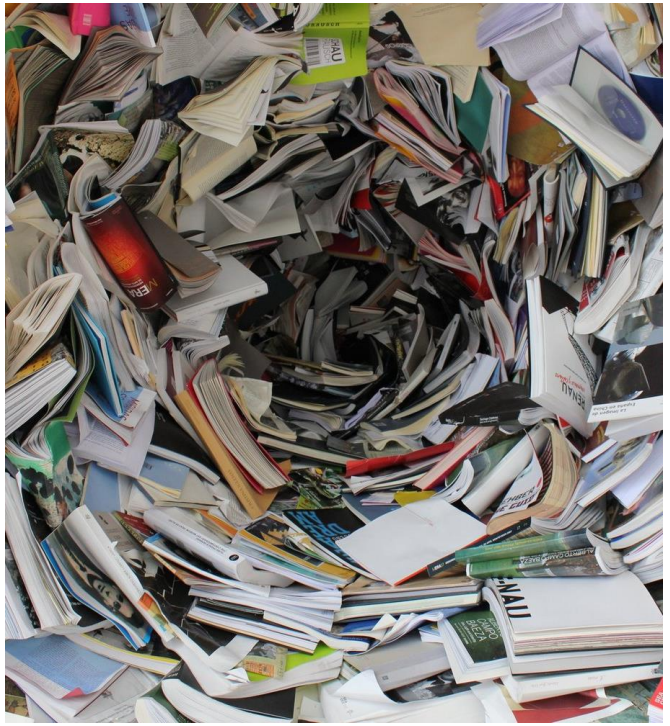
Основы работы с Git и GitHub

Проблема контроля изменений

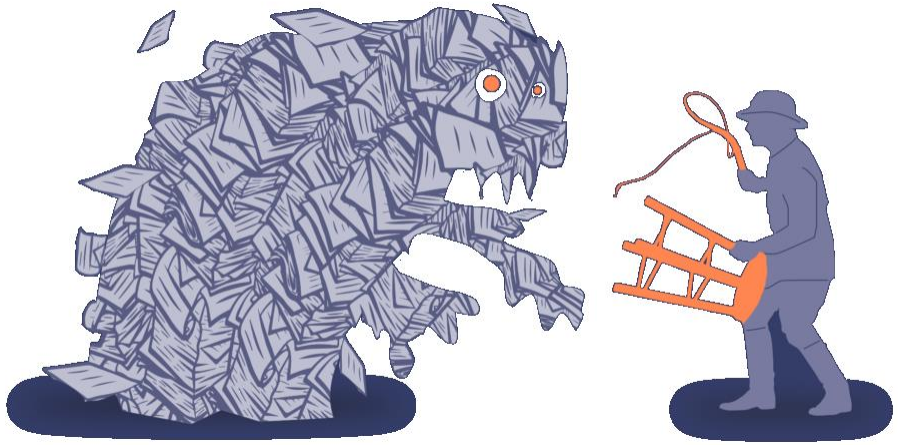
- Жизнь важного документа:
  - Вы создаёте важный документ.
  - Через какое-то время изменяете его. Потом ещё раз и ещё раз. Потом ваш коллега изменяет ещё что-то.
  - И потом ваш начальник находит там ошибку и вопрошает: "Кто это сделал?!". Но никто не признаётся, в результате вы остаётесь без премии (просто, потому что это же вы создали документ).
  - Ещё позже, из отдела поддержки вам приходи письмо: "Полгода назад вы прислали нам важный документ. Важный клиент запросил его вчера, но мы его потеряли. Перешлите ещё раз!". Но за полгода важный документ был почти полностью переписан, восстановить его невозможно, в результате вы остаётесь без премии (отдел поддержки не причём, потому что там работает Анна Петровна, которую лучше не злить).
  - Чтобы ускорить процесс, вы с коллегой решаете работать вместе над восстановлением старого документа для важного клиента. Вы и ваш коллега трудитесь несколько дней независимо. И когда наконец встречаетесь, оказывается, что ваш и документ коллеги совершенно разные. Вы тратите ещё несколько дней чтобы объединить их, проваливаете дедлайн, в результате вы остаётесь без премии (ваш коллега неприём, потому что это брат начальника, и вообще он вам помогла и за это получит вашу премию).



