Системы контроля версий

Git и GitHub

Основы систем контроля версий

https://git-scm.com/downloads — официальный сайт Git

https://github.com — официальный сайт GitHub

Книга "Pro Git" (на русском)

GitHub Skills (интерактивное обучение)

Генератор .gitignore

Проблема "10 финальных версий"

Знакомая ситуация?

```
курсач_финальная_версия.doc
курсач_финальная_версия_2.doc
курсач_финальная_версия_3_исправленная.doc
курсач_исправленная_после_проверки_финал.doc
```

Проблема: Невозможно отследить историю, вернуться к состоянию на прошлой неделе, работать в команде.

Что такое Git?

Git — это система контроля версий (Version Control System).

Разработана Линусом Торвальдсом (создателем Linux) в 2005 году.

Как это работает: Git сохраняет не просто файлы, а *слепки* (snapshots) состояния проекта в разные моменты времени.

Позволяет:

- Откатываться к любой версии.
- Видеть, кто и когда изменил конкретную строку.
- Создавать ответвления (ветки) для экспериментов.

Зачем нужна система контроля версий (CVS/VCS)?

- **История изменений:** Полный лог того, что, когда и кем было изменено.
- **Возможность отката:** Случайно сломали код? Можно вернуться к рабочей версии одной командой.
- **Резервное копирование:** Код хранится не только на вашем компьютере.
- **Совместная работа:** Несколько разработчиков могут работать над одним проектом, не затирая код друг друга.
- **Эксперименты:** Можно создать отдельную ветку, попробовать новую идею, и, если не взлетит — просто удалить её.

Альтернативы Git

- **SVN (Subversion):** Централизованная система. Был популярен до Git. Если сервер упал — всё, история потеряна.
- **Mercurial:** Распределенная система, похожая на Git, но менее популярная. Считалась проще в освоении.
- **TFS (Azure DevOps):** Решение от Microsoft для больших корпоративных проектов.

Почему Git стал стандартом? Скорость, мощная система ветвления, распределенность (у каждого полная копия истории).

GitHub и удаленные репозитории

Что такое GitHub?

- **GitHub** — это веб-сервис для хостинга `Git`-репозиториев и совместной разработки.
- Это "облако" для вашего кода и социальная сеть для программистов.
- Добавляет к возможностям Git:
    - Удобный веб-интерфейс.
    - Систему `Issues` (задачи/баги).
    - `Pull Requests` (механизм ревью кода).
    - `GitHub Actions` (автоматизация, CI/CD).

Альтернативы GitHub

- **GitLab:** Самый популярный конкурент. Бесплатный self-hosted вариант (можно поставить на свой сервер). Очень развитый CI/CD.
- **Bitbucket:** От компании Atlassian. Хорошо интегрируется с их продуктами (Jira, Trello).

Вывод: GitHub — стандарт индустрии для `open-source` проектов, но знать альтернативы полезно.

Git + GitHub

Локальный репозиторий (у вас на ПК): Здесь вы работаете, делаете коммиты (фиксации). Git управляет версиями.

Удаленный репозиторий (на GitHub): Здесь хранится "золотая копия" проекта. С ним синхронизируются все участники.

`git push` — отправить свои коммиты на GitHub.

`git pull` — забрать чужие коммиты с GitHub.

Работа с Git в VS Code и консоли

Git в VSCode vs Консоль

VS Code имеет встроенную поддержку `Git` (вкладка `Source Control`).

Удобно для быстрых действий: посмотреть изменения, сделать коммит, запушить.

Минус: Скрывает детали.

Консоль (Терминал):

Дает **глубокое понимание** процессов.

Едина для всех платформ и сред.

Необходима для сложных операций (ребейз, интерактивный `add`).

Наш подход: Сначала учим команды в консоли, потом пользуемся `GUI` для ускорения.

