Лабораторная работа 4

Продвинутое использование git

Головина Мария Игоревна

Содержание

# 1 Цель работы

Получение навыков правильной работы с репозиториями git

# 2 Задание

1. Выполнить работу для тестового репозитория.
2. Преобразовать рабочий репозиторий в репозиторий с git-flow и conventional commits.

# 3 Теоретическое введение

Рабочий процесс Gitflow

Рабочий процесс Gitflow Workflow. Будем описывать его с использованием пакета git-flow.

Общая информация

Gitflow Workflow опубликована и популяризована Винсентом Дриссеном. Gitflow Workflow предполагает выстраивание строгой модели ветвления с учётом выпуска проекта. Данная модель отлично подходит для организации рабочего процесса на основе релизов. Работа по модели Gitflow включает создание отдельной ветки для исправлений ошибок в рабочей среде. Последовательность действий при работе по модели Gitflow: Из ветки master создаётся ветка develop. Из ветки develop создаётся ветка release. Из ветки develop создаются ветки feature. Когда работа над веткой feature завершена, она сливается с веткой develop. Когда работа над веткой релиза release завершена, она сливается в ветки develop и master. Если в master обнаружена проблема, из master создаётся ветка hotfix. Когда работа над веткой исправления hotfix завершена, она сливается в ветки develop и master.

Процесс работы с Gitflow

Основные ветки (master) и ветки разработки (develop) Для фиксации истории проекта в рамках этого процесса вместо одной ветки master используются две ветки. В ветке master хранится официальная история релиза, а ветка develop предназначена для объединения всех функций. Кроме того, для удобства рекомендуется присваивать всем коммитам в ветке master номер версии.

При использовании библиотеки расширений git-flow нужно инициализировать структуру в существующем репозитории:

git flow init

Для github параметр Version tag prefix следует установить в v.

После этого проверьте, на какой ветке Вы находитесь:

git branch

Функциональные ветки (feature) Под каждую новую функцию должна быть отведена собственная ветка, которую можно отправлять в центральный репозиторий для создания резервной копии или совместной работы команды. Ветки feature создаются не на основе master, а на основе develop. Когда работа над функцией завершается, соответствующая ветка сливается обратно с веткой develop. Функции не следует отправлять напрямую в ветку master. Как правило, ветки feature создаются на основе последней ветки develop.

Создание функциональной ветки

Создадим новую функциональную ветку:

git flow feature start feature\_branch

Далее работаем как обычно.

Окончание работы с функциональной веткой

По завершении работы над функцией следует объединить ветку feature\_branch с develop:

git flow feature finish feature\_branch

Ветки выпуска (release) Когда в ветке develop оказывается достаточно функций для выпуска, из ветки develop создаётся ветка release. Создание этой ветки запускает следующий цикл выпуска, и с этого момента новые функции добавить больше нельзя — допускается лишь отладка, создание документации и решение других задач. Когда подготовка релиза завершается, ветка release сливается с master и ей присваивается номер версии. После нужно выполнить слияние с веткой develop, в которой с момента создания ветки релиза могли возникнуть изменения. Благодаря тому, что для подготовки выпусков используется специальная ветка, одна команда может дорабатывать текущий выпуск, в то время как другая команда продолжает работу над функциями для следующего.

Создать новую ветку release можно с помощью следующей команды:

git flow release start 1.0.0

Для завершения работы на ветке release используются следующие команды:

git flow release finish 1.0.0

Ветки исправления (hotfix) Ветки поддержки или ветки hotfix используются для быстрого внесения исправлений в рабочие релизы. Они создаются от ветки master. Это единственная ветка, которая должна быть создана непосредственно от master. Как только исправление завершено, ветку следует объединить с master и develop. Ветка master должна быть помечена обновлённым номером версии. Наличие специальной ветки для исправления ошибок позволяет команде решать проблемы, не прерывая остальную часть рабочего процесса и не ожидая следующего цикла релиза.

Ветку hotfix можно создать с помощью следующих команд:

git flow hotfix start hotfix\_branch

По завершении работы ветка hotfix объединяется с master и develop:

git flow hotfix finish hotfix\_branch

Семантическое версионирование

Семантический подход в версионированию программного обеспечения.

Краткое описание семантического версионирования

Семантическое версионирование описывается в манифесте семантического версионирования.

Кратко его можно описать следующим образом: Версия задаётся в виде кортежа МАЖОРНАЯ\_ВЕРСИЯ.МИНОРНАЯ\_ВЕРСИЯ.ПАТЧ. Номер версии следует увеличивать: МАЖОРНУЮ версию, когда сделаны обратно несовместимые изменения API. МИНОРНУЮ версию, когда вы добавляете новую функциональность, не нарушая обратной совместимости. ПАТЧ-версию, когда вы делаете обратно совместимые исправления. Дополнительные обозначения для предрелизных и билд-метаданных возможны как дополнения к МАЖОРНАЯ.МИНОРНАЯ.ПАТЧ формату.

Программное обеспечение

Для реализации семантического версионирования создано несколько программных продуктов. При этом лучше всего использовать комплексные продукты, которые используют информацию из коммитов системы версионирования. Коммиты должны иметь стандартизованный вид. В семантическое версионирование применяется вместе с общепринятыми коммитами.

Пакет Conventional Changelog Пакет Conventional Changelog является комплексным решением по управлению коммитами и генерации журнала изменений. Содержит набор утилит, которые можно использовать по-отдельности.

Общепринятые коммиты

Использование спецификации Conventional Commits.

Описание

Спецификация Conventional Commits:

Соглашение о том, как нужно писать сообщения commit’ов. Совместимо с SemVer. Даже вернее сказать, сильно связано с семантическим версионированием. Регламентирует структуру и основные типы коммитов.

Структура коммита

():

Или, по-русски:

(): [необязательное тело] [необязательный нижний колонтитул]

Заголовок является обязательным. Любая строка сообщения о фиксации не может быть длиннее 100 символов. Тема (subject) содержит краткое описание изменения. Используйте повелительное наклонение в настоящем времени: «изменить» (“change” not “changed” nor “changes”). Не используйте заглавную первую букву. Не ставьте точку в конце. Тело (body) должно включать мотивацию к изменению и противопоставлять это предыдущему поведению. Как и в теме, используйте повелительное наклонение в настоящем времени. Нижний колонтитул (footer) должен содержать любую информацию о критических изменениях. Следует использовать для указания внешних ссылок, контекста коммита или другой мета информации. Также содержит ссылку на issue (например, на github), который закрывает эта фиксация. Критические изменения должны начинаться со слова BREAKING CHANGE: с пробела или двух символов новой строки. Затем для этого используется остальная часть сообщения фиксации.

Типы коммитов

Базовые типы коммитов fix: — коммит типа fix исправляет ошибку (bug) в вашем коде (он соответствует PATCH в SemVer). feat: — коммит типа feat добавляет новую функцию (feature) в ваш код (он соответствует MINOR в SemVer). BREAKING CHANGE: — коммит, который содержит текст BREAKING CHANGE: в начале своего не обязательного тела сообщения (body) или в подвале (footer), добавляет изменения, нарушающие обратную совместимость вашего API (он соответствует MAJOR в SemVer). BREAKING CHANGE может быть частью коммита любого типа. revert: — если фиксация отменяет предыдущую фиксацию. Начинается с revert:, за которым следует заголовок отменённой фиксации. В теле должно быть написано: Это отменяет фиксацию (это SHA-хэш отменяемой фиксации). Другое: коммиты с типами, которые отличаются от fix: и feat:, также разрешены. Например, [**commitlint/config-conventional?**] (основанный на The Angular convention) рекомендует: chore:, docs:, style:, refactor:, perf:, test:, и другие.

Соглашения The Angular convention Одно из популярных соглашений о поддержке исходных кодов — конвенция Angular (The Angular convention).

