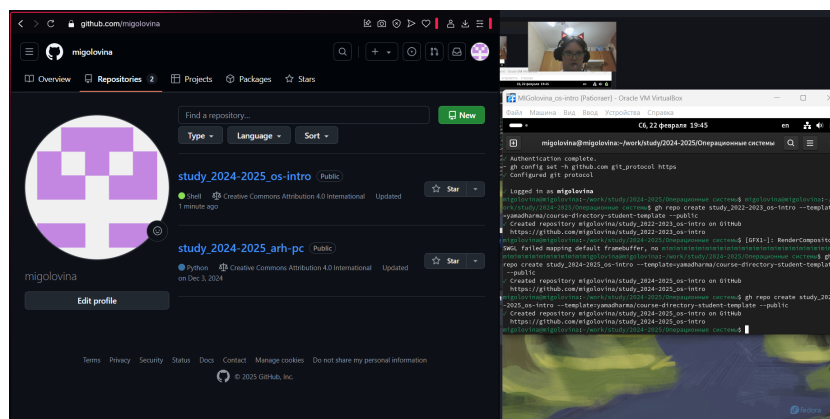


Рис. 4.3: Создание репозитория

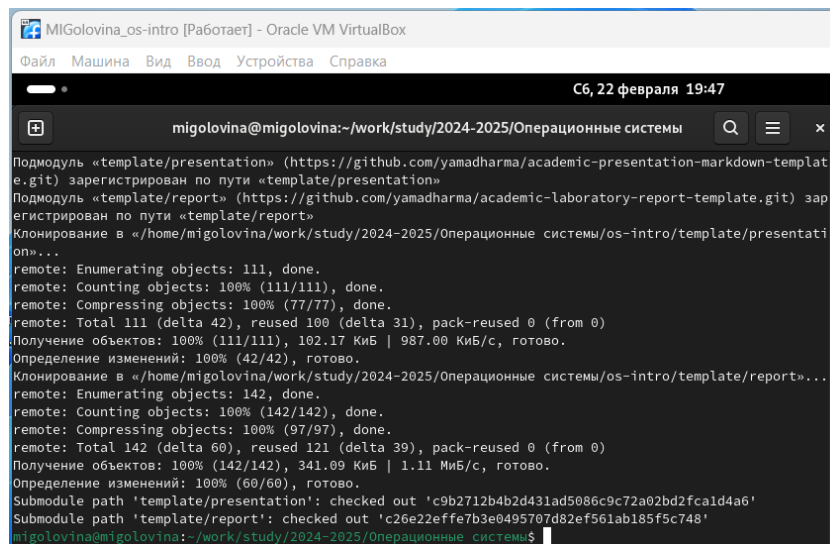


Рис. 4.4: Клонирование репозитория

4. Настройка каталога курса. Переход в каталог курса и удаление лишних файлов. (рис. 4.5).

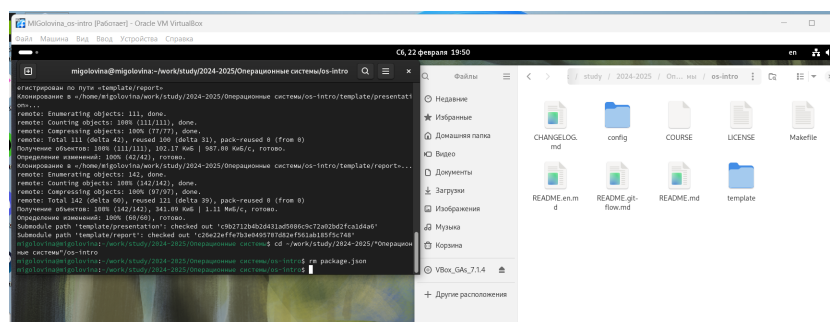
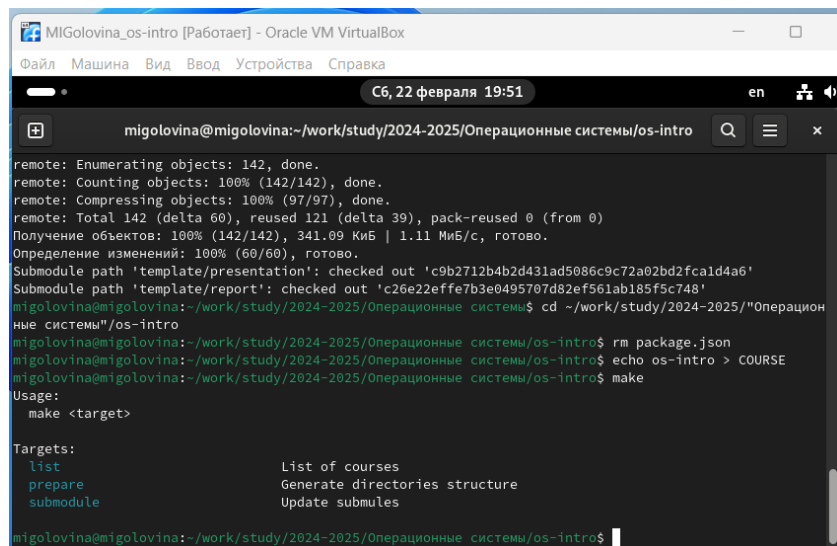


