Git

- Git: Основы
- Git: Работа с ветками

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 удобной работы с git используются следующие файлы:

Для того чтобы вы могли пулить и пушить не вводя пароль каждый раз, можно создать два файла, которые будут хранить эти данные. Однако, если вы работайте на многопользовательском сервере, то будьте аккуратнее чтобы к вашему юзеру не было доступа у других.

Однако пароль из-за требований безопасности не рекомендуется использовать. Вместо пароля используются более гибкие временные пароли - токены. Сгенерируйте токены на странице в GitLab (`https://gitlab.example.com/-/user_settings/ssh_keys`) или GitHub (https://github.com/settings/tokens) с нужными правами доступа. (Для новичков подойдёт классический токен - он проще настраивается)

Итак, создаём файлы в пользовательской директории `/home/you_user` для linux или `C:\Users\you_user` для windows:

- `.gitconfig` - файл, информация из которого используется в коммитах (имя из этого файла будет отображаться в коммитах).
  Пример заполнения:

  ```
  [credential]
      helper = store # указание что токен нужно брать из .git-credentials
  [user]
      name = Firstname Lastname
      email = your-email@gmail.com
  ```

- `.git-credentials` - файл с реквизитами доступа к git-серверам (файл с токенами). В этом файле могут быть credentials для разных удалённых серверов одновременно.
  Пример с одним сервером:

  ```
  https://username:ghp_some0very0very0very0very0long0token@github.com
  ```

  формат заполенения: каждая строчка состоит из: `https://username:token@domain.name`.
  Важное примечание: если в токене есть символ `@`, то vscode ломает аутентификацию. Лучше перегенерируйте токен.

Опционально можно создать дополнительные файлы:

- `.gitignore` - в папке юзера лежит глобальный gitignore, то есть применяющийся ко всем проектам git. (В папке проекта - локальный для проекта). Советую его заполнить чтобы случайно не закоммитить пароли или виртуальные окружения в любых проектах:

  ```
  .idea
  .idea/
  .vscode
  .ipynb_checkpoints
  __pycache__
  venv/
  .venv/

  clients/
  statistic/
  _statistic/
  userdata/
  logs/

  *.env*
  *.conf*
  ```

```
*.logs*
*.log*

*privatekey*
test.txt
text.txt
ignore/
```

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 используйте семантическое версионирование, то ваш тэг будет выглдяеть 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 ветвях 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