Типы коммитов The Angular convention

Конвенция Angular (The Angular convention) требует следующие типы коммитов: build: — изменения, влияющие на систему сборки или внешние зависимости (примеры областей (scope): gulp, broccoli, npm). ci: — изменения в файлах конфигурации и скриптах CI (примеры областей: Travis, Circle, BrowserStack, SauceLabs). docs: — изменения только в документации. feat: — новая функция. fix: — исправление ошибок. perf: — изменение кода, улучшающее производительность. refactor: — Изменение кода, которое не исправляет ошибку и не добавляет функции (рефакторинг кода). style: — изменения, не влияющие на смысл кода (пробелы, форматирование, отсутствие точек с запятой и т. д.). test: — добавление недостающих тестов или исправление существующих тестов.

Области действия (scope)

Областью действия должно быть имя затронутого пакета npm (как его воспринимает человек, читающий журнал изменений, созданный из сообщений фиксации).

Есть несколько исключений из правила «использовать имя пакета»: packaging — используется для изменений, которые изменяют структуру пакета, например, изменения общедоступного пути. changelog — используется для обновления примечаний к выпуску в CHANGELOG.md. отсутствует область действия — полезно для изменений стиля, тестирования и рефакторинга, которые выполняются во всех пакетах (например, style: добавить отсутствующие точки с запятой).

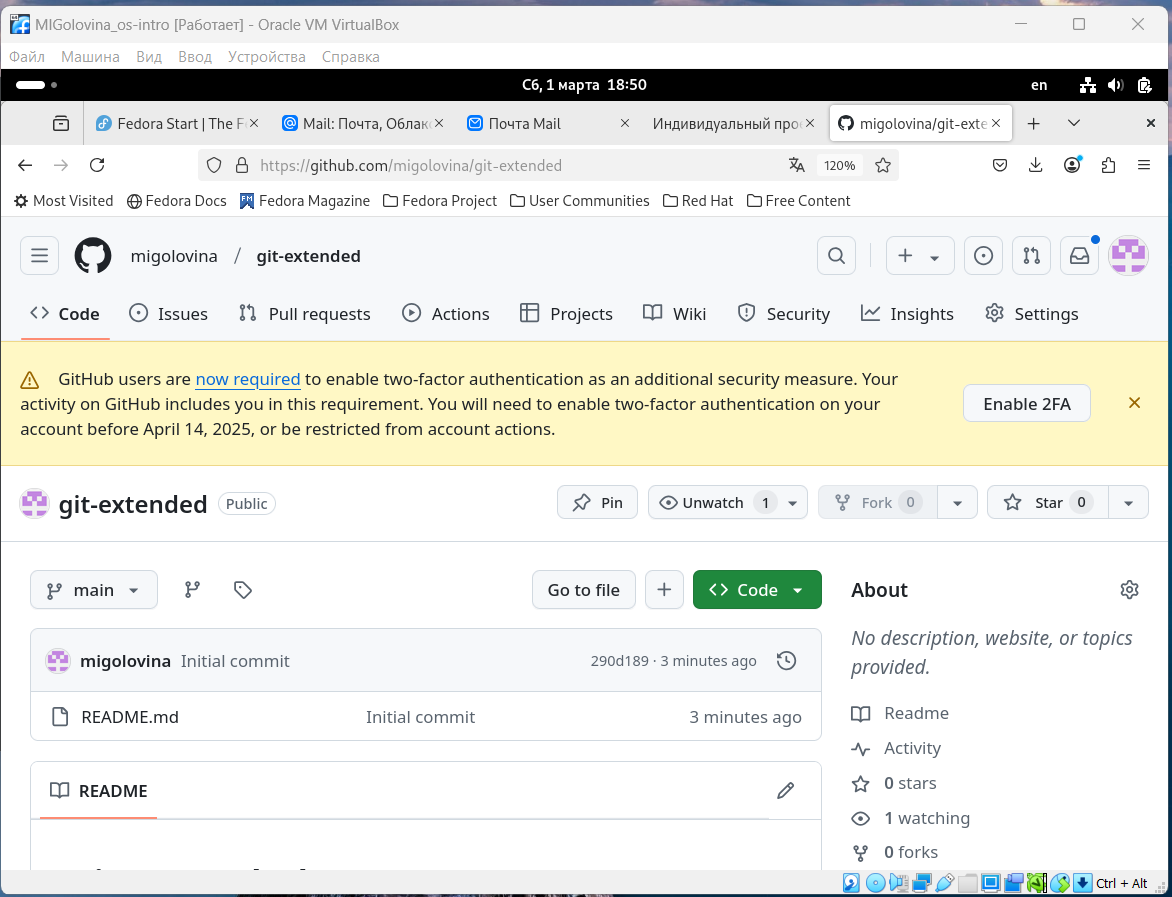
Соглашения [**commitlint/config-conventional?**]

Соглашение [**commitlint/config-conventional?**] входит в пакет Conventional Changelog. В целом в этом соглашении придерживаются соглашения Angular.

Более подробно о Linux см. в [1-7]

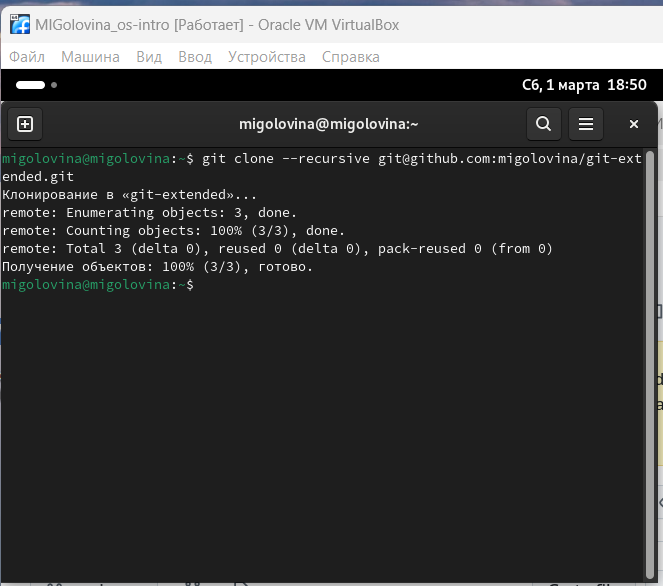
# 4 Выполнение лабораторной работы

1. Создание репозитория git-extended (рис. 4.1).



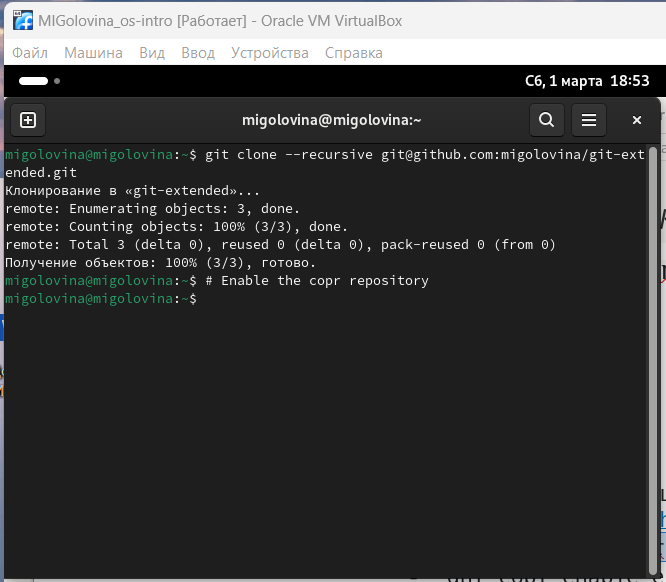
Репозиторий git-extended

1. Клонирование репозитория git-extended (рис. 4.2).

