

An abstract graphic on the left side of the slide, consisting of a network of thin, light green lines and small circles, resembling a circuit board or a branching diagram, set against a dark green background.

GIT-FLOW

МОДЕЛЬ ВЕТВЛЕНИЯ РАЗРАБОТКИ В GIT

АВТОРЫ

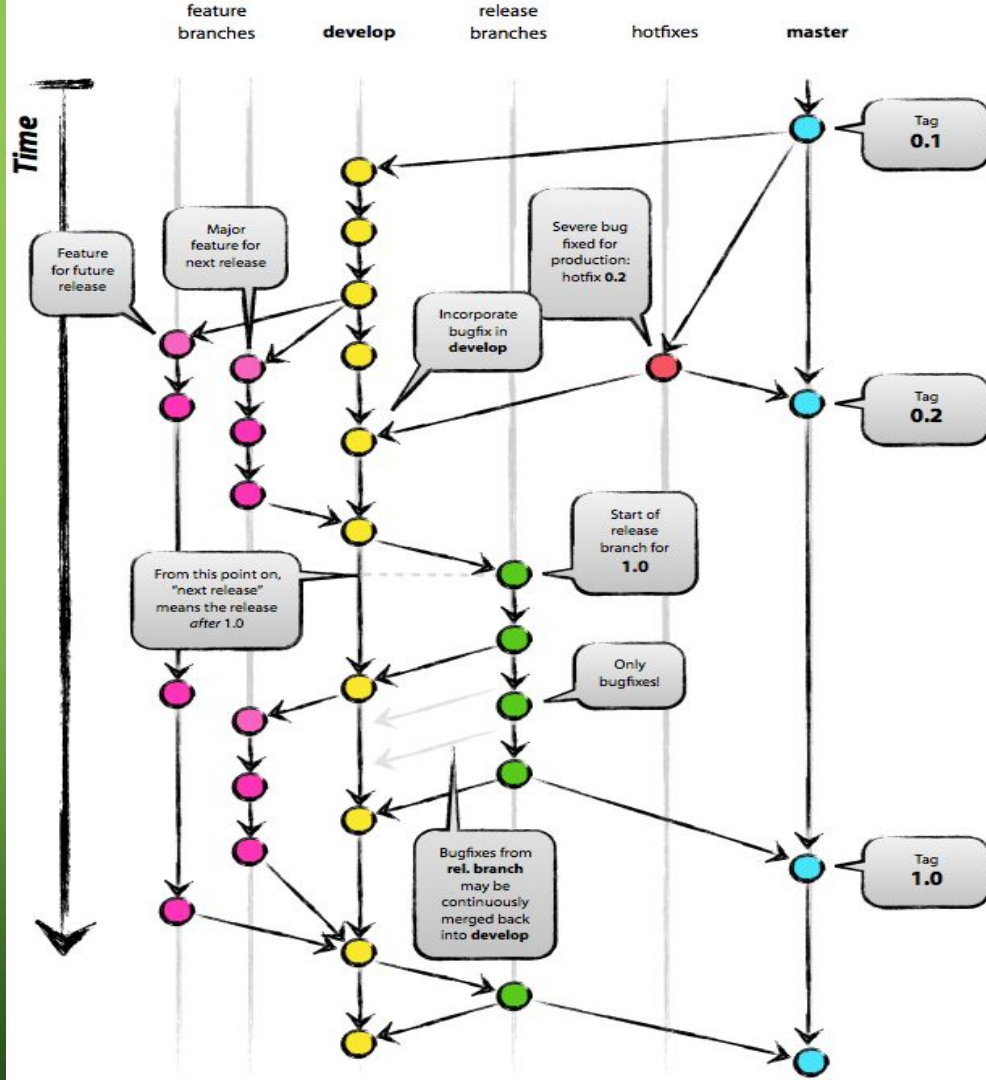
Git: Linus Torvalds

Git-flow: Vincent Driessen

Презентация: Sergey Chudakov

Актуальная версия:

www.ruops.dev



ДЛЯ ЧЕГО: КОМАНДНАЯ РАЗРАБОТКА ПО СТАНДАРТУ, СТАБИЛИЗАЦИЯ, CI/CD

Git Flow является методологией работы с Git. Это значит, она определяет, какие ветки нужно создать и как производить их слияние

git-flow является оберткой для Git. Команда `git flow init` является расширением стандартной команды `git init` и ничего не меняет в вашем репозитории, кроме того, что создает ветки. Есть интеграции с IDE и GUI менеджерами репозитория

