Система контроля версий Git

Что такое система контроля версий

- Хранит историю изменений
- Позволяет разботать надо документом совместно
- Позволяет создавать параллельные версии (ветки)
- Предоставляет инструменты для слияния нескольких версий

Популярные системы версий

- Git
- Subversion (централизованная)
- Mercurial (альтернатива Git)

Что такое Git

- Создана Линусом Торвальдсом в 2005 году для разработки ядра Linux
- Распределённая архитектура: каждый разработчик хранит полную копию репозитория (и его историю)
- Лёгкость создания веток
- Все операции выполняются локально
- Использование хеширования для хранения истории (невозможно переписать без следа)
- Изменения разбиваются на стадии, позволяет выбирать, что и когда добавлять в историю

Преимущества Git

- Гибкость: подходит и для маленьких, и для больших проектов
- Повышенная сохранность проекта благодаря децентрализации
- Мощные возможности для разрешения конфликтов редактирования
- Множество онлайн-репозиториев, интегрированных с дополнительными инструментами (код-ревью, автоматическая проверка кода, документация, трекер ошибок и т. д.)
- Большое сообщество

Недостатки Git

- Сложная система разрешения конфликтов
- Неудобно работать с нетекстовыми файлами (а также и с текстовыми, содержащими много текста в строках)
- Невозможно заблокировать файл от изменений
- Нет автоматического добавления изменений в историю

Онлайн-репозитории Git

- GitHub (https://github.com/)
- GitLab (https://gitlab.com/)
- Bitbucket (https://bitbucket.org/)
- SourceForge (https://sourceforge.net/)
- GitVerse (https://gitverse.ru/)

Введение в Git

Определения

- Проект располагается в **репозитории**
- Репозиторий можно **клонировать** в другую папку или на другой компьютер, один из клонов объявляется основным (origin)
- При редактировании файла его статус изменяется на **модифицированный**
- Модифицированные файлы можно **проиндексировать** (stage) для дальнейшего добавления в историю
- Проиндексированные файлы добавляются в историю посредством **фиксирования** или **коммита** (commit)
- Параллельные версии кода создаются путём **ветвления** (branching)
- Различные ветки можно **сливать** (merge) друг с другом
- Изменения из основного репозитория добавляются в локальный клон командой **Pull**
- Локальные изменения добавляются в основной репозиторий командой **Push**

Установка Git

- https://git-scm.com
- GitHub Desktop https://github.com/apps/desktop
- VSCode по умолчанию поддерживает работу с git-репозиториями

Начало работы с Git

- Проверка работы

  ```
  > git --version
  ```

- Первоначальная настройка

```
> git config --global user.name "Artem K"
> git config --global user.email "test@mail.com"
```

- Инициализация репозитория в текущей папке

  ```
  > git init
  ```

Добавление файлов в репозиторий

- Если создать файл в папке, оно не будет добавлен в репозиторий автоматически

- Проверка статуса репозитория

  ```
  > git status
  ```

- Индексирование нового файла

  ```
  > git add <filename>
  ```

- `git add --all` индексировать все новые файлы

- Добавление проиндексированных файлов в историю:

  ```
  > git commit -m "<message text>"
  ```

- `git commit -a` объединяет индексирование новых файлов и коммит

Удаление файлов из репозитория

- Удаление файла из индекса и с диска:

  ```
  > git remove <filename>
  ```

- `git remove --cached` удаляет файл только из индекса

- `git remove` не удаляет историю файла из репозитория!

Работа с ветками

- Пример:

    - Создали ветку для разработки новой функциональности или улучшения старой
    - Где-то в существующем коде обнаружили ошибку
    - Создали новую ветку из основной, в ней поправили ошибку
    - Слили ветку с исправленной ошибкой и основную
    - Закончили работу над новой функциональностью и слили её с основной (с предупреждением, что в основную ветку внесены изменения после разделения)

- Ветки легковесны и позволяют работать с разными версиями кода в одной и той же папке

Создание веток

```
> git branch <branch-name>
```

```
> git branch
  <branch-name>
* main
```

- Активирование ветки:

```
> git checkout <branch-name>
```

- `git checkout -b` создаст ветку, если она ещё не существует

Слияние веток

```
> git checkout main
Switched to branch 'main'
> git merge <branch-name>
Updating 09f4acd..dfa79db
Fast-forward
 main.jl | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
> git branch -d <branch-name>
```

Разрешение конфликтов

```
> git merge <branch-name>
Auto-merging main.jl
CONFLICT (content): Merge conflict in main.jl
Automatic merge failed; fix conflicts and then commit the
result.
```

```
> cat main.jl
<<<<<<< HEAD
using Plots
=======
using Makie
>>>>>>> new-feature-branch
```

- Редактируем `main.jl` и добавляем его в индекс:

```
> git add main.jl
```

Использование GitHub в качестве основного репозитория

- Как создать новый репозиторий:
  - Регистрируемся на https://www.github.com/
  - Создаём репозиторий в веб-интерфейсе
  - Запускаем локально команду

```
> git clone https://github.com/korzhimanov/Lecture2025.jl.git
```

Как добавить существующий репозиторий на GitHub:

- Создайте пустой репозиторий (без `README.md` и `.gitignore`, если они уже существуют в вашем репозитории) с нужным именем на GitHub

- Добавьте этот репозиторий, назвав его `origin`:

```
> git remote add origin https://github.com/korzhimanov/test.git
```

- Загрузите свой код, установив репозиторий на GitHub в качестве основного

```
> git push --set-upstream origin main
```