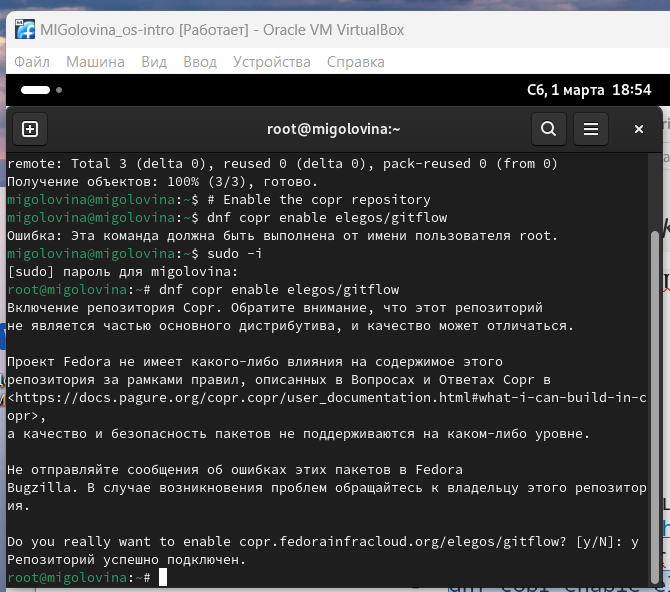


Клонирование

1. Установка из коллекции репозиториев Copr. Введение команд: # Enable the copr repository, dnf copr enable elegos/gitflow (рис. 4.3-4.4).

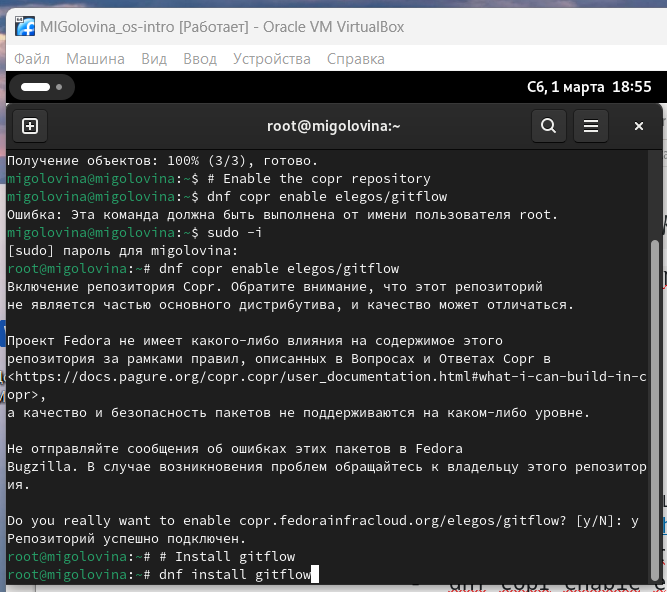


# Enable the copr repository

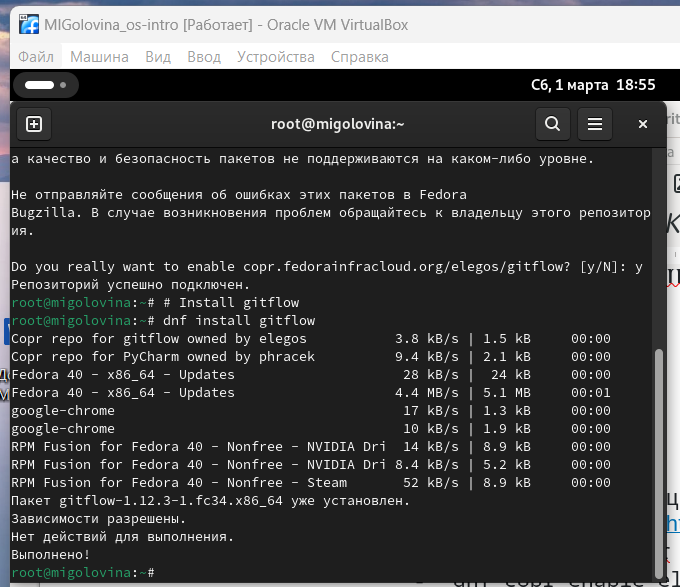


dnf copr enable elegos/gitflow

1. Установка из коллекции репозиториев Copr. Введение команд: # Install gitflow, dnf install gitflow (рис. 4.5-4.6).

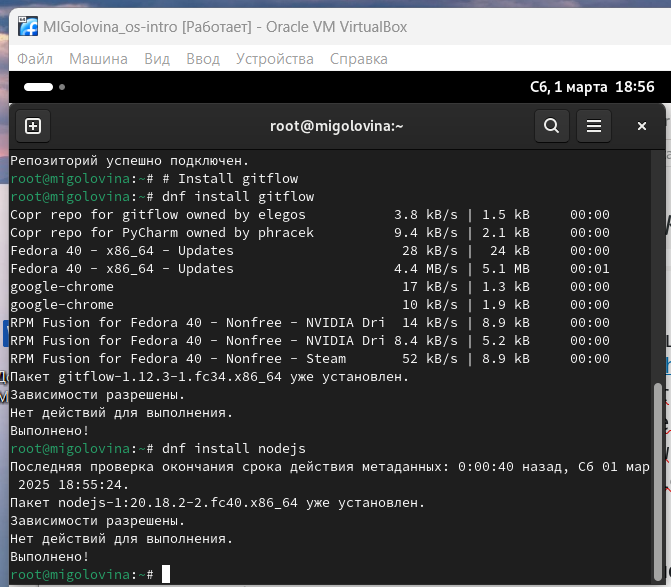


# Install gitflow

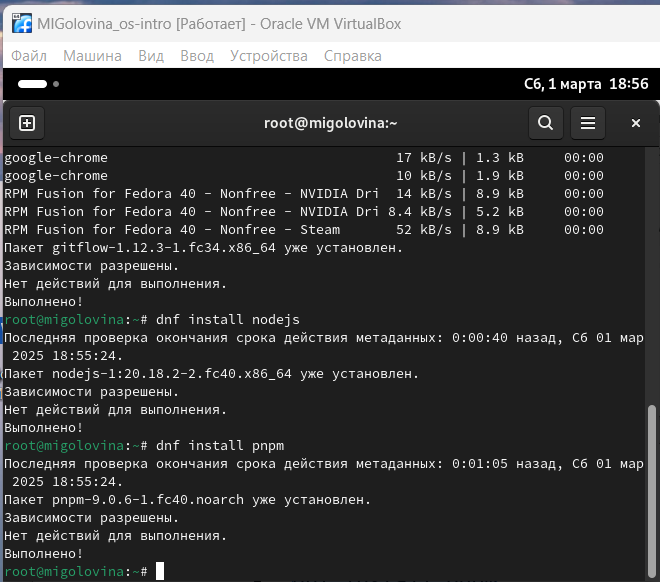


dnf install gitflow

1. Установка Node.js (рис. 4.7-4.8).

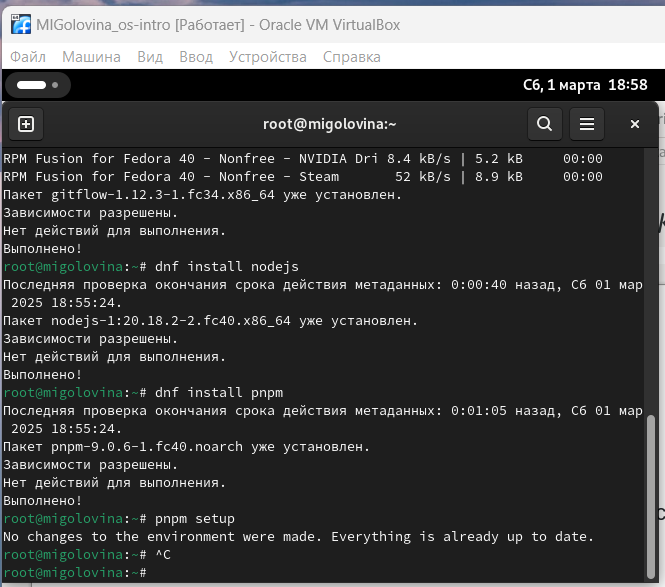


dnf install nodejs

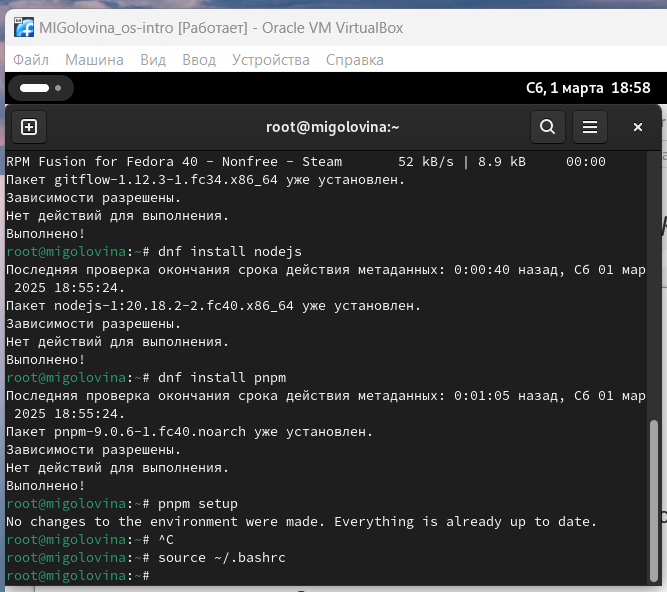


dnf install pnpm

1. Настройка Node.js (рис. 4.9-4.10).

