

Лабораторная работа №4

Продвинутое использование git

Mohamed Musa

Содержание

1	Цель работы	5
2	Теоретическое введение	6
2.1	Рабочий процесс Gitflow	6
2.2	Семантическое версионирование	6
2.3	Conventional Commits	7
3	Выполнение лабораторной работы	8
3.1	Установка программного обеспечения	8
3.2	Создание репозитория git	10
3.3	Работа с репозиторием git	14
4	Выводы	17
	Список литературы	19

Список иллюстраций

3.1	Настройка <code>pnpm</code>	9
3.2	Установка <code>commitizen</code>	10
3.3	Конфигурация <code>package.json</code>	11
3.4	Создание первого релиза	13
3.5	Работа с <code>changelog</code>	15
3.6	Добавление записей в <code>changelog</code>	15

Список таблиц

1 Цель работы

Получение навыков правильной работы с репозиториями git. Изучение работы с git-flow, семантическим версионированием и conventional commits.

2 Теоретическое введение

2.1 Рабочий процесс Gitflow

Gitflow Workflow — это модель ветвления Git, опубликованная и популяризованная Винсентом Дриссенем. Она предполагает выстраивание строгой модели ветвления с учётом выпуска проекта и отлично подходит для организации рабочего процесса на основе релизов.

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

2.2 Семантическое версионирование

Семантическое версионирование — это формальное соглашение о том, как назначать и увеличивать номера версий. Версия задаётся в виде кортежа

МАЖОРНАЯ_ВЕРСИЯ.МИНОРНАЯ_ВЕРСИЯ.ПАТЧ:

- МАЖОРНУЮ версию увеличивают, когда сделаны обратно несовместимые изменения API
- МИНОРНУЮ версию увеличивают, когда добавляется новая функциональность, не нарушая обратной совместимости
- ПАТЧ-версию увеличивают, когда делаются обратно совместимые исправления

2.3 Conventional Commits

Conventional Commits — это соглашение о том, как нужно писать сообщения коммитов. Оно совместимо с SemVer и регламентирует структуру и основные типы коммитов.

Структура коммита:

```
<тип> (<описание>) : <заголовок>
<тег> <исходный код>
[ <ссылка на документацию> ]
<исходный код>
[ <ссылка на документацию> <исходный код> <ссылка на документацию> ]
```

Основные типы коммитов:

- **feat:** — добавление новой функции
- **fix:** — исправление ошибки
- **docs:** — изменения только в документации
- **style:** — изменения форматирования кода
- **refactor:** — рефакторинг кода
- **test:** — добавление или исправление тестов
- **chore:** — изменения в процессе сборки или вспомогательных инструментах

3 Выполнение лабораторной работы

3.1 Установка программного обеспечения

3.1.1 Установка git-flow

Для работы с моделью Gitflow необходимо установить пакет git-flow. В системе Fedora установка выполняется следующими командами:

```
dnf copr enable elegos/gitflow
dnf install gitflow
```

3.1.2 Установка Node.js

Node.js необходим для работы с инструментами семантического версионирования и conventional commits:

```
dnf install nodejs
dnf install npm
```

3.1.3 Настройка Node.js

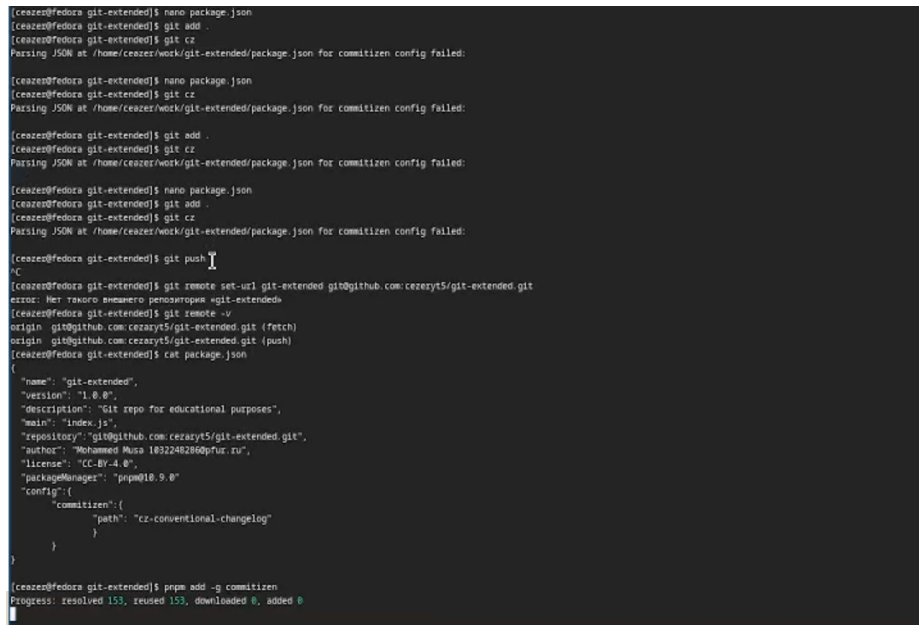
После установки Node.js необходимо настроить окружение для работы с npm. Выполняем команду настройки:

