Отчет по лабораторной работе №4

Операционные системы

Дмитрий Павлович Стрижов

Содержание

List of Figures

List of Tables

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 выпу
Данная модель отлично подходит для организации рабочего процесса на основе релизо
Работа по модели Gitflow включает создание отдельной ветки для исправлений ошибок
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 devel
- Если в master обнаружена проблема, из master создаётся ветка hotfix.
- Когда работа над веткой исправления hotfix завершена, она сливается в ветки d

Процесс работы с Gitflow

Основные ветки (master) и ветки разработки (develop)

- Для фиксации истории проекта в рамках этого процесса вместо одной ветки maste

При использовании библиотеки расширений git-flow нужно инициализировать струк

```
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 можн
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

```
git flow feature finish feature_branch
```

Ветки выпуска (release)

Когда в ветке develop оказывается достаточно функций для выпуска, из ветки de

Благодаря тому, что для подготовки выпусков используется специальная ветка, с

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 исправлен

Наличие специальной ветки для исправления ошибок позволяет команде решать про

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 версионирова

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 с

ПАТЧ-версию, когда вы делаете обратно совместимые исправления.

Дополнительные обозначения для предрелизных и билд-метаданных возможны как до

Программное обеспечение

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

Пакет Conventional Changelog

Пакет Conventional Changelog является комплексным решением по управлению комм

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 версиони
Регламентирует структуру и основные типы коммитов.

Структура коммита

<type>(<scope>): <subject>
<BLANK LINE>

<body>

<BLANK LINE>

<footer>

Или, по-русски:

<тип>(<область>): <описание изменения>

<пустая линия>

[необязательное тело]

<пустая линия>

[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» ("ch

Не используйте заглавную первую букву.

Не ставьте точку в конце.

Тело (body) должно включать мотивацию к изменению и противопоставлять это пре

Как и в теме, используйте повелительное наклонение в настоящем времени.

Нижний колонтитул (footer) должен содержать любую информацию о критических из

Следует использовать для указания внешних ссылок, контекста коммита или д

Также содержит ссылку на issue (например, на github), который закрывает э

Критические изменения должны начинаться со слова BREAKING CHANGE: с пробе

Типы коммитов

Базовые типы коммитов

fix: – коммит типа fix исправляет ошибку (bug) в вашем коде (он соответст

feat: — коммит типа feat добавляет новую функцию (feature) в ваш код (он

BREAKING CHANGE: — коммит, который содержит текст BREAKING CHANGE: в нача

revert: — если фиксация отменяет предыдущую фиксацию. Начинается с revert хэш отменяемой фиксации).

Другое: коммиты с типами, которые отличаются от fix: и feat:, также разре conventional (основанный на The Angular convention) рекомендует: chore:, docs:, s

Соглашения The Angular convention

Одно из популярных соглашений о поддержке исходных кодов — конвенция Angu

Типы коммитов The Angular convention

Конвенция Angular (The Angular convention) требует следующие типы коммито

build: — изменения, влияющие на систему сборки или внешние зависимост

ci: — изменения в файлах конфигурации и скриптах CI (примеры областей