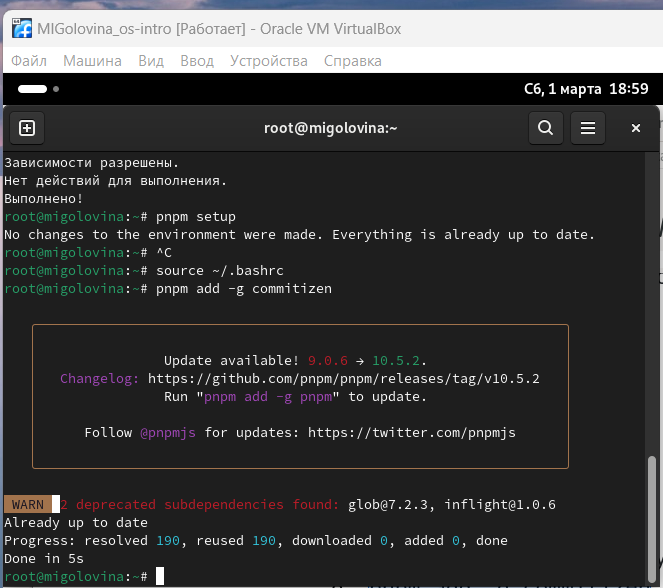


pnpm setup

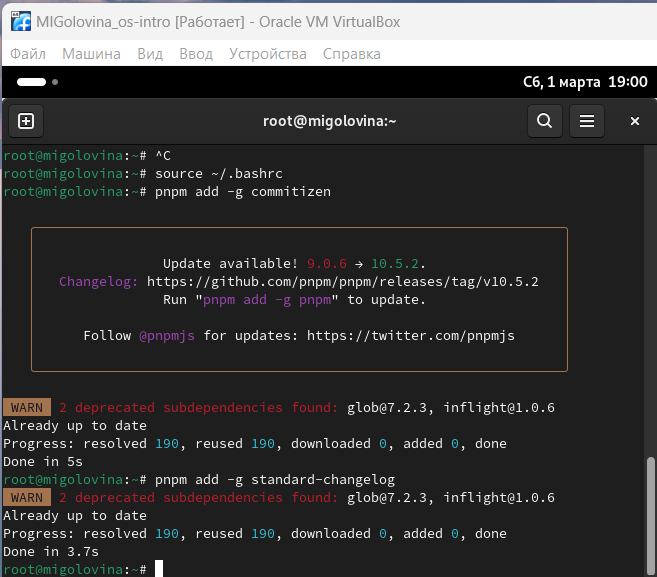


source ~/.bashrc

1. Общепринятые коммиты (рис. 4.11-4.12).

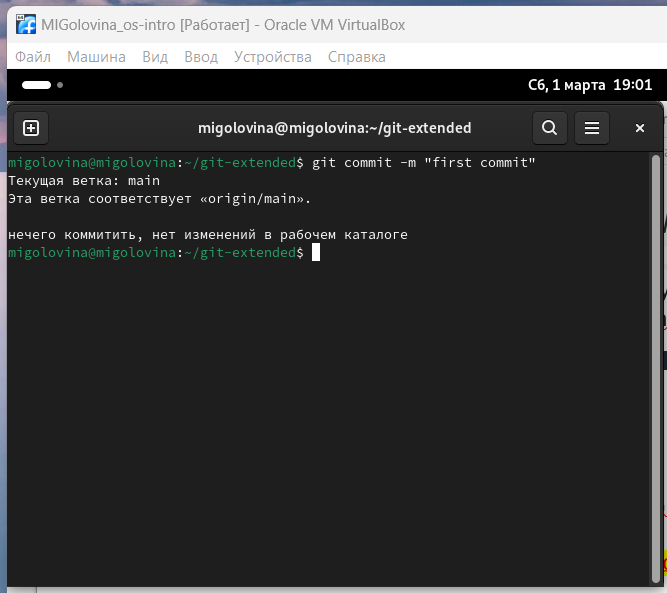


pnpm add -g commitizen

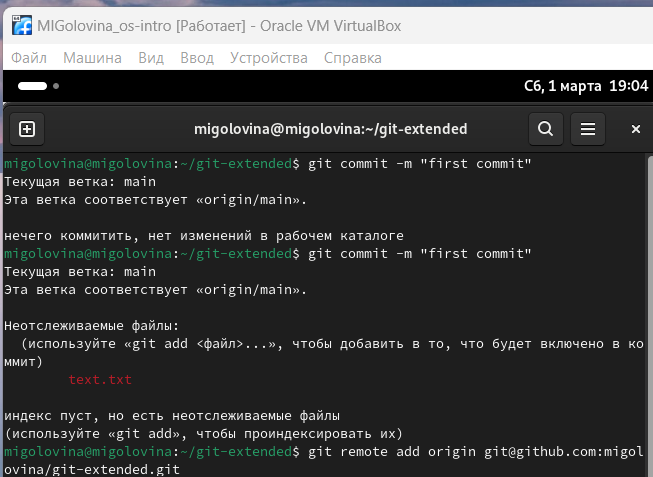


pnpm add -g standard-changelog

1. Делаем первый коммит (рис. 4.13-4.14).

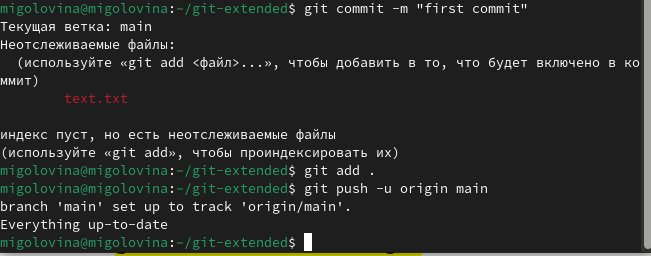


git commit -m “first commit”



git remote add origin git@github.com:migolovina/git-extended.git

1. Выкладываем на github (рис. 4.15).

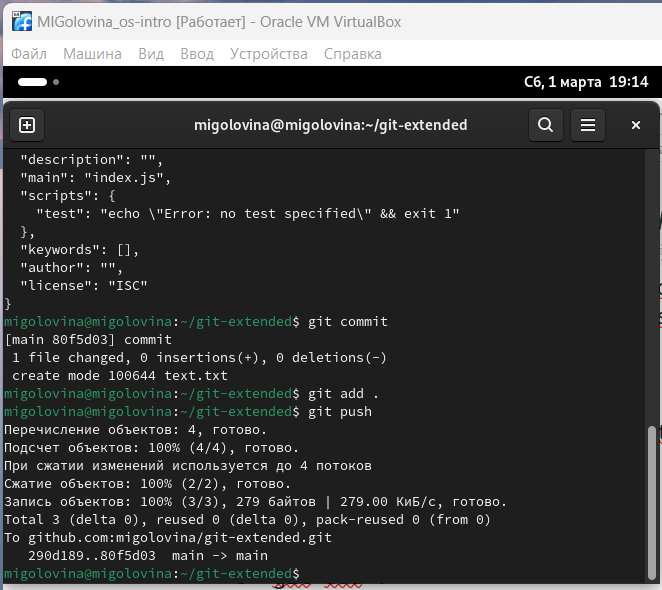


git push -u origin main

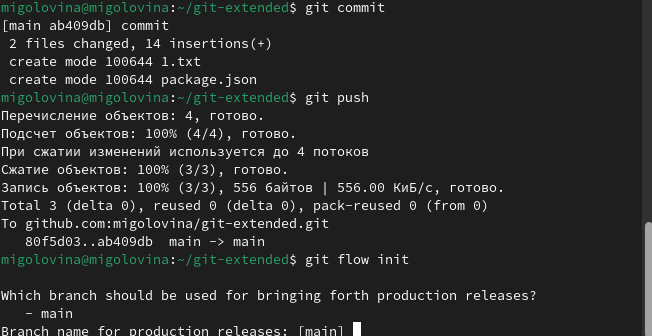
1. Конфигурация общепринятых коммитов. (рис. 4.16-4.17).