- Вместо использования одной ветки **master**, в этой модели используется две ветки для записи истории проекта. В ветке **master** хранится официальная история релиза, а ветка **develop** служит в качестве интеграционной ветки для новых функций. Также, удобно тегировать все коммиты в ветке **master** номером версии

Master

Develop



ИНИЦИАЛИЗАЦИЯ GIT FLOW

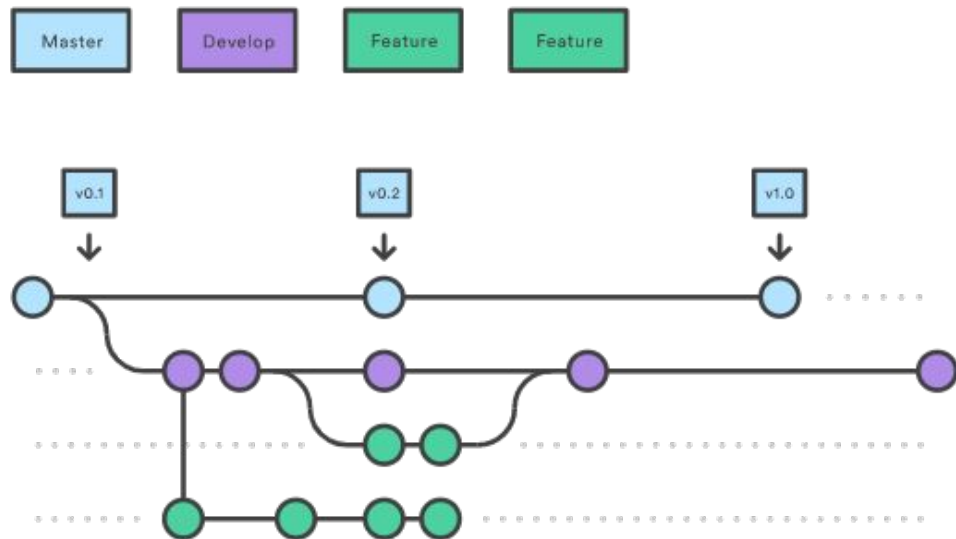
- Первым шагом является создание ветки **develop** от ветки **master**
- В этой ветке будет находиться вся история проекта, в то время как **master** содержит частичную историю
- Остальные разработчики теперь должны клонировать

```
0 ✓ csred@CHUDOPC /d/Sergei/Dev/git/angular-architecture-production-nodejs-typescript $ git flow init
Which branch should be used for bringing forth production releases?
- master
Branch name for production releases: [master]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? [] r
Hooks and filters directory? [D:/Sergei/Dev/git/angular-architecture-production-nodejs-typescript/.git/hooks]
```

ВЕТКИ

- Каждая новая функциональность должна разрабатываться в отдельной ветке, которую нужно отправлять (push) в центральный репозиторий (origin) для создания резервной копии/для совместной работы команды
- Ветки функций создаются не на основе **master**, а на основе **develop**. Когда работа над новой функциональностью завершена, она вливается назад в **develop**
- Новый код не должен отправляться напрямую в **master**



ОБРАТИТЕ
ВНИМАНИЕ, ЧТО
ВЕТКИ ФУНКЦИЙ
ОБЪЕДИНЯЮТСЯ С
ВЕТКОЙ **DEVELOP** И
УДАЛЯЮТСЯ ПОСЛЕ
СЛИЯНИЯ

СОЗДАНИЕ ВЕТКИ ФУНКЦИИ

Для примера создадим ветку с названием **production-build**

- Без использования расширений git-flow:

```
git checkout -b feature/production-build develop
```

