

Шпаргалки по технологиям

В этой книге собраны инструкции по различным инструментам и технологиям

- Git
 - Git: Основы
 - Git: Работа с ветками
- SSH
- WireShark
- Databases

Git

Git: Основы

Определение

Git - система версионирования текстовых файлов. С помощью неё удобно отслеживать сделанные изменения, переключаться на старую версию файлов проекта, совмещать изменения сделанные разными людьми и просматривать разницу между текстовыми файлами.

Основные понятия

- commit - зафиксированное состояние (и как действие: фиксация изменения)
- commit message - сообщение, поясняющее сделанные изменения
- branch - ответвление состояния из другого состояния. Ветка.
- push - процесс отправки локальных изменений на удалённый сервер
- pull - процесс скачивания изменения с удалённого сервера
- fetch - обновление истории коммитов на локальной машине с сервера (без скачивания самих изменений)
- checkout - процесс переключения на определённое состояние (на коммит/тэг/ветку)
- tag - пометка на коммите (например версия: v1.2.3)

Базовая работа с git

Для работы с git используются следующие файлы:

Тут описать где лежат, что в них нужно вписать, как их минимально настроить

.gitconfig TBD (имя почта, указание где взять .gitcredentials) @ в vscode ломает аутентификацию

.git-credentials TBD (как вписать свой токен/пароль + для нескольких репок)

.gitignore TBD (глобальный и локальный)

Extra:

.gitattributes TBD (можно ещё чонить про LFS)

.githooks TBD (тут можно пару слов всего)

Версионирование

Для продукта чаще всего используют семантическое версионирование vX.Y.Z (например v1.7.42)

Вкратце договорённость такая:

- X - мажорная версия, растёт при потере обратной совместимости API (под API тут понимается в том числе взаимосвязи между модулями одного приложения)
- Y - минорная версия, растёт при добавлении новой функциональности
- Z - патч/багфикс-версия, растёт при починке багов без новой функциональности

“ZeroVer - софт, который имеет нулевую мажорную версию принято считать не готовым для использования, тем не менее, такого софта много. Старайтесь соблюдать правило: если считаете софт не готовым для использования пользователями, то придерживайтесь нулевой мажорной версии.

Тэги

Теги используются для отметки специальных состояний проекта, например релиза или конкретной версии. Если вы используете семантическое версионирование, то ваш тэг будет выглядеть примерно так `v1.2.3`

Читайте далее:

[Git: Работа с ветками](#)

Git: Работа с ветками

master / main / default / trunk - стандартные названия основных ветвей. Стабильные ветки, из которых обычно выпускают релизы.

dev / development - основная ветка разработки, туда вносятся текущие изменения, не считается стабильной. Если изменений вносится немного (поправить пару строчек / заменить конфиг / обновить библиотеку с сохранением совместимости) и эти изменения безопасны допустимо добавление коммитов непосредственно в dev-ветвь. Во всех остальных случаях стоит использовать feature-branch.

feature-branch / ticket-branch - ветка, унаследованная от develop в которую вносятся точечные изменения, например добавление фичи, исправление бага. Тикет бранчи именуются с номером соответствующего тикета/задачи. Так можно по названию ветки найти задачу и понять цель изменений. И наоборот: по тикету найти ветку. В этой ветке может существовать частично нерабочее состояние репозитория.

Полный классический Git flow

Screenshot 2024-10-05 211241.png

Более простая схема для роли developer

Screenshot 2024-10-05 211246.png

Процесс работы в feature-branch

Screenshot 2024-10-05 211252.png

Безопасное удаление ветвей

Если неслитые ветки удалять через интерфейс гитлаба/гитхаба, то все изменения в этих ветках навсегда будут утеряны. Чтобы такого не происходило, можно перед удалением

ветки навесить на неё тег: (например тег archive/назв.ветки)

```
git clone <url репозитория>  (если нет локальной копии)
cd <репозиторий>

git checkout <имя ветки для закрытия>
git tag archive/<имя ветки для закрытия> <имя ветки для закрытия>
git checkout <любая другая ветка>
git branch -D <имя ветки для закрытия>
git branch -d -r origin/<имя ветки для закрытия>
git push --tags
git push -d origin <имя ветки для закрытия>
```

Как работает MR

На самом деле при merge/MR происходит чуть более сложная логика чем просто наложение diff разных коммитов последовательно друг на друга. [Подробнее можно почитать здесь](#).

В настройках репозитория можно выбрать метод для merge:

Screenshot 2024-10-05 211819.png

SSH

Что такое SSH

Как сгенерировать ключи, как пользоваться

Как их подгрузить в vscode

SCP

From local to remote

```
scp localfile.zip username@server.domain.com:/home/username
```

WireShark

Wireshark-cheatsheet-1.png

Wireshark-cheatsheet-2.png

Databases

databases.jpg