- Всё это совершенно типичные проблемы, которые существуют столько сколько существуют сами документы.

Системы контроля версий

- Простейший подход, это копирование документа каждый раз перед внесением изменений, с нумерацией и сохранением всех копий.
- Это работает! И имеет смысл, когда документ не большой и меняется редко. Но если изменений много, контролировать их становится тяжело и рутинно.
- Потому не удивительно что этот процесс был автоматизирован. Особенно широко с внедрением электронного документа оборота.
- В частности нас, как программистов, интересует контроль версий исходных текстов. Которым присущи все те же проблемы что и другим документам.
- Существует много систем контроля версий для программистов. Сегодня мы познакомимся с одной из самых популярных и простых, с Git.



Работа с Git

- В Git нет никакой магии, по большому счёту он делает всё то же копирование документов (файлов), но делает это автоматически и на уровне строк текста. Плюс, имеет дополнительные инструменты для работы с копиями файлов.

- Для начала попробуем несколько базовых команд:
  - **git init** - инициализирует систему контроля версий. Иначе говоря, команда создаёт репозиторий (место, где будут хранится все копии наших файлов и конфигурация самого Git).
  - **git add .** - добавление фалов в репозиторий. После добавления Git отслеживать измерения в файлах.
  - **git commit -am "<комментарий>"** - создаёт коммит ("коммитит"), т.е. сохраняет изменения сделанные в отслеживаемых файлах, грубо говоря сохраняет новую (следующую) копию файлов.
  - **git show** - показывает изменения сохранённые в предыдущем коммите.

- Больше команд можно найти здесь: Основы Git

Работа с GitHub

- В GitHub тоже нету никакой магии, фактически это просто хостинг для файлов (такой как DropBox или GoogleDrive), но хостятся на нём файлы исходного кода.
- GitHub работает совместно с Git. Потому для того чтобы разместить на GitHub файлы нужно также создать там репозиторий и "подключить" его к нашему Git репозиторию. Что сейчас мы и попробуем сделать.
- Создание аккаунта GitHub такое же, как и на других веб порталах, потому не будем останавливаться подробно на этом.
- Для создания репозитория выберете **New repository**. Введите имя репозитория и нажмите **Create repository**
- Добавить аккаунт в Git можно командами:
  - **git config --global user.name "<имя>"**
  - **git config --global user.email "<емейл GitHub>"**
  - **git config --global user.password "<пароль GitHub>"**
  - **git config --global credential.helper store**

  (!) Делать именно так, в общем случае не безопасно (лучше использовать SSH ключи), но зато просто.
- Команды которые нам пригодятся:
  - **git remote add origin <ссылка>** - подключает GitHub репозиторий к локальному Git репозиторию
  - **git push --set-upstream origin master** - отправляет ("пушет") изменения файлов на GitHub
  - **git clone <ссылка>** - клонирует репозиторий (свой или чужой)

Работ с ветками

- Работая в команде или просто над каким-то большим изменением, удобно иметь отдельную копию репозитория и возможность эти копии объединять ("мержить"). Такие копии называются ветками ("бранчами").

- Для работы с ветками в Git есть несколько основных команд:
  - **git branch <имя_ветки>** - создаёт новую ветку из текущей.
  - **git checkout <имя_ветки>** - переключение между ветками. После выполнения этой команды, все последующие commit'ы будут сделаны в выбранную ветку.
  - **git merge <имя_ветки>** - локальное объединение веток. Однако сегодня мы рассмотрим только объедение через pull request, так как оно нам скоро понадобятся.

- В каждой из веток можно делать commit'ы и push'ы, в этом плане они не отличаются.

