

Отчет по лабораторной работе №4

Операционные системы

Дмитрий Павлович Стрижов

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	14
5	Выводы	19
	Список литературы	20

List of Figures

4.1	Подключение репозитория copr	14
4.2	Установка gitflow	14
4.3	Установка nodejs	14
4.4	Установка npnm	14
4.5	Запуск npnm	14
4.6	Продолжение настройки nodejs	15
4.7	Добавление пртгграммы для поомщи в создании логов	15
4.8	Первый коммит репозитория git-extended	15
4.9	Конфигурация для пакетов Node.js	15
4.10	Измененныйе конфигурации	15
4.11	Отправка файлов	15
4.12	Отправка файлов	16
4.13	Инициализация git-flow	16
4.14	Проверка ветки	16
4.15	Загрузка всего репозитория	16
4.16	Создание релиза версии 1.0.0	16
4.17	Создание журнала изменений	16
4.18	Добавление журнала изменений в индекс	16
4.19	Отправка данных и создание релиза	17
4.20	Создание ветки для новой функциональности и объединение дан- ной ветки с develop	17
4.21	Создание релиза версии 1.2.3	17
4.22	Создание журнала изменений	17
4.23	Добавление жернала изменений в индекс	18
4.24	Отправка данных на github	18
4.25	Создание релиза версии 1.2.3	18

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 gitflow.

Общая информация

Gitflow Workflow опубликована и популяризована Винсентом Дриссенем.

Gitflow Workflow предполагает выстраивание строгой модели ветвления с учётом выпуска релизов.

Данная модель отлично подходит для организации рабочего процесса на основе релизов.

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 develop.

- Если в master обнаружена проблема, из master создаётся ветка hotfix.

- Когда работа над веткой исправления hotfix завершена, она сливается в ветки master и develop.

Процесс работы с Gitflow

Основные ветки (master) и ветки разработки (develop)

Для фиксации истории проекта в рамках этого процесса вместо одной ветки master

При использовании библиотеки расширений git-flow нужно инициализировать структуру

```
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

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 develop создаётся новая ветка release. 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 исправлений. 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 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 с

ПАТЧ-версию, когда вы делаете обратно совместимые исправления.

Дополнительные обозначения для пререлизных и билд-метаданных возможны как до

Программное обеспечение

Для реализации семантического версионирования создано несколько программных продук

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 версионир

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» ("change")

Не используйте заглавную первую букву.

Не ставьте точку в конце.

Тело (body) должно включать мотивацию к изменению и противопоставлять это прежнему состоянию.

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 для ссылки на issue.

Также содержит ссылку на issue (например, на github), который закрывает это изменение.

Критические изменения должны начинаться со слова BREAKING CHANGE: с пробелом.

Типы коммитов

Базовые типы коммитов

fix: — коммит типа fix исправляет ошибку (bug) в вашем коде (он соответствует

`feat:` – коммит типа `feat` добавляет новую функцию (feature) в ваш код (он начинается с `feat:`);

`BREAKING CHANGE:` – коммит, который содержит текст `BREAKING CHANGE:` в начале сообщения;

`revert:` – если фиксация отменяет предыдущую фиксацию. Начинается с `revert` и хэш отменяемой фиксации).

Другое: коммиты с типами, которые отличаются от `fix:` и `feat:`, также разрезаны на `conventional` (основанный на `The Angular convention`) рекомендует: `chore:`, `docs:`, `style:`, `test:`, `perf:`.

Соглашения `The Angular convention`

Одно из популярных соглашений о поддержке исходных кодов – конвенция `Angular`.

Типы коммитов `The Angular convention`

Конвенция `Angular` (`The Angular convention`) требует следующие типы коммитов:

- `build:` – изменения, влияющие на систему сборки или внешние зависимости;
- `ci:` – изменения в файлах конфигурации и скриптах CI (примеры областей действия);
- `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 новую функцию;
- `style:` – изменения, не влияющие на смысл кода (пробелы, форматирование);
- `test:` – добавление недостающих тестов или исправление существующих тестов.

Области действия (scope)

Областью действия должно быть имя затронутого пакета `npm` (как его воспринимает `npm`).

Есть несколько исключений из правила «использовать имя пакета»:

`packaging` – используется для изменений, которые изменяют структуру пакета.

changelog – используется для обновления примечаний к выпуску в CHANGELOG
отсутствует область действия – полезно для изменений стиля, тестирования

Соглашения @commitlint/config-conventional

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

4 Выполнение лабораторной работы

Устанавливаем gitflow (рис. 4.1, 4.2).