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 добавля

style: — изменения, не влияющие на смысл кода (пробелы, форматировани

test: — добавление недостающих тестов или исправление существующих те

Области действия (scope)

Областью действия должно быть имя затронутого пакета npm (как его воспри

Есть несколько исключений из правила «использовать имя пакета»:

packaging — используется для изменений, которые изменяют структуру па

changelog — используется для обновления примечаний к выпуску в CHANGE

отсутствует область действия — полезно для изменений стиля, тестирова

Соглашения @commitlint/config-conventional

Соглашение @commitlint/config-conventional входит в пакет Conventional Change

4 Выполнение лабораторной работы

Устанавливаем gitflow (рис. 4.1, 4.2).

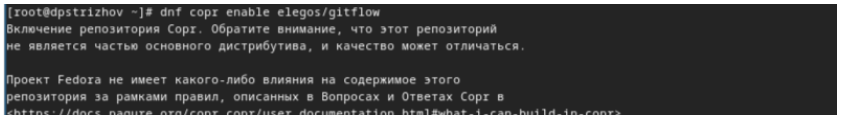

Рис. 4.1: Подключение репозитория copr

Рис. 4.2: Установка gitflow

Устанавливаем nodejs (рис. 4.3, 4.4).

Рис. 4.3: Установка nodejs

Рис. 4.4: Установка pnpm

Запускаем pnpm (рис. 4.5).

Рис. 4.5: Запуск pnpm

Перелогинимся с помощью команды source,а затем добавляем программу для помощи в формировании коммитов (рис. 4.6).

Рис. 4.6: Продолжение настройки nodejs

Добавляем программу для помощи в создании логов (рис. 4.7).

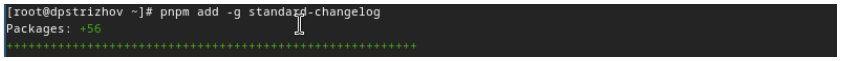

Рис. 4.7: Добавление пртграммы для поомщи в создании логов

Создаем новый репозиторий на github,делаем первый коммит и выкладываем его на github(рис. 4.8).

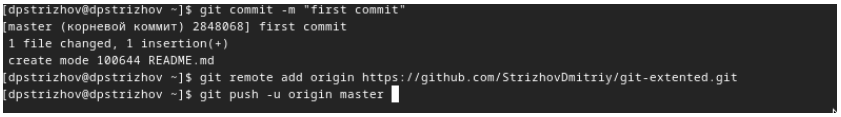

Рис. 4.8: Первый комминт репозитория git-extended

Конфигурация для пакетов Node.js (рис. 4.9).

Рис. 4.9: Конфигурация для пакетов Node.js

Изменяем конфигурации под наши нужды (рис. 4.10).

Рис. 4.10: Измененныйе конфигурации

Отправляем файлы на github(рис. 4.11, 4.12).

Рис. 4.11: Отправка файлов

Рис. 4.12: Отправка файлов

Инициализируем git-flow(рис. 4.13).

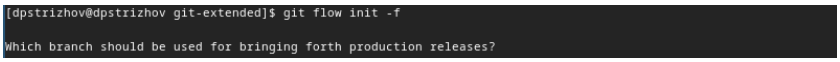

Рис. 4.13: Инициализация git-flow

Проверяем ветку, на которой мы находимся(рис. 4.14).

Рис. 4.14: Проверка ветки

Загружаем весь репозиторий(рис. 4.15).

Рис. 4.15: Загрузка всего репозитория

Создаем релиз версии 1.0.0(рис. 4.16).

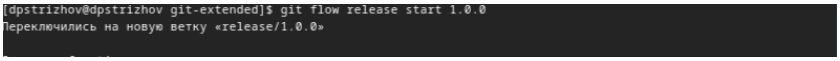

Рис. 4.16: Создание релиза версии 1.0.0

Создаем журнал изменений(рис. 4.17).

Рис. 4.17: Создание журнала изменений

Добавляем журнал изменений в индекс(рис. 4.18).

Рис. 4.18: Добавление журнала изменений в индекс

Отправим данные и создадим релиз(рис. 4.19).

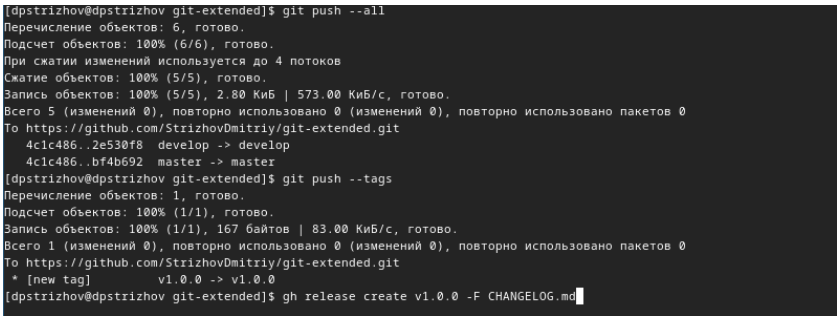

Рис. 4.19: Отправка данных и создание релиза

Создаем ветку для новой функциональности и объединеняем данную ветку с develop (рис. 4.20).

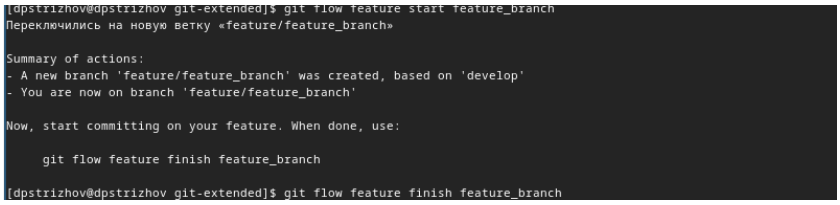

Рис. 4.20: Создание ветки для новой функциональности и объединение данной ветки с develop

Создаем релиз версии 1.2.3 (рис. 4.21).

Рис. 4.21: Создание релиза версии 1.2.3

Создаем журнал изменений (рис. 4.22).

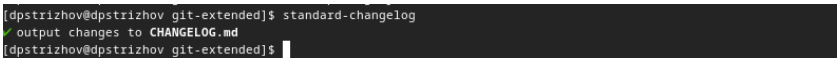

Рис. 4.22: Создание журнала изменений

Добавляем журнал изменений в индекс и заливаем релизную ветку в основную (рис. 4.23).

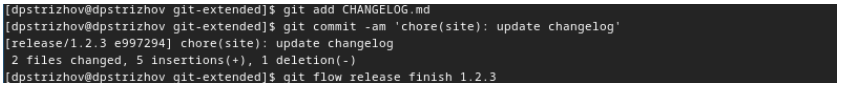

Рис. 4.23: Добавление жернала изменений в индекс

Отправляем данные на github(рис. 4.24).

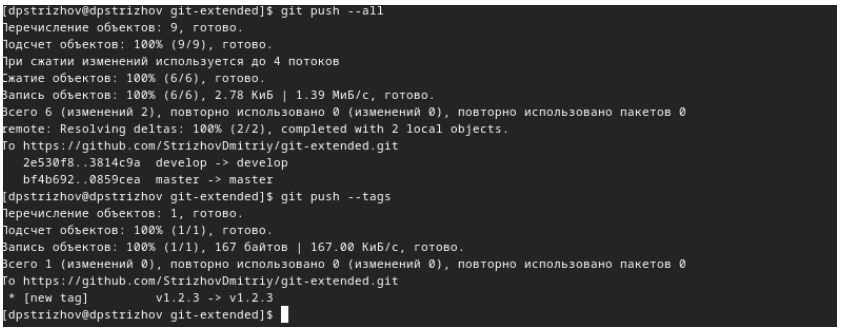

Рис. 4.24: Отправка данных на github

Создаем релиз версии 1.2.3(рис. 4.25).

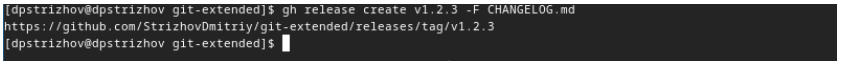

Рис. 4.25: Создание релиза версии 1.2.3

5 Выводы

В процессе выполнения данной лабораторной работы я научился правильной работе с репозиториями git.

Список литературы

Лабораторная работа №4: https://esystem.rudn.ru/mod/page/view.php?id=1098794

::: {#refs} :::