package.json

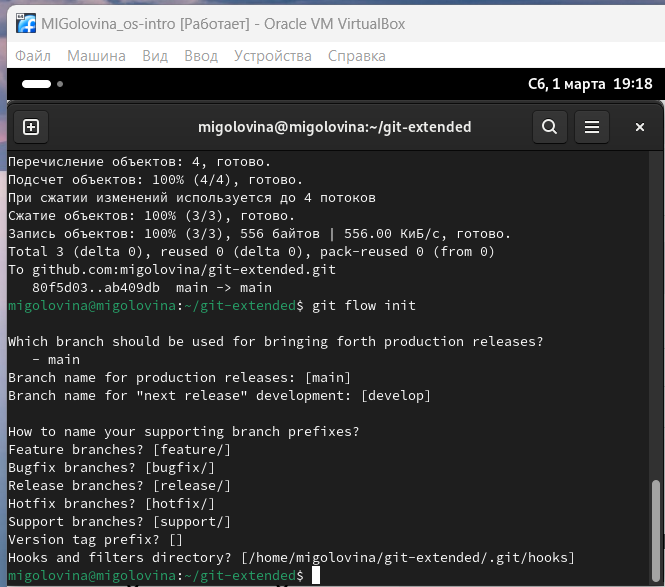


Добавляем новые файлы, выполняем коммит и отправляем на github

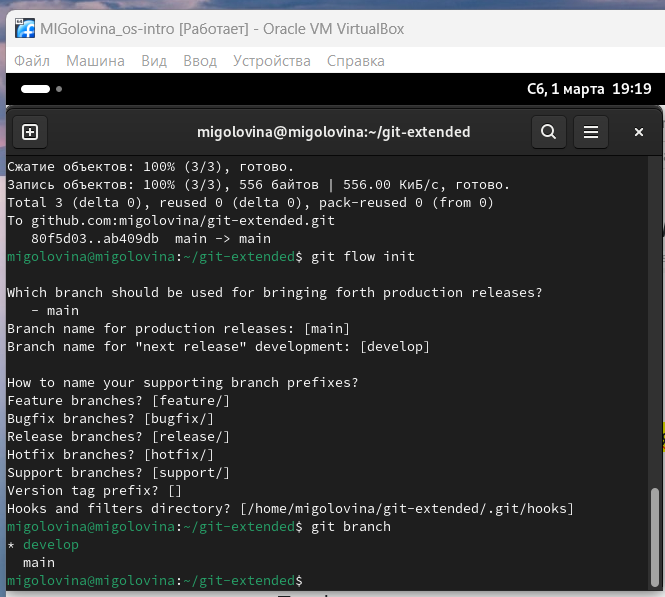


Отправляем на github

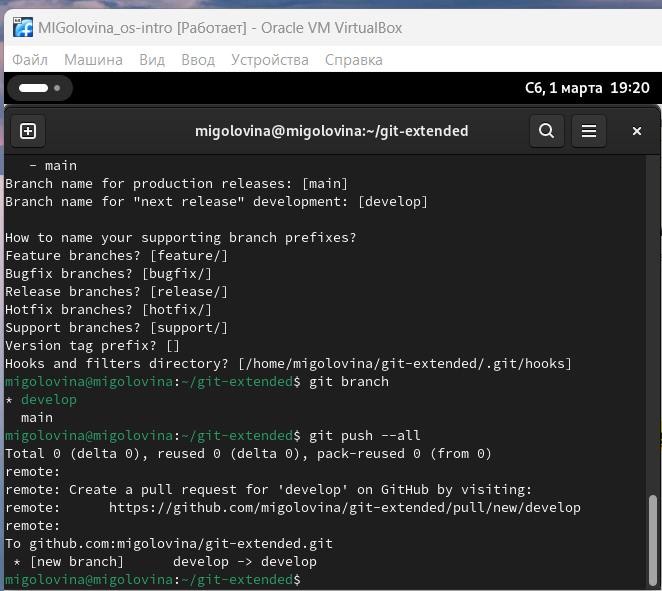
1. Конфигурация git-flow. (рис. 4.19-4.31).