```
[root@dpstrizhov ~]# dnf copr enable elegos/gitflow
Включение репозитория Copr. Обратите внимание, что этот репозиторий
не является частью основного дистрибутива, и качество может отличаться.

Проект Fedora не имеет какого-либо влияния на содержимое этого
репозитория за рамками правил, описанных в Вопросах и Ответах Copr в
https://docs.fedoraproject.org/copr-copr/user-documentation.html#what-it-can-build-in-copr
```

Рис. 4.1: Подключение репозитория copr

```
[root@dpstrizhov ~]# dnf install gitflow
Copr repo for gitflow owned by elegos
3.6 kB/s | 1.5 kB    00:00
Зависимости разрешены.
```

Рис. 4.2: Установка gitflow

Устанавливаем nodejs (рис. 4.3, 4.4).

```
[root@dpstrizhov ~]# dnf install nodejs
Последняя проверка окончания срока действия метаданных: 0:01:11 назад, Пт 08 мар 2024 23:35:45.
Зависимости разрешены.
```

Рис. 4.3: Установка nodejs

```
[root@dpstrizhov ~]# dnf install pnpm
Последняя проверка окончания срока действия метаданных: 0:05:25 назад, Пт 08 мар 2024 23:35:45.
Зависимости разрешены.
```

Рис. 4.4: Установка pnpm

Запускаем pnpm (рис. 4.5).

```
[root@dpstrizhov ~]# pnpm setup
Appended new lines to /root/.bashrc
```

Рис. 4.5: Запуск pnpm

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

```
[root@dpstrizhov ~]# source ~/.bashrc
[root@dpstrizhov ~]# pnpm add -g commitizen
```

Рис. 4.6: Продолжение настройки nodejs

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

```
[root@dpstrizhov ~]# pnpm add -g standard-changelog
Packages: +56
*****
```

Рис. 4.7: Добавление программы для помощи в создании логов

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

```
dpstrizhov@dpstrizhov ~]$ git commit -m "first commit"
[master (корневой коммит) 2848068] first commit
1 file changed, 1 insertion(+)
create mode 100644 README.md
dpstrizhov@dpstrizhov ~]$ git remote add origin https://github.com/StrizhovDmitriy/git-extended.git
dpstrizhov@dpstrizhov ~]$ git push -u origin master
```

Рис. 4.8: Первый коммит репозитория git-extended

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

```
dpstrizhov@dpstrizhov ~]$ pnpm init
Wrote to /home/dpstrizhov/package.json
```

Рис. 4.9: Конфигурация для пакетов Node.js

Изменяем конфигурации под наши нужды (рис. 4.10).

```
{
  "name": "git-extended",
  "version": "1.0.0",
  "description": "git repo for education purposes",
  "main": "index.js",
  "repository": "https://github.com/StrizhovDmitriy/git-extended.git",
  "author": "StrizhovDmitriy <1132236054@pfur.ru>",
  "license": "CC-BY-4.0",
  "config": {
    "commitizen": {
      "path": "cz-conventional-changelog"
    }
  }
}
```

Рис. 4.10: Измененные конфигурации

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

```
dpstrizhov@dpstrizhov git-extended]$ git add .
dpstrizhov@dpstrizhov git-extended]$ git cz
cz-cli@4.3.0, cz-conventional-changelog@3.3.0
```

Рис. 4.11: Отправка файлов

```
[dpstrizhov@dpstrizhov git-extended]$ git push
Перечисление объектов: 4, готово.
Подсчет объектов: 100% (4/4), готово.
Всего объектов: 4, готово.
Всего изменений: 4, готово.
```

Рис. 4.12: Отправка файлов

Инициализируем git-flow(рис. 4.13).

```
[dpstrizhov@dpstrizhov git-extended]$ git flow init -f
Which branch should be used for bringing forth production releases?
```

Рис. 4.13: Инициализация git-flow

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

```
[dpstrizhov@dpstrizhov git-extended]$ git branch
* develop
master
```

Рис. 4.14: Проверка ветки

Загружаем весь репозиторий(рис. 4.15).

```
[dpstrizhov@dpstrizhov git-extended]$ git push --all
To https://github.com/StrizhovDmitriy/git-extended.git
! [rejected] master -> master (non-fast-forward)
```

Рис. 4.15: Загрузка всего репозитория

Создаем релиз версии 1.0.0(рис. 4.16).

```
[dpstrizhov@dpstrizhov git-extended]$ git flow release start 1.0.0
Переключились на новую ветку «release/1.0.0»
```

Рис. 4.16: Создание релиза версии 1.0.0

Создаем журнал изменений(рис. 4.17).