Рис. 4.5: Переход в каталог и удаление лишних файлов

5. Создание необходимых каталогов. (рис. 4.6)

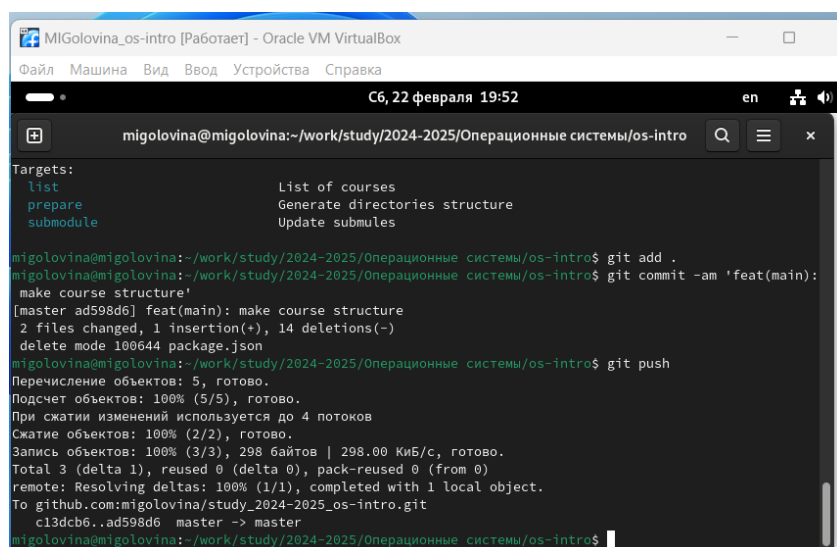


```
remote: Enumerating objects: 142, done.
remote: Counting objects: 100% (142/142), done.
remote: Compressing objects: 100% (97/97), done.
remote: Total 142 (delta 60), reused 121 (delta 39), pack-reused 0 (from 0)
Получение объектов: 100% (142/142), 341.09 КиБ | 1.11 МБ/с, готово.
Определение изменений: 100% (60/60), готово.
Submodule path 'template/presentation': checked out 'c0b2712b4b2d431ad5086c9c72a02bd2fca1d4a6'
Submodule path 'template/report': checked out 'c26e22effe7b3e0495707d82ef561ab185f5c748'
migolovina@migolovina:~/work/study/2024-2025/Операционные системы$ cd ~/work/study/2024-2025/"Операционные системы"/os-intro
migolovina@migolovina:~/work/study/2024-2025/Операционные системы/os-intro$ rm package.json
migolovina@migolovina:~/work/study/2024-2025/Операционные системы/os-intro$ echo os-intro > COURSE
migolovina@migolovina:~/work/study/2024-2025/Операционные системы/os-intro$ make
Usage:
  make <target>

Targets:
  list           List of courses
  prepare        Generate directories structure
  submodule      Update submodules
migolovina@migolovina:~/work/study/2024-2025/Операционные системы/os-intro$
```

Рис. 4.6: Создание необходимых каталогов

5. Отправка файлов на сервер. (рис. 4.7)



```
Targets:
  list           List of courses
  prepare        Generate directories structure
  submodule      Update submodules
migolovina@migolovina:~/work/study/2024-2025/Операционные системы/os-intro$ git add .
migolovina@migolovina:~/work/study/2024-2025/Операционные системы/os-intro$ git commit -am 'feat(main): make course structure'
[main: ad598d6] feat(main): make course structure
2 files changed, 1 insertion(+), 14 deletions(-)
delete mode 100644 package.json
migolovina@migolovina:~/work/study/2024-2025/Операционные системы/os-intro$ git push
Перечисление объектов: 5, готово.
Подсчет объектов: 100% (5/5), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (2/2), готово.
Запись объектов: 100% (3/3), 298 байтов | 298.00 КиБ/с, готово.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:migolovina/study_2024-2025_os-intro.git
 c13dcb6..ad598d6 master -> master
migolovina@migolovina:~/work/study/2024-2025/Операционные системы/os-intro$
```

Рис. 4.7: Отправка файлов на сервер

Контрольные вопросы 1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначены? Система контроля версий (VCS) - программное обеспечение для облегчения работы с изменяющейся информацией.

Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое. Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище – репозиторий - место хранения всех версий и служебной информации. Commit - это команда для записи индексированных изменений в репозиторий. История – место, где сохраняются все коммиты, по которым можно посмотреть данные о коммитах. Рабочая копия – текущее состояние файлов проекта, основанное на версии, загруженной из хранилища.