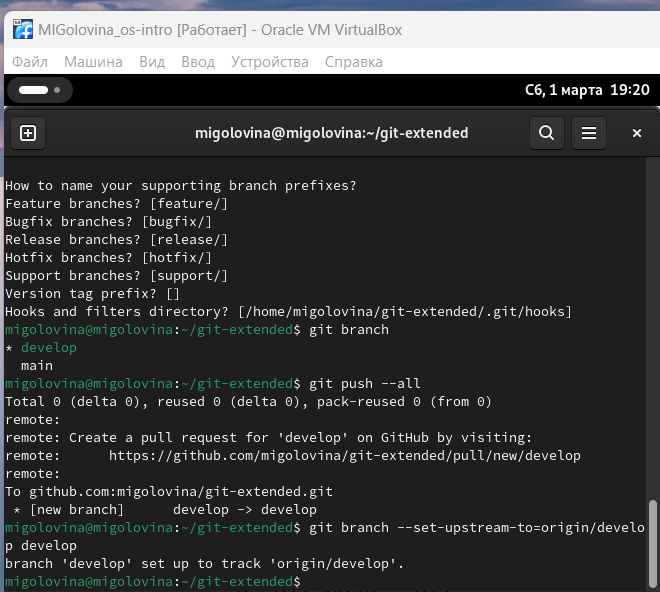
Инициализируем git-flow



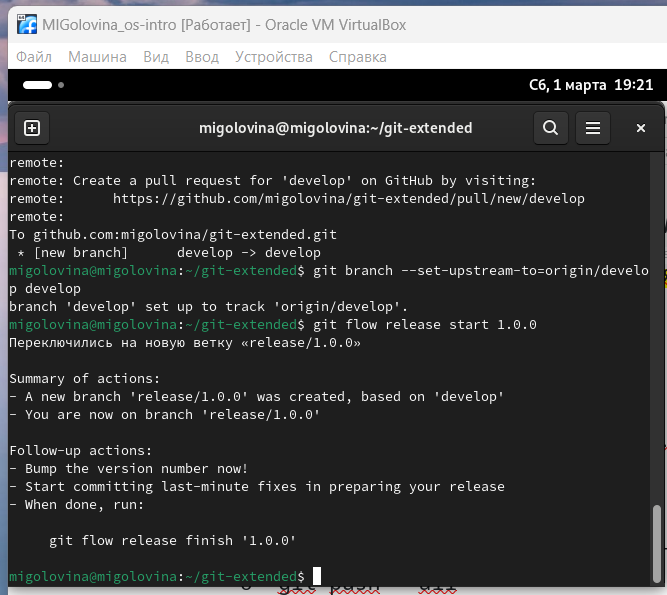
Проверка



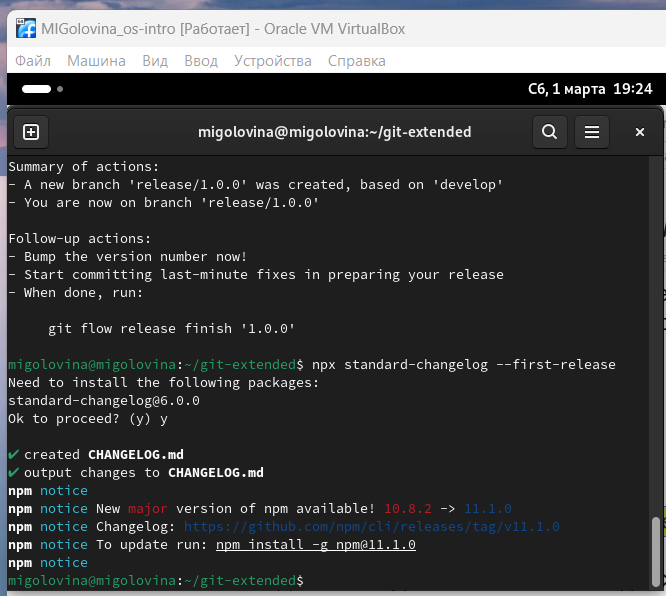
Загрузка всего репозитория в хранилище



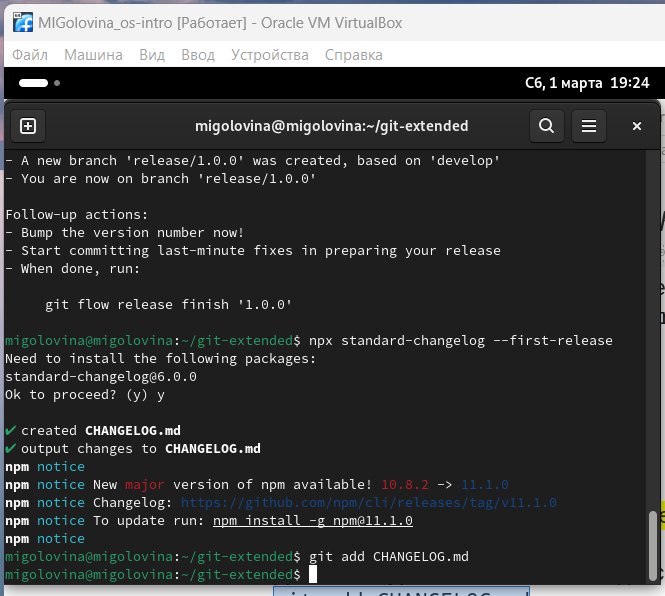
Установка внешней ветки как вышестоящую для этой ветки