```
[dpstrizhov@dpstrizhov git-extended]$ standard-changelog --first-release
✓ created CHANGELOG.md
✓ output changes to CHANGELOG.md
```

Рис. 4.17: Создание журнала изменений

Добавляем журнал изменений в индекс(рис. 4.18).

```
[dpstrizhov@dpstrizhov git-extended]$ git add CHANGELOG.md
[dpstrizhov@dpstrizhov git-extended]$ git commit -am 'chore(site): add changelog'
[release/1.0.0 f418d70] chore(site): add changelog
1 file changed, 9 insertions(+)
create mode 100644 CHANGELOG.md
[dpstrizhov@dpstrizhov git-extended]$
```

Рис. 4.18: Добавление журнала изменений в индекс

Отправим данные и создадим релиз(рис. 4.19).

```
[dpstrizhov@dpstrizhov git-extended]$ git push --all
Перечисление объектов: 6, готово.
Подсчет объектов: 100% (6/6), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (5/5), готово.
Запись объектов: 100% (5/5), 2.80 КиБ | 573.00 КиБ/с, готово.
Всего 5 (изменений 0), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
To https://github.com/StrizhovDmitriy/git-extended.git
4c1c486..2e530f8 develop -> develop
4c1c486..bf4b692 master -> master
[dpstrizhov@dpstrizhov git-extended]$ git push --tags
Перечисление объектов: 1, готово.
Подсчет объектов: 100% (1/1), готово.
Запись объектов: 100% (1/1), 167 байтов | 83.00 КиБ/с, готово.
Всего 1 (изменений 0), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
To https://github.com/StrizhovDmitriy/git-extended.git
* [new tag] v1.0.0 -> v1.0.0
[dpstrizhov@dpstrizhov git-extended]$ gh release create v1.0.0 -F CHANGELOG.md
```

Рис. 4.19: Отправка данных и создание релиза

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

```
[dpstrizhov@dpstrizhov git-extended]$ git flow feature start feature_branch
Переключились на новую ветку «feature/feature_branch»

Summary of actions:
- A new branch 'feature/feature_branch' was created, based on 'develop'
- You are now on branch 'feature/feature_branch'

Now, start committing on your feature. When done, use:

    git flow feature finish feature_branch

[dpstrizhov@dpstrizhov git-extended]$ git flow feature finish feature_branch
```

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

Создаем релиз версии 1.2.3 (рис. 4.21).

```
[dpstrizhov@dpstrizhov git-extended]$ git flow release start 1.2.3
Переключились на новую ветку «release/1.2.3»
```

Рис. 4.21: Создание релиза версии 1.2.3

Создаем журнал изменений (рис. 4.22).

```
[dpstrizhov@dpstrizhov git-extended]$ standard-changelog
output changes to CHANGELOG.md
[dpstrizhov@dpstrizhov git-extended]$
```

Рис. 4.22: Создание журнала изменений

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

```
[dpstrizhov@dpstrizhov git-extended]$ git add CHANGELOG.md
[dpstrizhov@dpstrizhov git-extended]$ git commit -am 'chore(site): update changelog'
[release/1.2.3 e997294] chore(site): update changelog
2 files changed, 5 insertions(+), 1 deletion(-)
[dpstrizhov@dpstrizhov git-extended]$ git flow release finish 1.2.3
```

Рис. 4.23: Добавление журнала изменений в индекс

Отправляем данные на github(рис. 4.24).

```
[dpstrizhov@dpstrizhov git-extended]$ git push --all
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (6/6), 2.78 КиБ | 1.39 МБ/с, готово.
Всего 6 (изменений 2), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/StrizhovDmitriy/git-extended.git
  2e530f8..3814c9a develop -> develop
  bf4b692..0859cea master -> master
[dpstrizhov@dpstrizhov git-extended]$ git push --tags
Перечисление объектов: 1, готово.
Подсчет объектов: 100% (1/1), готово.
Запись объектов: 100% (1/1), 167 байтов | 167.00 КиБ/с, готово.
Всего 1 (изменений 0), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
To https://github.com/StrizhovDmitriy/git-extended.git
 * [new tag]          v1.2.3 -> v1.2.3
[dpstrizhov@dpstrizhov git-extended]$
```

Рис. 4.24: Отправка данных на github

Создаем релиз версии 1.2.3(рис. 4.25).

```
[dpstrizhov@dpstrizhov git-extended]$ gh release create v1.2.3 -F CHANGELOG.md
https://github.com/StrizhovDmitriy/git-extended/releases/tag/v1.2.3
[dpstrizhov@dpstrizhov git-extended]$
```

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} :::