Лабораторная работа №4

Отчёт

Ермишина Мария Кирилловна

Содержание

# 1 Цель работы

Целью данной работы является получение навыков правильной работы с репозиториями git. [**tuis4?**]

# 2 Теоретическое введение

1. Gitflow Workflow опубликована и популяризована Винсентом Дриссеном.
2. Gitflow Workflow предполагает выстраивание строгой модели ветвления с учётом выпуска проекта.
3. Данная модель отлично подходит для организации рабочего процесса на основе релизов.
4. Работа по модели Gitflow включает создание отдельной ветки для исправлений ошибок в рабочей среде.
5. 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.

# 3 Выполнение лабораторной работы

1. Установка git-flow Установка из коллекции репозиториев Copr происходит благодаря следующим командам: (рис. 1)

* sudo dnf copr enable elegos/gitflow
* sudo dnf install gitflow

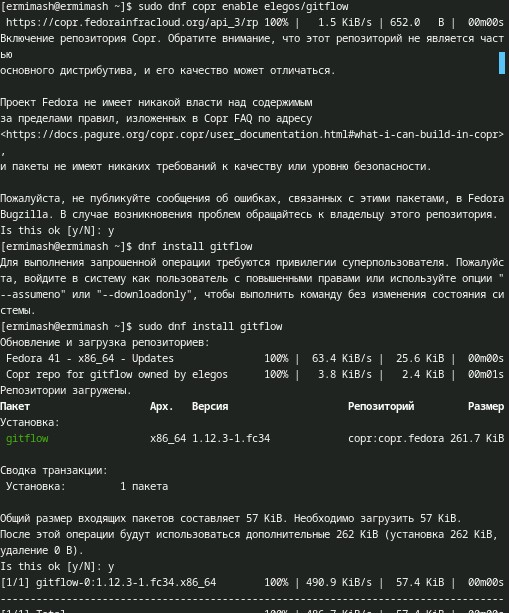


Рис. 1: Установка git-flow

1. Установка Node.js На Node.js базируется программное обеспечение для семантического версионирования и общепринятых коммитов. Устанавливаем с помощью команд: (рис. 2)

* dnf install nodejs
* dnf install pnpm

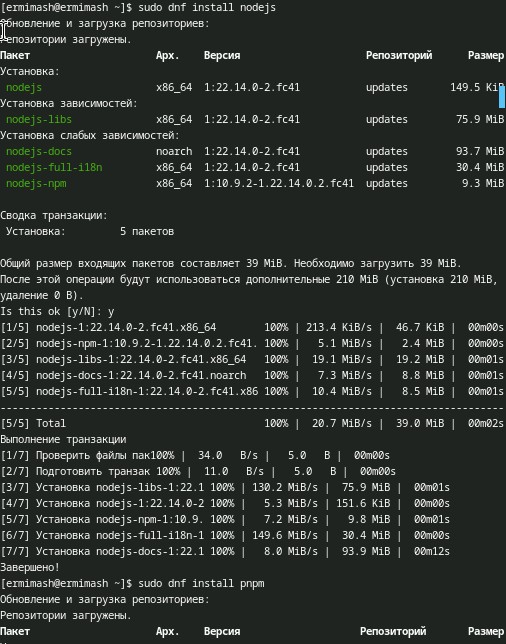


Рис. 2: Установка Node.js

1. Настройка Node.js Для работы с Node.js добавим каталог с исполняемыми файлами, устанавливаемыми yarn, в переменную PATH. Сначала запускаем, а после перелогиниваемся и выполняем: (рис. 3)

* pnpm setup
* source ~/.bashrc

1. Общепринятые коммиты
2. commitizen: (рис. 3)

* pnpm add -g commitizen

1. standard-changelog (рис. 3)