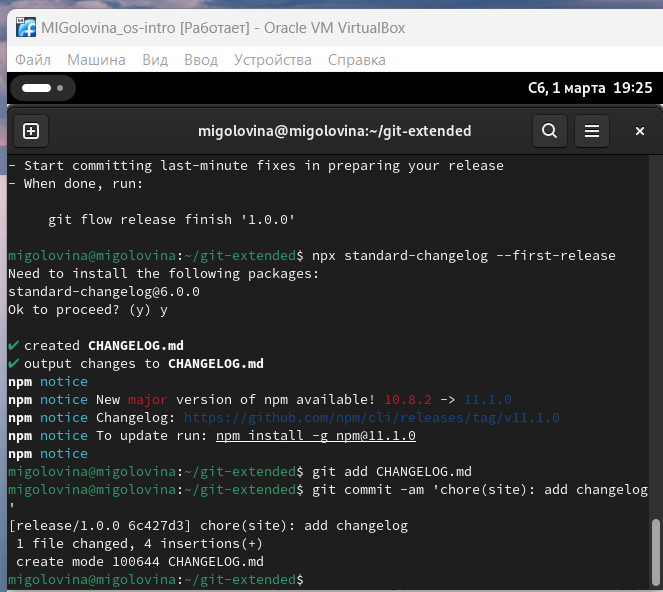
Создадние релиза с версией 1.0.0



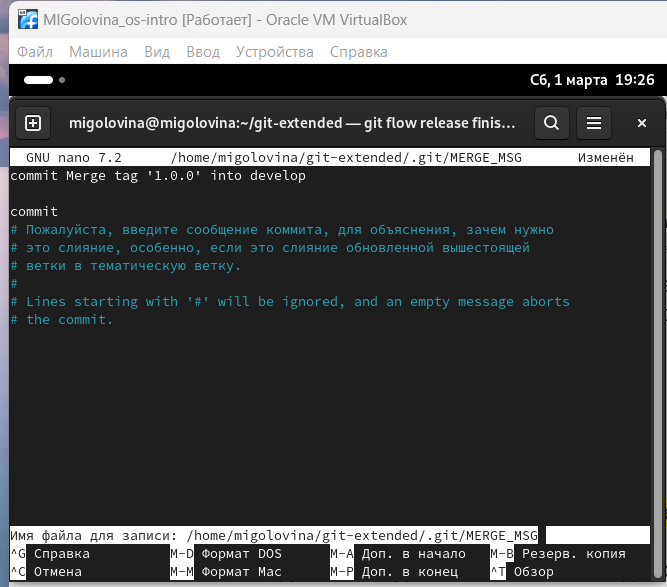
Создадние журнала изменений



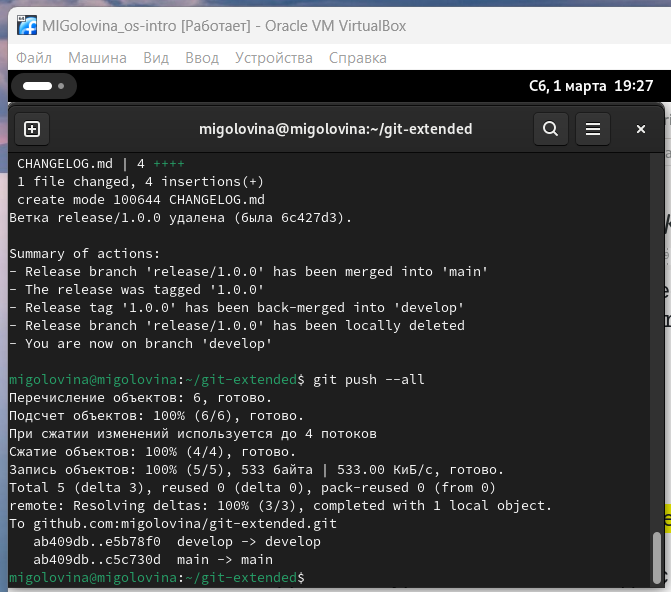
Добавление журнала изменений в индекс



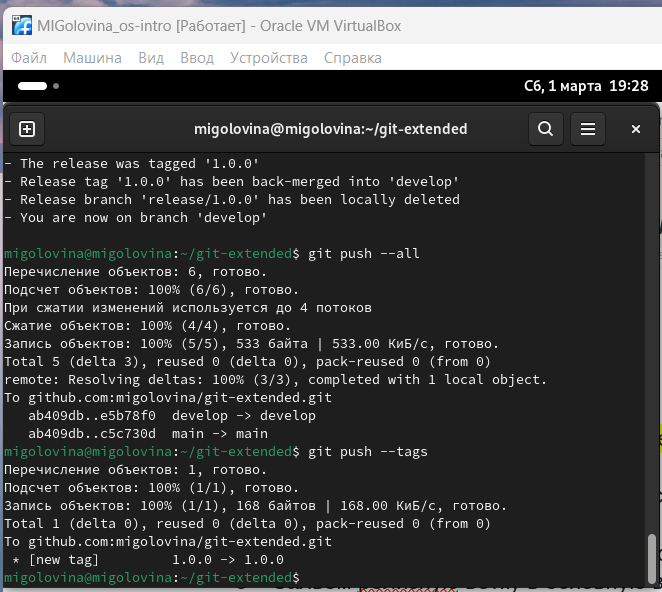
Добавление журнала изменений в индекс



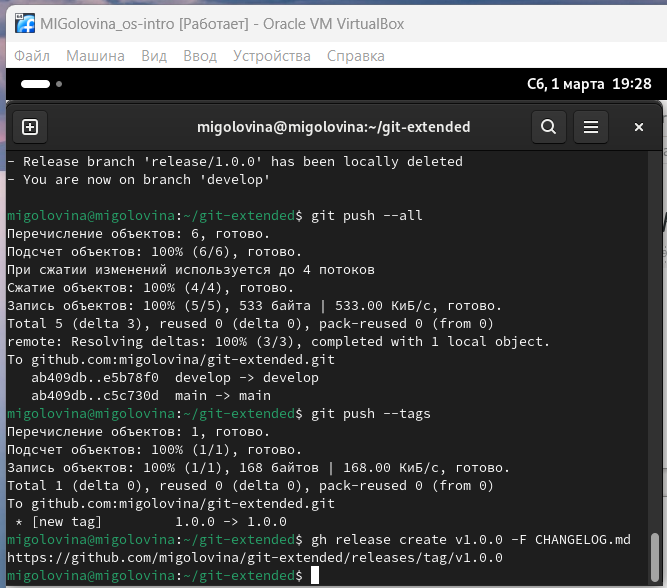
Залитие релизной ветки в основную ветку



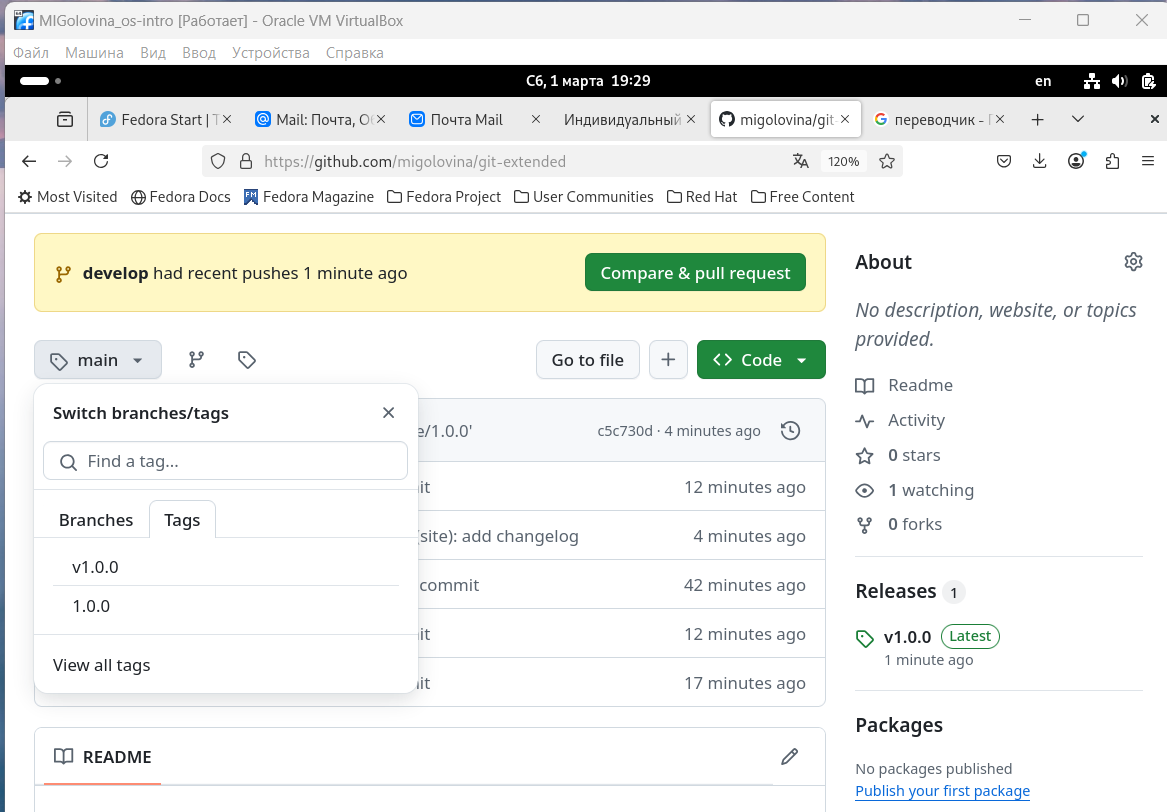
Отправление данных на github



Отправление данных на github

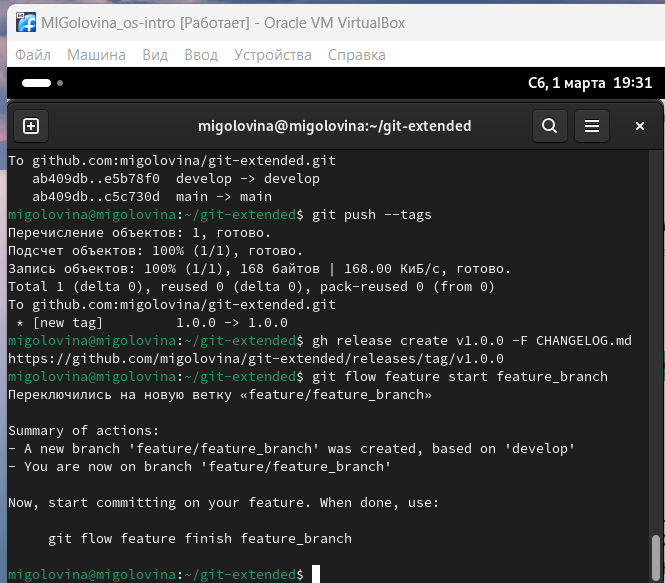


Создание релиза на github

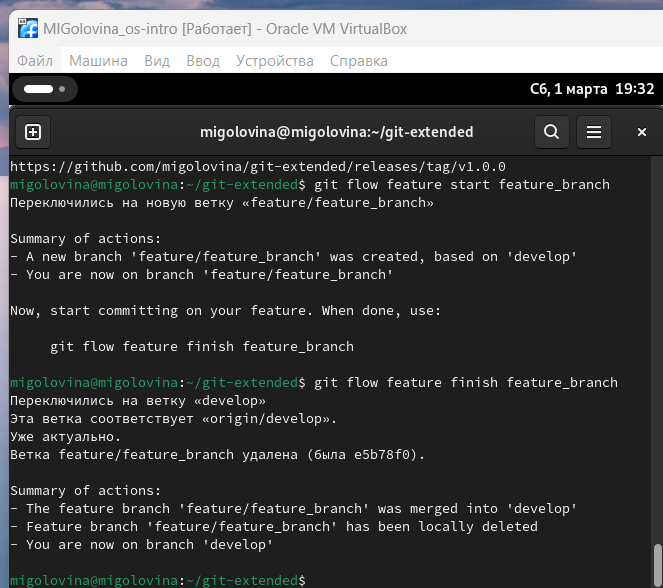


Проверка

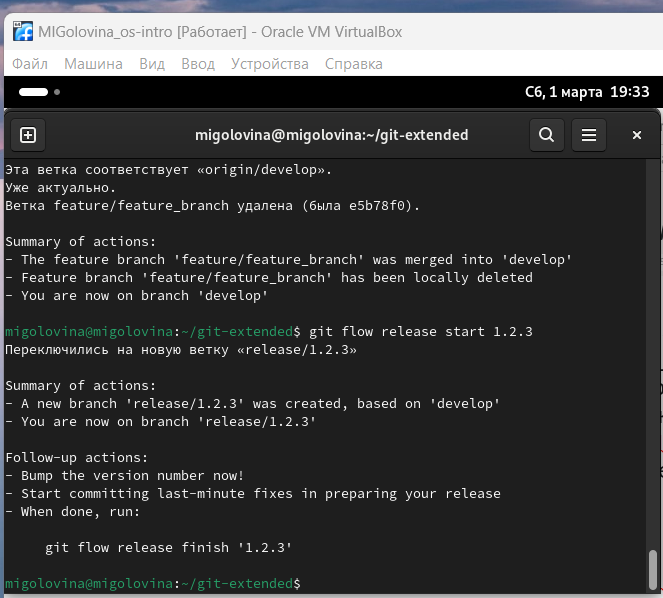
1. Работа с репозиторием git. (рис. 4.32-4.40).