- Создать pull request можно чрез GitHub UI: **Pull requests** -> **New pull request**

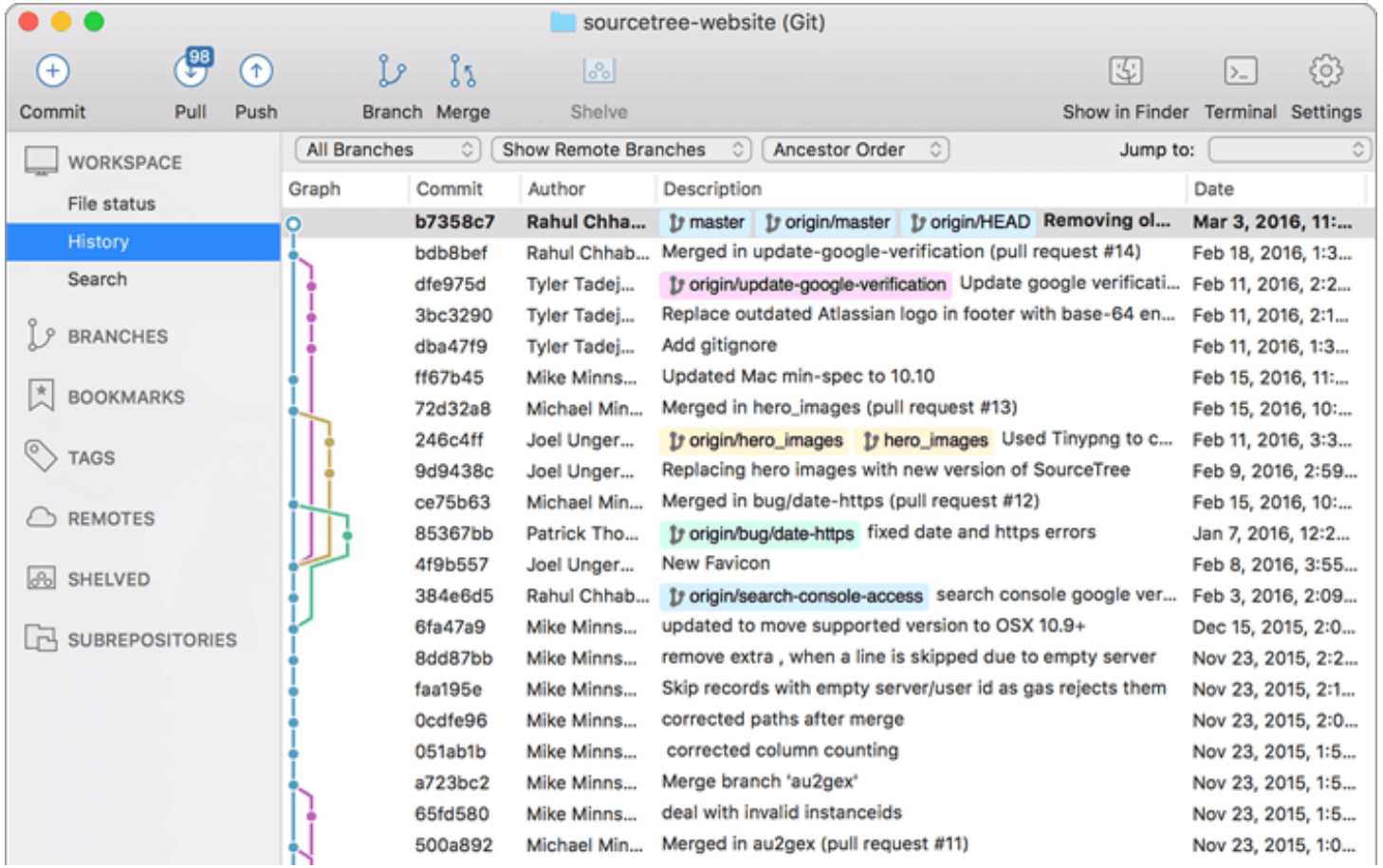
- Когда пул pull request создан, другие пользователи могут его проверить ("ревъювить") и подтвердить ("заапрувить") или не подтвердить ("режектнуть"). А также могут оставлять комментарии

Origin Master

```
>git merge
```

SourceTree (опционально)

- Для тех кто не любит консоль (как я) есть GUI утилита SourceTree для работы с Git.



Спасибо за внимание!
Вопросы?