- При использовании git-flow:

```
git flow feature start production-build
```

ПУБЛИКАЦИЯ, ПОЛУЧЕНИЕ, ОТСЛЕЖИВАНИЕ

- Публикация фичи для совместной разработки:

`git flow feature publish production-build`

- Получение фичи, опубликованной другим разработчиком:

`git flow feature pull origin production-build`

- Отслеживать фичу в репозитории origin:

`git flow feature track production-build`

ЗАВЕРШЕНИЕ РАБОТЫ С ВЕТКОЙ

- Без использования расширений git-flow:

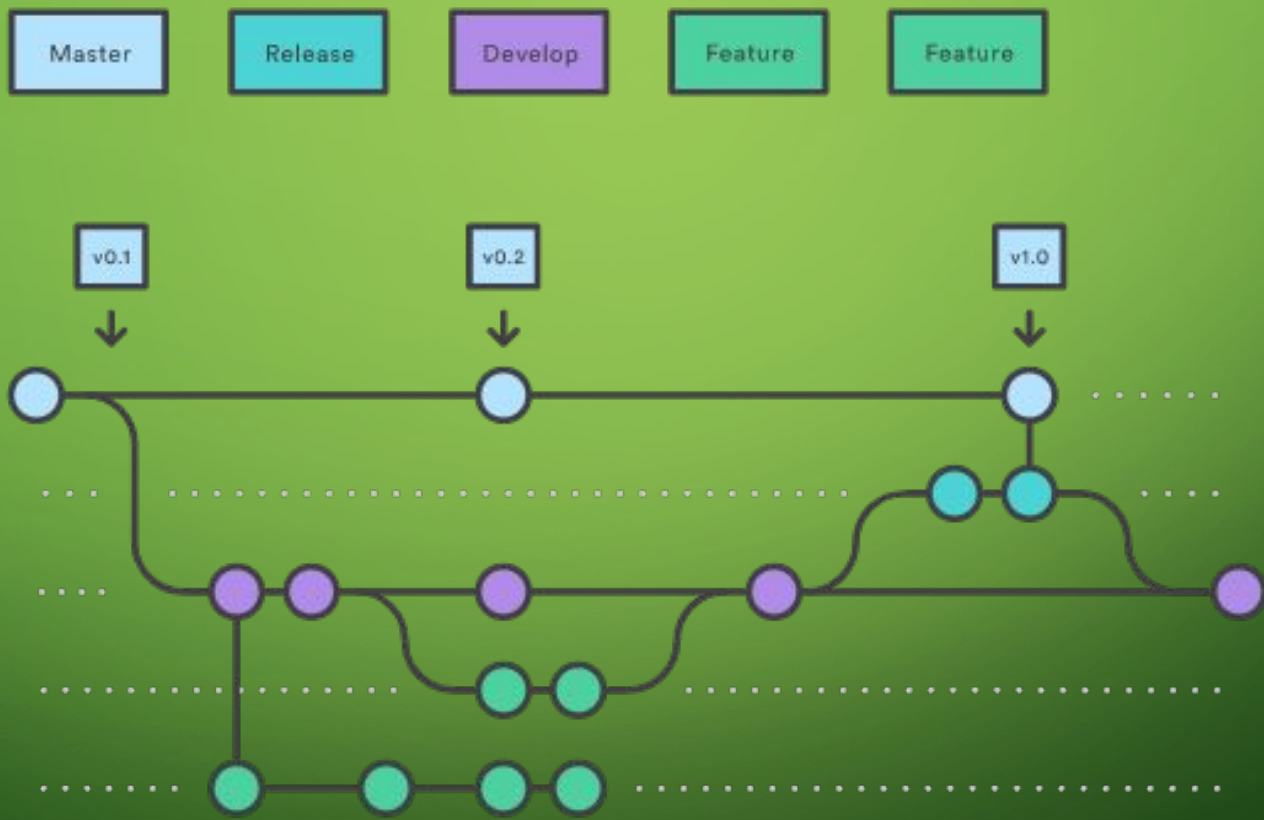
```
git checkout develop
```

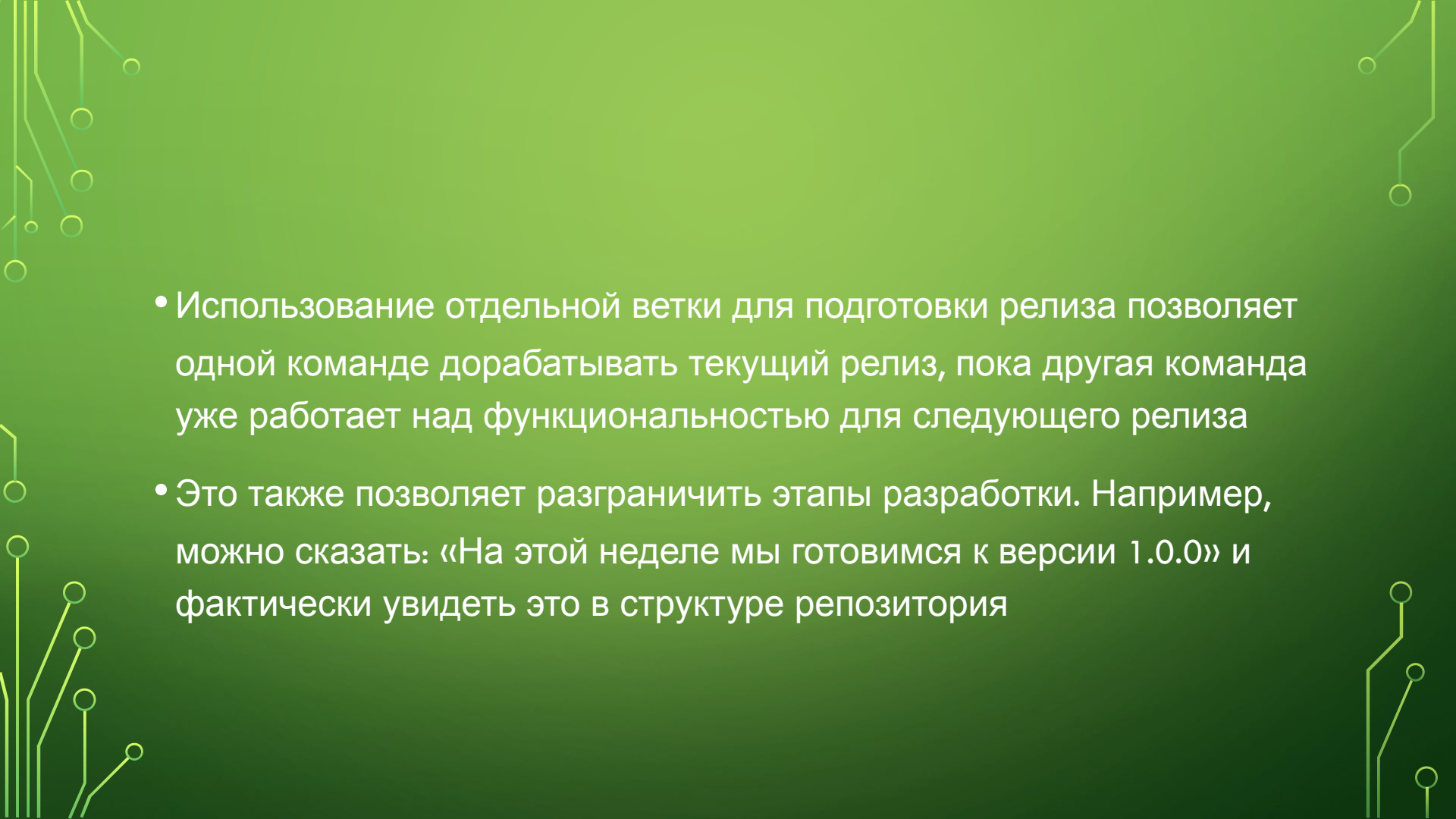
```
git merge --no-ff production-build
```