pnpm setup

Затем перезагружаем конфигурацию оболочки:

```
source ~/.bashrc
```

На рисунке Рисунок 3.1 показан процесс настройки pnpm.



```
(ceazer@fedora git-extended)$ nano package.json
(ceazer@fedora git-extended)$ git add .
(ceazer@fedora git-extended)$ git cz
Parsing JSON at /home/ceazer/work/git-extended/package.json for commitizen config failed:

(ceazer@fedora git-extended)$ nano package.json
(ceazer@fedora git-extended)$ git cz
Parsing JSON at /home/ceazer/work/git-extended/package.json for commitizen config failed:

(ceazer@fedora git-extended)$ nano package.json
(ceazer@fedora git-extended)$ git add .
(ceazer@fedora git-extended)$ git cz
Parsing JSON at /home/ceazer/work/git-extended/package.json for commitizen config failed:

(ceazer@fedora git-extended)$ git push
^C
(ceazer@fedora git-extended)$ git remote set-url git@github.com:ceazyt5/git-extended.git
(ceazer@fedora git-extended)$ git remote -v
origin  git@github.com:ceazyt5/git-extended.git (fetch)
origin  git@github.com:ceazyt5/git-extended.git (push)
(ceazer@fedora git-extended)$ cat package.json
{
  "name": "git-extended",
  "version": "1.0.0",
  "description": "Git repo for educational purposes",
  "main": "index.js",
  "repository": "git@github.com:ceazyt5/git-extended.git",
  "author": "Mohammed Musa 1832248286@pwr.ru",
  "license": "CC-BY-4.0",
  "packageManager": "pnpm@10.9.0",
  "config": {
    "commitizen": {
      "path": "cz-conventional-changelog"
    }
  }
}
(ceazer@fedora git-extended)$ pnpm add -g commitizen
Progress: resolved 153, reused 153, downloaded 0, added 0
```

Рисунок 3.1: Настройка pnpm

3.1.4 Установка программ для conventional commits

Устанавливаем commitizen — программу для помощи в форматировании КОММИТОВ:

```
pnpm add -g commitizen
```

При этом устанавливается скрипт git-cz, который используется для создания КОММИТОВ.

Также устанавливаем standard-changelog для автоматического создания журнала изменений:

```
pnpm add -g standard-changelog
```

Процесс установки commitizen показан на рисунке Рисунок 3.2.

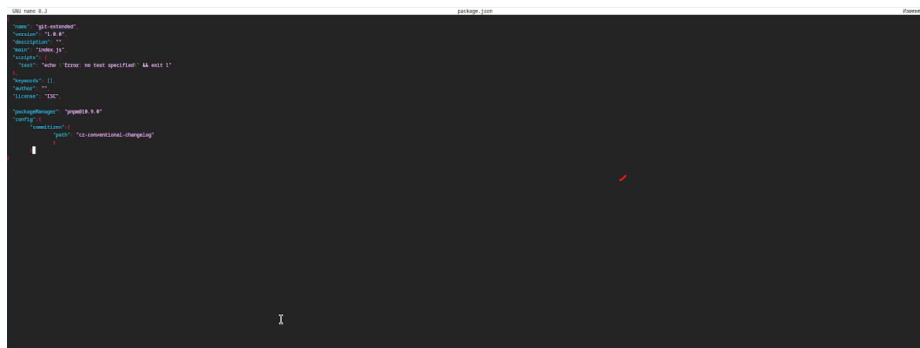


Рисунок 3.2: Установка commitizen

3.2 Создание репозитория git

3.2.1 Инициализация репозитория

Создаем новый репозиторий на GitHub с именем git-extended. Затем выполняем первоначальную настройку локального репозитория:

```
git init
git add .
git commit -m "first commit"
git remote add origin git@github.com:cezaryt5/git-extended.git
git push -u origin master
```