- Если выдаёт ошибку `fatal: refusing to merge unrelated histories`, сначала загрузите изменения с GitHub, разрешив объединение историй из двух несвязанных репозиториев:

```
> git pull origin main --allow-unrelated-histories
```

- Или принудительно перезапишите историю репозитория на GitHUb:

```
> git push --set-upstream origin main --force
```

Обновление локального репозитория из основного

```
> git pull origin
```

```
> git fetch origin
> git log origin/main # посмотреть информацию о новых коммитах в origin
> git diff origin/main # посмотреть разницу в файлах в локальном main и origin
> git merge origin/main
```

Добавление локальных изменений в основной репозиторий

```
> git pull origin # добавить новые изменения из основного репозитория, разрешить конфликты
> git push origin # сработает только если в локальном репозитории учтены все коммиты из основного репозитория
```

GitHub Pull Request

- Это механизм взаимодействия с другими пользователями репозитория и проверки ими вносимых в него изменений
- Создаём новую ветку, вносим изменения в ней
- Коммитим изменения и отправляем (`push`) их на GitHub
- Открываем **Pull Request** (запрос на слияние)

Внесение изменений в чужие репозитории

- Делаем **fork** (ответвление, но не путать с `branch` !) репозитория
- Клонируем основной репозиторий

  ```
  > git clone https://github.com/JuliaLang/julia.git
  ```

- Мы не можем напрямую добавлять наши изменения в основной репозиторий, поэтому переименовываем его:

  ```
  > git remote rename origin upstream
  ```

- Объявляем основным репозиторием наш форк:

  ```
  > git remote add origin https://github.com/korzhimanov/julia.git
  ```

- Редактируем локально и отправляем всё в наш форк:

  ```
  > git push origin
  ```

- Идём на GitHub и делаем Pull Request в основной репозиторий

Удобные настройки Git

`.gitignore`

- Файл `.gitignore` позволяет настроить, что не надо коммитить в репозиторий

  ```
  # ignore ALL .log files
  *.log

  # ignore ALL files in ANY directory named temp
  temp/
  ```

- https://github.com/github/gitignore

Настройка SSH

- SSH (secure shell network) это популярный протокол защищённого доступа к удалённой машине

- Использует пару ключей: публичный и приватный

- Публичный ключ лежит на удалённой машине и доступен для просмотра всем

- Приватный ключ лежит локально и должен быть недоступен для просмотра извне

- Зная оба ключа, можно получить доступ к удалённой машине

- Генерация приватного ключа:

  ```
  > ssh-keygen -t rsa -b 4096 -C "test@mail.com"
  ```

- `id-rsa` это приватный ключ
- `id-rsa.pub` это публичный ключ

- Содержимое `id-rsa.pub` можно добавить на GitHub

- Чтобы использовать SSH соединение в Git:

```
> git remote add origin git@github.com:korzhimanov/test.git
```

- Или, если уже есть HTTPS соединение, его можно заменить на SSH:

```
> git remote set-url origin git@github.com:korzhimanov/test.git
```

Возврат к старой версии в Git

- Находим старый коммит

```
> git log --oneline
52418f7 (HEAD -> master) Just a regular update, definitely no
accidents here...
9a9add8 (origin/master) Added .gitignore
81912ba Corrected spelling error
```

- Просмотр старого коммита:
  - `git show 81912ba` покажет изменения, внесённые в результате этого коммита
  - `git diff 81912ba` покажет разницу между этим коммитом и текущей версией
  - `git diff 81912ba 9a9add8` покажет разницу между двумя коммитами
  - `git diff 81912ba 9a9add8 -- <filename>` покажет разницу в указанном файле

- Отмена коммита:

```
> git revert 81912ba
```

- Это создаст новый коммит, в котором будет удалено всё, что было внесено коммитом `81912ba` (возможно, потребуется разрешение конфликтов)

- `git revert 81912ba --no-edit` создаст коммит с автоматическим сообщением

- `git revert 81912ba --no-commit` отменит коммит, но без создания нового коммита

- `git revert HEAD` отменяет последний коммит

- `git revert 81912ba..52418f7` удалит все коммиты с `81912ba` по `52418f7`
- `git restore .` отменит все непроиндексированные изменения
- `git checkout -- .` отменит все непроиндексированные изменения в уже существоваших файлах (но не затронет новые)
- `git restore --staged .` отменит все незакоммиченные изменения
- `git reset .` отменит всю индексацию (но не откатит изменения)
- `git reset --hard` отменит все изменения в существовавших файлах (но не затронет новые)
- `git clean -fd` удалит все непроиндексированные файлы и папки
- Вместо `.` можно использовать имя файла или папки, тогда отмена будет сделана только в этом файле или папке
- `git reset 81912ba` откатит историю к коммиту `81912ba` (но остальные коммиты сохранятся и к ним можно будет вернуться, если требуется)

Литература

- https://git-scm.com/doc
- https://docs.github.com
- https://gitverse.ru/docs/

Дома

- Создать git-репозиторий в папке с файлом из предыдущей домашней работы
- Зарегистрироваться на GitHub https://github.com
- Создать репозиторий на GitHub и объединить его с локальным репозиторием из 1-го пункта
- Сделать форк репозитория https://github.com/korzhimanov/Lectures2025.jl
- Создать его локальную копию
- Создать в src подпапку со своей фамилией
- Добавить файл с домашним заданием в эту подпапку
- Загрузить обновлённый репозиторий на GitHub и создать Pull Request на его объединение с основным репозиторием