* pnpm add -g standard-changelog

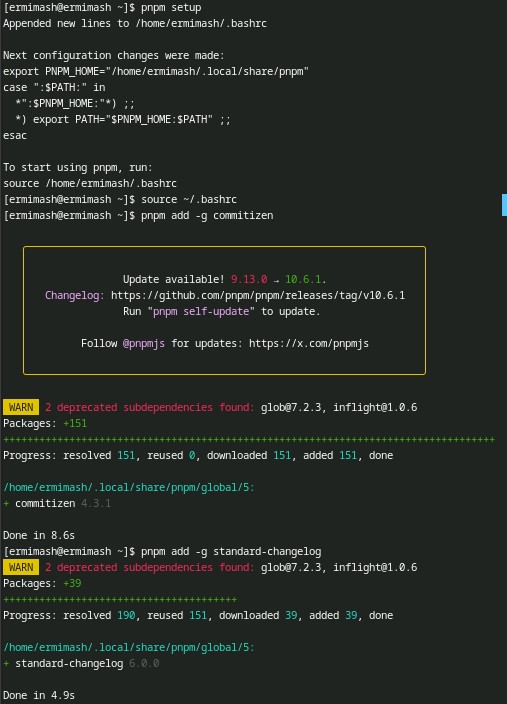


Рис. 3: Настройка Node.js

1. Создание репозитория git
2. Подключение репозитория к github Создаём репозиторий на GitHub. Для примера назовём его git-extended. Делаем первый коммит и выкладываем на github: (рис. 4)

* git commit -m “first commit”
* git remote add origin (SSH репозитория)
* git push -u origin master (используем main, так как не задавли название ветки)

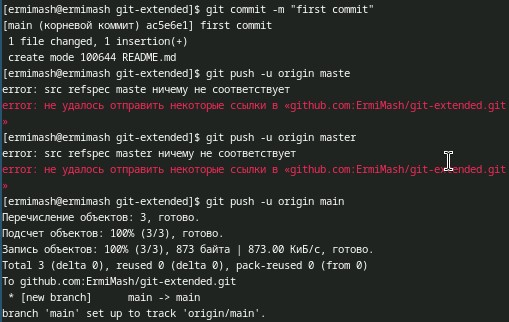


Рис. 4: Создание репозитория

1. Конфигурация общепринятых коммитов Конфигурация для пакетов Node.js выполняется с помощью команды pnpm init. (рис. **¿fig:005?**) В файле нам необходимо изменить данные, приводим файл к следующему виду: (рис. 5)

{ “name”: “git-extended”, “version”: “1.0.0”, “description”: “Git repo for educational purposes”, “main”: “index.js”, “repository”: “git@github.com:username/git-extended.git”, “author”: “Name Surname [username@gmail.com](mailto:username@gmail.com)”, “license”: “CC-BY-4.0”, “config”: { “commitizen”: { “path”: “cz-conventional-changelog” } } }

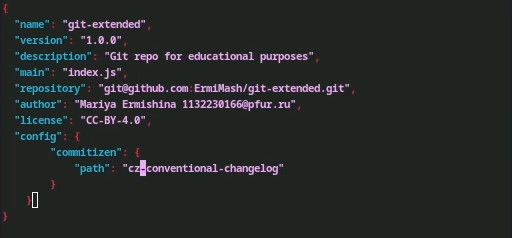
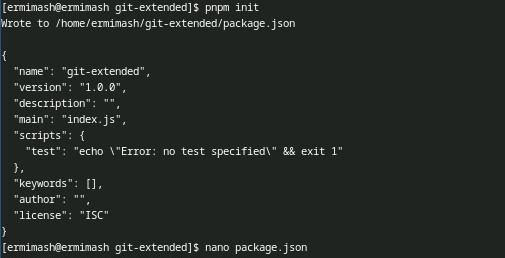
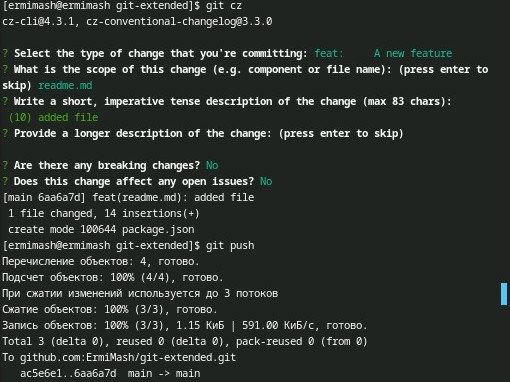


Рис. 5: Структура файла

Далее отправляем файлы на Git: (рис. **¿fig:005?**), (рис. **¿fig:007?**) - git add . (добавим новые файлы) - git cz (выполним коммит) - git push (отправим на github)