3.2.2 Конфигурация для Node.js пакетов

Инициализируем проект Node.js:

pnpm init

При инициализации заполняем следующие параметры:

- Название пакета: git-extended
- Версия: 1.0.0
- Описание: Git repo for educational purposes
- Автор: Mohamed Musa 1032248286@pfur.ru
- Лицензия: CC-BY-4.0

Добавляем в файл package.json конфигурацию для commitizen:

```
"config": {  
  "commitizen": {  
    "path": "cz-conventional-changelog"  
  }  
}
```

Итоговый файл package.json показан на рисунке Рисунок 3.3.

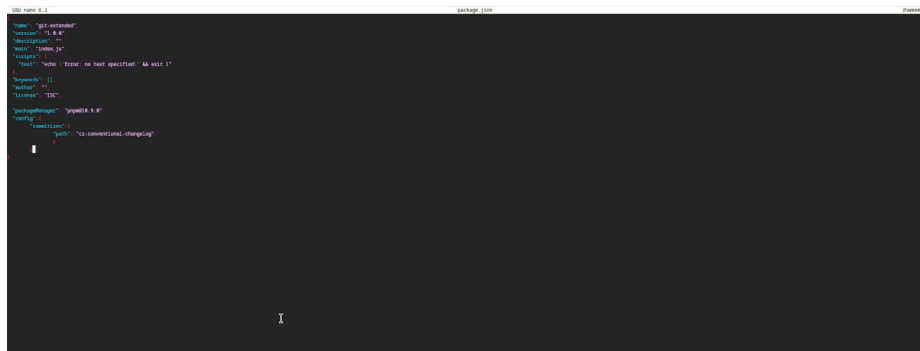


Рисунок 3.3: Конфигурация package.json

Добавляем файлы в индекс и выполняем коммит с использованием git-cz:

```
git add .  
git cz
```

Отправляем изменения на GitHub:

```
git push
```

3.2.3 Конфигурация git-flow

Инициализируем git-flow в репозитории:

```
git flow init
```

При инициализации устанавливаем префикс для тегов версий в значение v.

Проверяем, что находимся на ветке develop:

```
git branch
```

Загружаем все ветки в удаленный репозиторий:

```
git push --all
```

Устанавливаем внешнюю ветку как вышестоящую:

```
git branch --set-upstream-to=origin/develop develop
```

3.2.4 Создание первого релиза

Создаем релиз с версией 1.0.0:

```
git flow release start 1.0.0
```

Создаем журнал изменений:

```
standard-changelog --first-release
```

Добавляем журнал изменений в индекс и делаем коммит:

```
git add CHANGELOG.md
git commit -am 'chore(site): add changelog'
```

Завершаем релиз:

```
git flow release finish 1.0.0
```

Процесс создания первого релиза показан на рисунке Рисунок 3.4.

```
$ git merge master --no-commit
$ git push --set-upstream origin develop
To have this happen automatically for branches without a tracking upstream, see "push.autoSetupRemote" in "git help config".
[ce9eef8afa git-extend@] git push --set-upstream origin develop
C
[ce9eef8afa git-extend@] git push --all
W
[ce9eef8afa git-extend@] git mv package.json
[ce9eef8afa git-extend@] git cz
No files added so staging did your friend to get you set up!
[ce9eef8afa git-extend@] git push
Pull: The current branch develop has no upstream branch.
Push: The current branch and set the remote as upstream, use
    $ git push --set-upstream origin develop
To have this happen automatically for branches without a tracking upstream, see "push.autoSetupRemote" in "git help config".
[ce9eef8afa git-extend@] git push --all
Total 0 (delta #), reused 0 (delta #), pack-reused 0 (from #)
remote: Create a pull request for 'develop' on Github by visiting:
remote:   https://github.com/coryjerr/git-extended/pull/new/develop
remote: 
github.com:coryjerr/git-extended@git
# [new branch]      develop -> develop
[ce9eef8afa git-extend@] git branch --set-upstream-to=origin/develop develop
Partial setup succeeded via cpx7tool
[ce9eef8afa git-extend@] git branch --set-upstream-to=origin/develop develop
branch 'develop' set up to track 'origin/develop'
[ce9eef8afa git-extend@] git branch --set-upstream-to=origin/develop develop
branch 'develop' set up to track 'origin/develop'
[ce9eef8afa git-extend@] git flow release start 1.0.0
DeprecationWarning: module.serve() deprecated @ 1.6.0

Summary of actions:
1 You are now on branch 'release/1.0.0' was created, based on 'develop'
You are now on branch 'release/1.0.0'

Follow-up action:
Run the version number tool
Start committing last-minute files in preparing your release
When done, run

    $ git flow release finish 1.0.0

[ce9eef8afa git-extend@] streamcat-changelog --first-release
[ce9eef8afa git-extend@] git commit -m "(Automatic) new changelog"
[release/1.0.0 bdaebf8] (Automatic) new changelog
1 file changed, 1 insertion(+), 1 deletion(-)
[ce9eef8afa git-extend@]
```