Создание ветки для новой функциональности



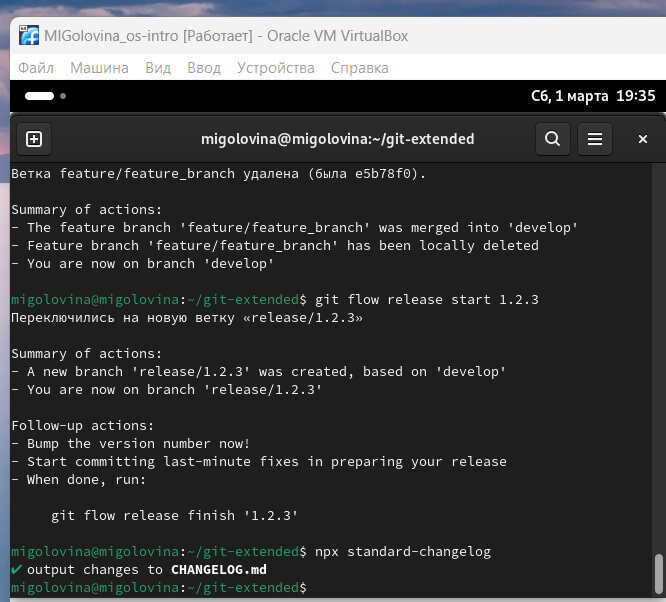
Объединение ветки feature\_branch c develop



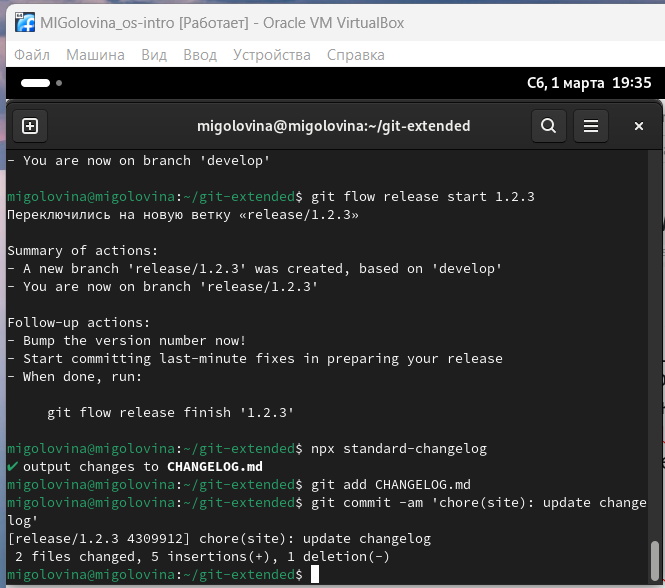
Создание релиза с версией 1.2.3:



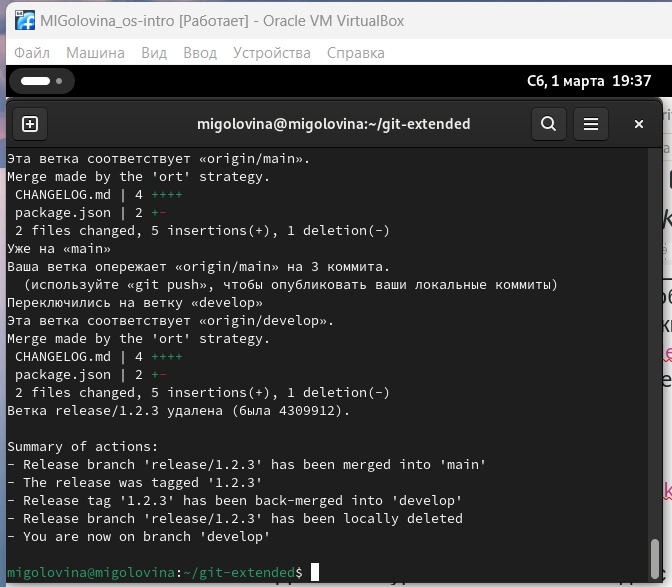
Обновление номера версии в файле package.json.



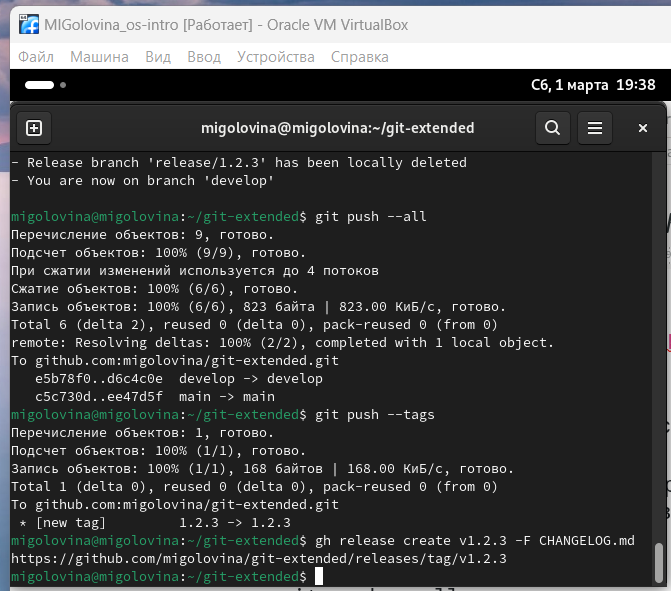
Создание журнала изменений



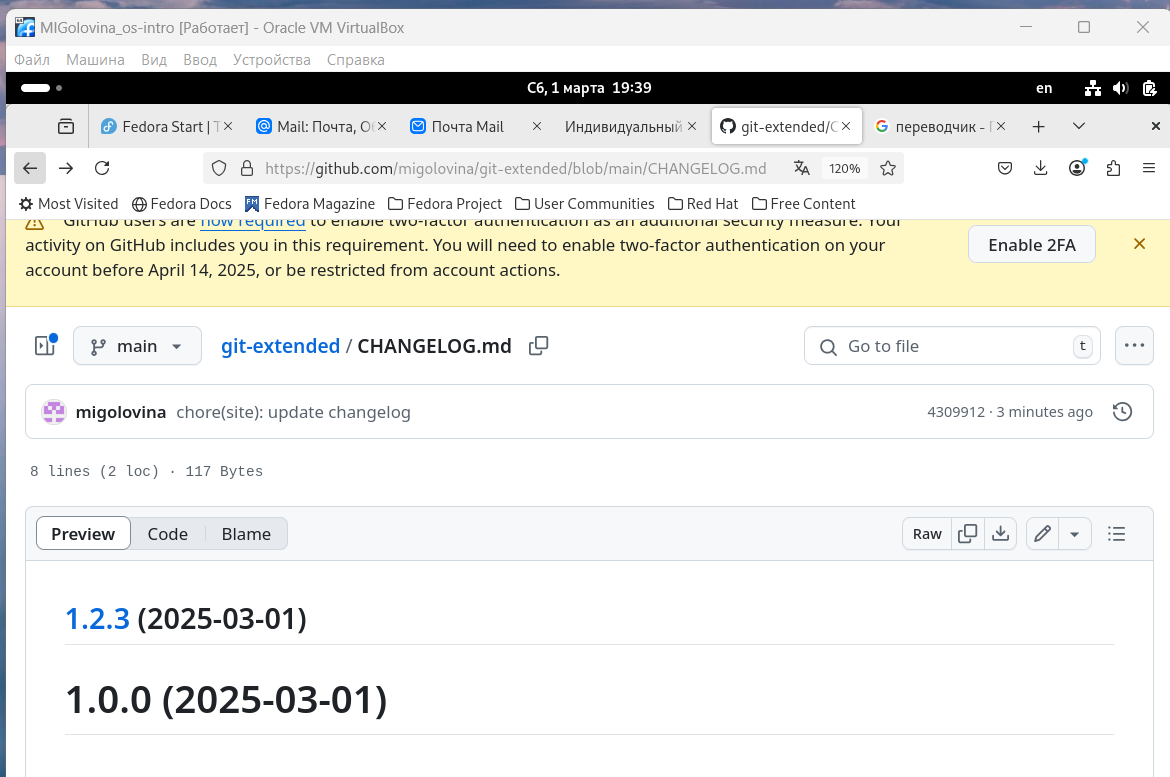
Добавление журнала изменений в индекс



Залитие релизной ветки в основную ветку



Отправление данных на github и создание релиза на github с комментарием из журнала изменений



Проверка релизов

# 5 Выводы

Я получила навыки правильной работы с репозиториями 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.