3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. Централизованные системы – это системы, в которых одно основное хранилище всего проекта, и каждый пользователь копирует необходимые ему файлы, изменяет и вставляет обратно. Пример – Subversion. Децентрализованные системы – система, в которой каждый пользователь имеет свой вариант репозитория и есть возможность добавлять и забирать изменения из репозитория. Пример – Git.
4. Опишите действия с VCS при единоличной работе с хранилищем. В рабочей копии, которую исправляет человек, появляются правки, которые отправляются в хранилище на каждом из этапов. То есть в правки в рабочей копии появляются, только если человек делает их (отправляет их на сервер) и никак по-другому.
5. Опишите порядок работы с общим хранилищем VCS. Если хранилище общее, то в рабочую копию каждого, кто работает над проектом, приходят

изменения, отправленные на сервер одним из команды. Рабочая правка каждого может изменяться вне зависимости оттого, делает ли конкретный человек правки или нет.

6. Каковы основные задачи, решаемые инструментальным средством git? У Git две основных задачи: первая — хранить информацию обо всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.
7. Назовите и дайте краткую характеристику командам git. создание основного дерева репозитория: `git init` получение обновлений (изменений) текущего дерева из центрального репозитория: `git pull` отправка всех произведённых изменений локального дерева в центральный репозиторий: `git push` просмотр списка изменённых файлов в текущей директории: `git status` просмотр текущих изменений: `git diff` сохранение текущих изменений: —добавить все изменённые и / или созданные файлы и/или каталоги: `git add .` добавить конкретные изменённые и/или созданные файлы и/или каталоги: `git add` удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): `git rm` имена_файлов сохранить все добавленные изменения и все изменённые файлы: `git commit -am` 'Описание коммита' сохранить добавленные изменения с внесением комментария через встроенный редактор: `git commit` создание новой ветки, базирующейся на текущей: `git checkout -b` имя_ветки переключение на некоторую ветку: `git checkout` имя_ветки (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) отправка изменений конкретной ветки в центральный репозиторий: `1 git push origin` имя_ветки слияние ветки с текущим деревом: `1 git merge —no-ff` имя_ветки удаление локальной уже слитой с основным деревом ветки: `git branch -d` имя_ветки принудительное удаление локальной ветки: `git branch -D` имя_ветки удаление ветки с центрального репозитория: `git push origin`

:имя_ветки

8. Приведите примеры использования при работе с локальным и удалённым репозиториями. Работа с удаленным репозиторием: `git remote`—просмотр списка настроенных удаленных репозиториях. Работа с локальным репозиторием: `git status`-выводит информацию обо всех изменениях, внесенных в дерево директорий проекта по сравнению с последним коммитом рабочей ветки
9. Что такое и зачем могут быть нужны ветви (branches)? Ветка(англ.branch)—это последовательность коммитов, в которой ведётся параллельная разработка какого-либо функционала. Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже файлом одновременно, при этом, не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций: чтобы не повредить основному проекту, создается новая ветка специально для экспериментов.
10. Как и зачем можно игнорировать некоторые файлы при commit? Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты. В Git нет специальной команды для указания игнорируемых файлов: вместо этого необходимо вручную отредактировать файл. Временно игнорировать изменения в файле можно командой `git update-index - assumeunchanged`

5 Выводы

Я изучила идеологию и применение средств контроля версий, а также освоила умения по работе с git.

Список литературы

1. Dash, P. Getting Started with Oracle VM VirtualBox / P. Dash. – Packt Publishing Ltd, 2013. – 86 сс.
2. Colvin, H. VirtualBox: An Ultimate Guide Book on Virtualization with VirtualBox. VirtualBox / H. Colvin. – CreateSpace Independent Publishing Platform, 2015. – 70 сс.
3. Vugt, S. van. Red Hat RHCSA/RHCE 7 cert guide : Red Hat Enterprise Linux 7 (EX200 and EX300) : Certification Guide. Red Hat RHCSA/RHCE 7 cert guide / S. van Vugt. – Pearson IT Certification, 2016. – 1008 сс.
4. Робачевский, А. Операционная система UNIX / А. Робачевский, С. Немнюгин, О. Стесик. – 2-е изд. – Санкт-Петербург : БХВ-Петербург, 2010. – 656 сс.
5. Немет, Э. Unix и Linux: руководство системного администратора. Unix и Linux / Э. Немет, Г. Снайдер, Т.Р. Хейн, Б. Уэйли. – 4-е изд. – Вильямс, 2014. – 1312 сс.
6. Колисниченко, Д.Н. Самоучитель системного администратора Linux : Системный администратор / Д.Н. Колисниченко. – Санкт-Петербург : БХВ-Петербург, 2011. – 544 сс.
7. Robbins, A. Bash Pocket Reference / A. Robbins. – O'Reilly Media, 2016. – 156 сс.