Рисунок 3.4: Создание первого релиза

Отправляем данные на GitHub:

```
git push --all
git push --tags
```

Создаем релиз на GitHub:

```
gh release create v1.0.0 -F CHANGELOG.md
```

3.3 Работа с репозиторием git

3.3.1 Разработка новой функциональности

Создаем ветку для новой функциональности:

```
git flow feature start feature_branch
```

Работаем с кодом как обычно, добавляя и изменяя файлы. По окончании разработки объединяем ветку feature_branch с develop:

```
git flow feature finish feature_branch
```

3.3.2 Создание нового релиза

Создаем релиз с версией 1.2.3:

```
git flow release start 1.2.3
```

Обновляем номер версии в файле package.json на 1.2.3.

Создаем обновленный журнал изменений:

```
standard-changelog
```

Процесс работы с changelog показан на рисунке Рисунок 3.5.

[illegible]

Рисунок 3.5: Работа с changelog

Добавляем журнал изменений в индекс:

```
git add CHANGELOG.md
git commit -am 'chore(site): update changelog'
```

Дополнительные операции с changelog показаны на рисунке Рисунок 3.6.

```

$ nano .gitignore
#
# Write a message for tag:
#
# 1.0.0
#
# Lines starting with '#' will be ignored.
#
updated the package.json

```

Рисунок 3.6: Добавление записей в changelog

Завершаем релиз:

```
git flow release finish 1.2.3
```

Отправляем данные на GitHub:

```
git push --all  
git push --tags
```

Создаем релиз на GitHub:

```
gh release create v1.2.3 -F CHANGELOG.md
```


4 Выводы

В ходе выполнения лабораторной работы были получены практические навыки правильной работы с репозиториями git. Освоены следующие технологии и инструменты:

1. **Git-flow** — модель ветвления для организации рабочего процесса с использованием веток master, develop, feature, release и hotfix
2. **Семантическое версионирование** — формальное соглашение о назначении номеров версий в формате МАЖОРНАЯ.МИНОРНАЯ.ПАТЧ
3. **Conventional Commits** — спецификация для написания стандартизированных сообщений коммитов
4. **Инструменты автоматизации:**
 - pnpm для управления пакетами Node.js
 - commitizen для создания правильно отформатированных коммитов
 - standard-changelog для автоматической генерации журнала изменений

Успешно выполнены следующие задачи:

- Установлено и настроено необходимое программное обеспечение (git-flow, Node.js, pnpm, commitizen, standard-changelog)
- Создан тестовый репозиторий git-extended с поддержкой git-flow
- Настроена конфигурация для conventional commits
- Созданы релизы с автоматической генерацией changelog
- Отработаны практические сценарии работы с ветками feature и release

Полученные навыки позволяют организовать профессиональный рабочий процесс разработки с четкой структурой ветвления, автоматическим версионированием и документированием изменений.

Список литературы

- Gitflow Workflow: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>
- Semantic Versioning: <https://semver.org/>
- Conventional Commits: <https://www.conventionalcommits.org/>
- pnpm Documentation: <https://pnpm.io/>
- Commitizen: <https://github.com/commitizen/cz-cli>
- Standard Changelog: <https://github.com/conventional-changelog/conventional-changelog>
- Git-flow: <https://github.com/nvie/gitflow>