1. Конфигурация git-flow Инициализируем git-flow с помощью git flow init. Префикс для ярлыков установим в v. После проверяем нахождение на ветке develop: (рис. **¿fig:008?**)

* git branch

Загрузите весь репозиторий в хранилище с помощью git push –all. (рис. **¿fig:009?**)

Установите внешнюю ветку как вышестоящую для этой ветки: (рис. **¿fig:009?**) - git branch –set-upstream-to=origin/develop develop

Создадим релиз с версией 1.0.0: (рис. **¿fig:009?**) - git flow release start 1.0.0

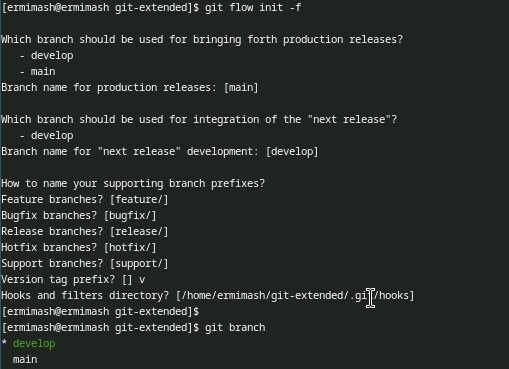
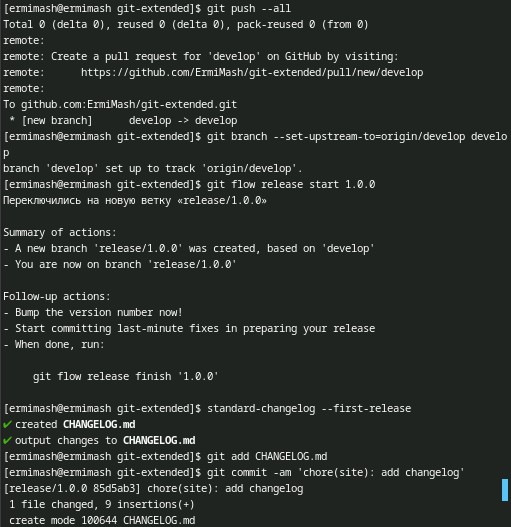
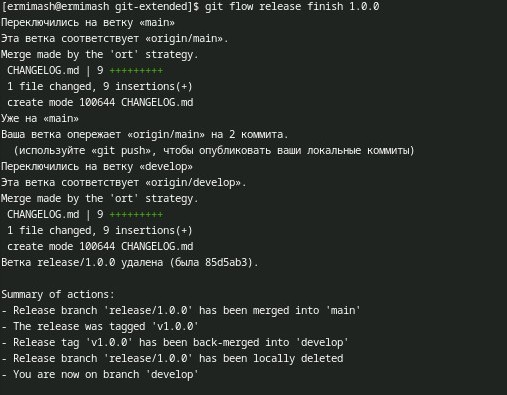
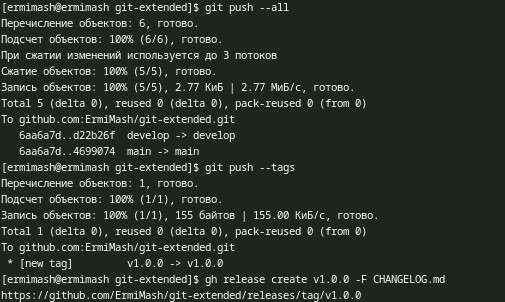
Создадим журнал изменений: (рис. **¿fig:009?**) - standard-changelog –first-release

Добавим журнал изменений в индекс: (рис. **¿fig:009?**) - git add CHANGELOG.md - git commit -am ‘chore(site): add changelog’

Зальём релизную ветку в основную ветку: (рис. **¿fig:010?**) - git flow release finish 1.0.0

Отправим данные на github: (рис. **¿fig:011?**) - git push –all - git push –tags

Создадим релиз на github. Для этого будем использовать утилиты работы с github: (рис. **¿fig:011?**) - gh release create v1.0.0 -F CHANGELOG.md

1. Работа с репозиторием git
2. Разработка новой функциональности (рис. 6) Создадим ветку для новой функциональности:

* git flow feature start feature\_branch

Далее, продолжаем работу c git как обычно. По окончании разработки новой функциональности следующим шагом следует объединить ветку feature\_branch c develop: - git flow feature finish feature\_branch

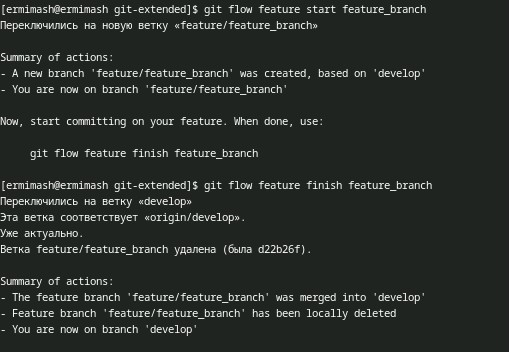


Рис. 6: Новая ветка

1. Создание релиза git-flow Создадим релиз с версией 1.2.3: (рис. 7)

* git flow release start 1.2.3

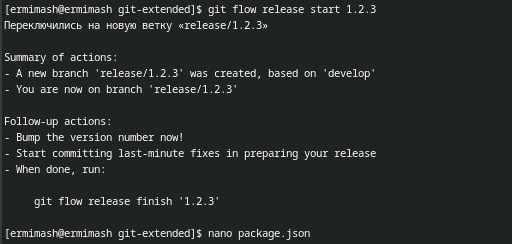


Рис. 7: Создание релиза

Обновляем номер версии в файле package.json. Установите её в 1.2.3. (рис. 8)

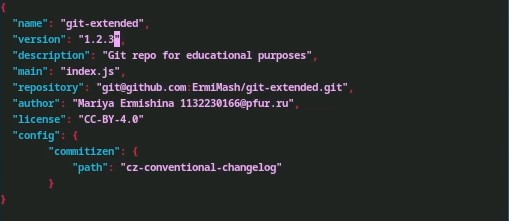


Рис. 8: Обновление инф. в файле

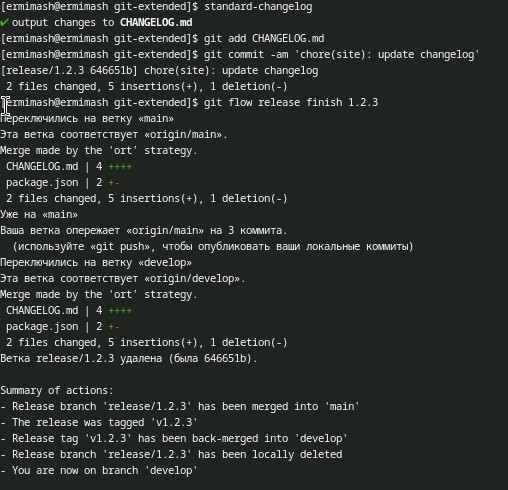
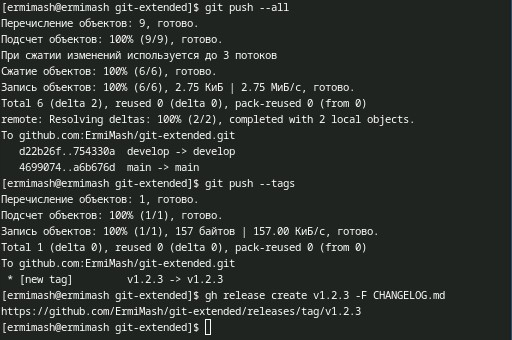
Создадим журнал изменений: (рис. **¿fig:015?**) - standard-changelog

Добавим журнал изменений в индекс: (рис. **¿fig:015?**) - git add CHANGELOG.md - git commit -am ‘chore(site): update changelog’

Зальём релизную ветку в основную ветку: (рис. **¿fig:015?**) - git flow release finish 1.2.3

Отправим данные на github: (рис. **¿fig:016?**) - git push –all - git push –tags

Создадим релиз на github с комментарием из журнала изменений: (рис. **¿fig:016?**) - gh release create v1.2.3 -F CHANGELOG.md

# 4 Выводы

В ходе данной работы были получены навыки правильной работы с репозиториями git.