- При использовании git-flow:

```
git flow feature finish production-build
```

- Когда в ветку **develop** уже слито достаточно нового кода для релиза (или подходит установленная дата предрелиза), от ветки **develop** создается релизная ветка, например, **release/0.3.0**
- Создание данной ветки означает начало следующего цикла релиза, в ходе которой новая функциональность уже не добавляется, а производится только отладка багов, создание документации и решение других задач, связанных с релизом
- Когда все готово, ветка **release** сливается в **master**, и ей присваивается тег с версией (**r0.3.0**). Кроме этого, она должна быть также слита обратно в ветку **develop**, в которой с момента создания ветки релиза могли добавляться изменения с момента создания ветки релиза



- 
- The slide features a dark green background with decorative circuit-like lines in a lighter green shade. These lines, consisting of straight segments and small circles, are positioned in the top-left, top-right, bottom-left, and bottom-right corners, framing the central text area.
- Использование отдельной ветки для подготовки релиза позволяет одной команде дорабатывать текущий релиз, пока другая команда уже работает над функциональностью для следующего релиза
 - Это также позволяет разграничить этапы разработки. Например, можно сказать: «На этой неделе мы готовимся к версии 1.0.0» и фактически увидеть это в структуре репозитория

ВЕТКИ РЕЛИЗОВ ОСНОВАНЫ НА ВЕТКЕ **DEVELOP**

НОВАЯ ВЕТКА **RELEASE** МОЖЕТ БЫТЬ СОЗДАНА С ИСПОЛЬЗОВАНИЕМ СЛЕДУЮЩИХ КОМАНД

- Без использования расширений git-flow:

```
git checkout develop
```

```
git checkout -b release/0.5.0
```

- При использовании git-flow:

```
git flow release start 0.5.0 # ещё одним параметром может быть указан [BASE], сейчас это develop
```

```
git flow release publish 0.5.0
```

При желании вы можете указать [BASE] - коммит в виде его хеша sha-1, чтобы начать релиз с него, коммит должен принадлежать ветке develop

- Когда релиз готов к отправке, он сливается в **master** и **develop**, а ветка релиза удаляется (*может быть сохранена при продуктовой разработке и необходимости поддержки нескольких релизов*). Важно влить **release** обратно в **develop**, поскольку в ветку релиза могут быть добавлены критические обновления и они должны быть доступны в дальнейшем. Если ваша команда делает акцент на проверку кода, этот момент идеален для мердж-реквеста (MR, PR, Merge/Pull Request)
- Релиз помечается тегом равным его имени в ветке **master**. При инициализации может быть задан префикс для тега версии или указан в файле `.git/config` позже. *Префикс тега не добавляется к release ветке*

- Без использования расширений git-flow:

```
git checkout develop
```

```
git merge release/0.7.0
```

```
git checkout master
```

```
git merge release/0.7.0
```

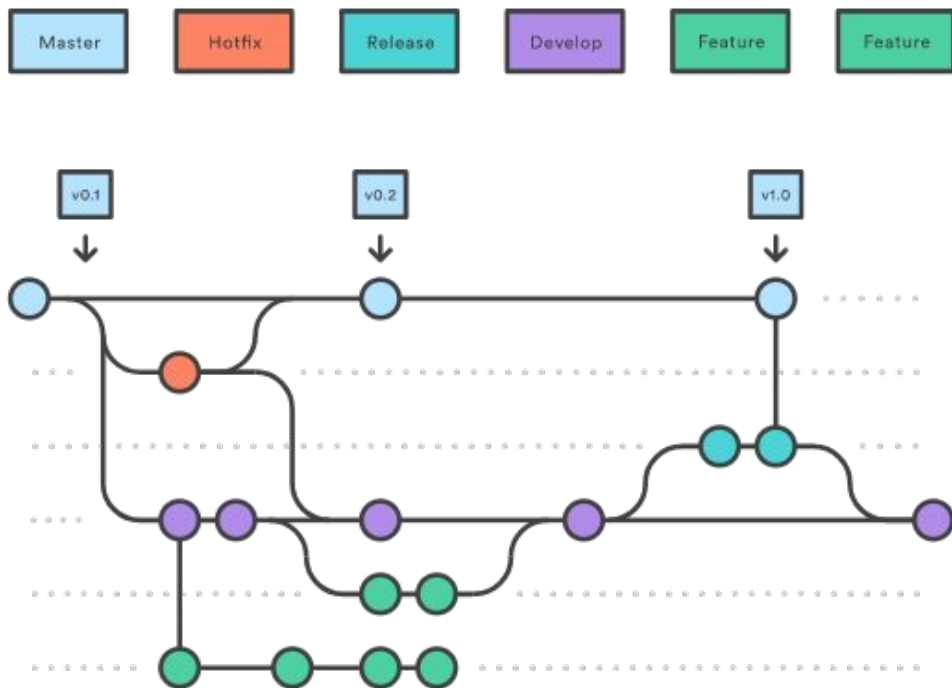
```
git tag r0.7.0
```

- Не забудьте отправить изменения в тегах с помощью команды:

```
git push --tags
```

- Или при использовании git-flow:

```
git flow release finish '0.7.0'
```



- Ветки hotfix используются для быстрого внесения исправлений в рабочую версию кода
- Ветки hotfix очень похожи на ветки release и feature, за исключением того, что они созданы от master, а не от develop

- **hotfix** - это единственная ветка, которая должна быть создана непосредственно от **master**. Как только исправление завершено, ветка **hotfix** должна быть объединена как с **master**, так и с **develop** (или с веткой текущего релиза), а **master** должен быть помечен обновленным номером версии
- Наличие специальной ветки для исправления ошибок позволяет команде решать проблемы, не прерывая остальную часть рабочего процесса и не ожидая следующего цикла подготовки к релизу. Можно говорить о ветках **hotfix** как об особых ветках **release**, которые работают напрямую с **master**

ВЕТКА HOTFIX МОЖЕТ БЫТЬ СОЗДАНА С ПОМОЩЬЮ СЛЕДУЮЩИХ КОМАНД

- Без использования расширений git-flow:

```
git checkout master
```

```
git checkout -b hotfix_branch
```

- Или при использовании git-flow:

```
git flow hotfix start hotfix_branch [BASE]
```

ВЕТКА HOTFIX ОБЪЕДИНЯЕТСЯ КАК С MASTER, ТАК И С DEVELOP

```
git checkout master
```

```
git merge hotfix_branch
```

```
git checkout develop
```

```
git merge hotfix_branch
```

```
git branch -d hotfix_branch
```

или через git-flow:

```
git flow hotfix finish hotfix_branch
```

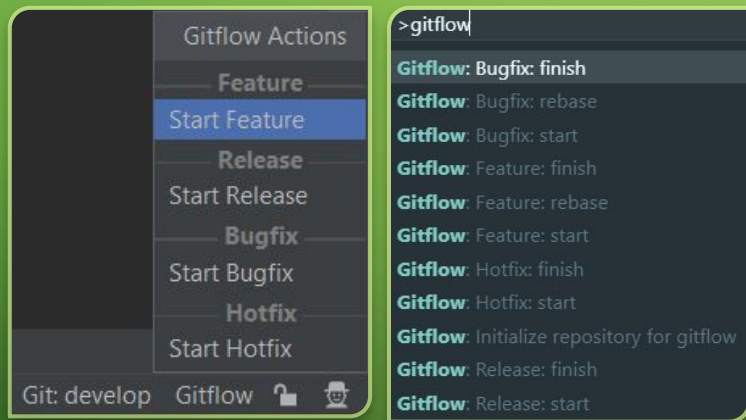
КЛЮЧЕВЫЕ ИДЕИ, КОТОРЫЕ НУЖНО ЗАПОМНИТЬ О GIT FLOW

- Данная модель отлично подходит для организации рабочего процесса на основе релизов
- Git-flow предлагает создание отдельной ветки для исправлений ошибок в продуктовой среде
- Модель может быть дополнена использованием **Project-ID** в названии ветки для интеграции task-трекера: **feature/VK-342-ci**
- Модель Git-flow достаточна, проста и стандартна. Для добавления Staging и Pre-production веток используйте модель GitLab Flow, а для упрощения и личного проекта GitHub Flow

ПОСЛЕДОВАТЕЛЬНОСТЬ РАБОТЫ ПРИ ИСПОЛЬЗОВАНИИ МОДЕЛИ **GIT-FLOW**

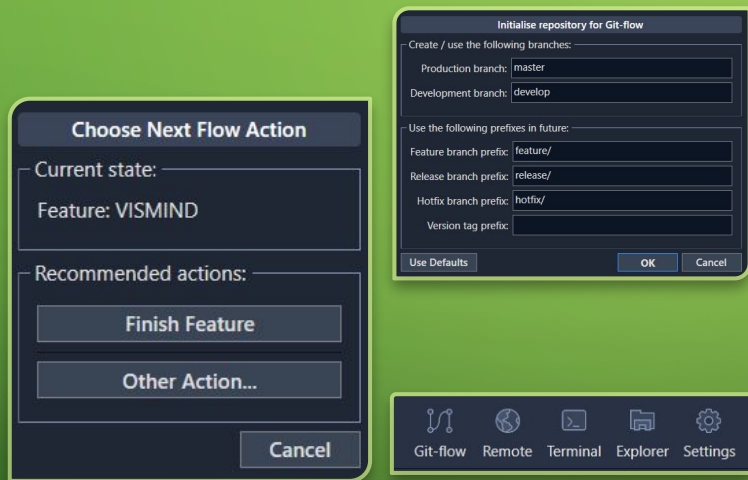
1. Из **master** создается ветка **develop**
2. Из **develop** создаются ветки **feature**
3. Когда разработка новой функциональности завершена – **фичеветка** объединяется с веткой **develop**
4. Из **develop** создается ветка **release**
5. Когда ветка релиза готова, она объединяется с **develop** и **master**
6. Если в **master** обнаружена проблема, из нее создается ветка **hotfix**
7. Как только исправление на ветке **hotfix** завершено, она объединяется с **develop** и **master**

ПЛАГИНЫ ДЛЯ IDE



- IntelliJ IDEA:
<https://plugins.jetbrains.com/plugin/7315-git-flow-integration>
- Visual Studio Code: <https://marketplace.visualstudio.com/items?itemName=vector-of-bool.gitflow>
- VS Code GUI:
<https://github.com/PsykoSoldi3r/vscode-git-flow>

ATLASSIAN SOURCETREE



- Инициализация репозитория
- Старт ветки `feature/release`
- Работа по реализации фичи и исправлений в рамках ветки
- Завершение ветки `feature` перед закрытием таска
- При завершении ветки `release` удаляется ветка и создаётся тег с номером версии в ветке `master`
- Скачать бесплатно: source-treeapp.com
choco install sourcetree

УСТАНОВКА ДЛЯ КОМАНДНОЙ СТРОКИ

- Linux

```
$ sudo apt install git-flow
```

- OS X

```
$ brew install git-flow-avh
```

- Windows:

ВХОДИТ В КОМПЛЕКТ И РАБОТАЕТ ИЗ Git bash И cmd: gitforwindows.org

BONUS

Автодополнение команды git flow с помощью Tab для Bash и Zsh

- <https://github.com/bobthecow/git-flow-completion>

Шпаргалка по git-flow с подсказками на первое время

- https://danielkummer.github.io/git-flow-cheatsheet/index.ru_RU.html

Хуки для автоматического повышения версии и сообщения для тега

- <https://github.com/jaspenbrouwer/git-flow-hooks>

При подготовке презентации использовались материалы:

- <https://bitworks.software/2019-03-12-gitflow-workflow.html>

Презентация подготовлена для доклада в компании БИТ

- bittechno.ru