Настройка окружения (SSH-ключи и конфиг пользователя)

1. **Конфиг пользователя (делается 1 раз):**

```
git config --global user.name "Ваше Имя"
git config --global user.email "ваш_email@example.com"
```

2. SSH-ключи (для безопасного соединения с GitHub):

- Проверяем, есть ли ключ: `ls -al ~/.ssh`
- Генерируем новый: `ssh-keygen -t ed25519 -C "ваш_email@example.com"`
- Копируем ключ: `cat ~/.ssh/id_ed25519.pub`
- Вставляем в настройках GitHub: `Settings -> SSH and GPG keys -> New SSH key`.

Базовые команды Git (Шпаргалка)

Инициализация и проверка состояния

- `git init` — создает новый пустой репозиторий в текущей папке (появляется скрытая папка .git).

```
cd my-project
git init
```

- `git status` — самая часто используемая команда. Показывает, какие файлы изменены, добавлены или не отслеживаются.

```
git status
# On branch master
# Untracked files: (файлы, которые Git пока не видит)
```

Добавление и фиксация изменений (add & commit)

`git add` — добавляет изменения в "индекс" (staging area). Говорит Git-у: "Следи за этим файлом и подготовь его к коммиту".

```
# Добавить конкретный файл
git add README.md
# Добавить все изменения в текущей папке
git add -A   # или git add .
```

`git commit -m "сообщение"` — фиксирует изменения. Создает слепок (точку восстановления).

```
git commit -m "Добавлен главный компонент приложения"
```

Привязка удаленного репозитория (remote)

- Предполагаем, что на GitHub уже создан пустой репозиторий.
- `git remote add origin <URL>` — связывает локальный репозиторий с удаленным.
  - `origin` — это просто имя (псевдоним) для удаленного сервера (традиция).
  - `<URL>` — адрес вашего репозитория (SSH или HTTPS).

```
git remote add origin git@github.com:username/repo-name.git
```

  - Проверить привязку:

```
git remote -v
```

Отправка и получение изменений (push & pull)

Первая отправка (push):

```
git push -u origin master
# (или git push -u origin main, если ветка называется main)
```

- `-u (--set-upstream)` связывает локальную ветку с удаленной, чтобы в следующий раз писать просто git push.

Последующие отправки:

```
git push
```

Получение изменений (перед началом работы):

```
git pull
```

(Забирает изменения с удаленного репозитория и сливает с вашей текущей веткой)

Культура работы с Git

Правила хорошего тона: Маленькие и частые коммиты

Правило: Каждый логически завершенный кусочек работы — отдельный коммит.

Плохо: Один коммит "Добавил весь проект" (100 файлов, 10000 строк).

Хорошо:

- `git commit -m "Добавлена верстка шапки сайта"`
- `git commit -m "Реализована логика авторизации"`
- `git commit -m "Исправлен баг с отправкой формы"`

Преимущества: Легче находить баги (`git bisect`), понятнее история, проще откатить конкретное изменение.

Игнорируем лишнее (.gitignore)

- В проекте есть файлы, которые не нужно хранить в репозитории:
  - Временные файлы (`.DS_Store`, `thumbs.db`).
  - Папка с зависимостями (`node_modules`/).
  - Файлы с секретами (`.env` с паролями).
  - Результаты сборки (`dist/, build`/).
- **Решение:** Файл `.gitignore` в корне проекта.

```
# Пример .gitignore для React проекта

node_modules/
dist/
.env
.DS_Store
```

Лабораторная работа № 2_2
"Основы Git и работа с GitHub"

Система контроля версий Git

Базовая работа с репозиторием и удаленным хостингом GitHub

Цель работы:
Понять необходимость использования систем контроля версий, освоить базовые команды `Git`, научиться публиковать код на `GitHub` и соблюдать правила хорошего тона при работе с репозиторием.

Почему это важно:

- Контроль изменений кода
- Командная работа
- Резервное копирование
- История проекта

Теория: Зачем нужна система контроля версий?

Система контроля версий (VCS) — это программное обеспечение, которое помогает отслеживать изменения в файлах, хранить историю правок и координировать работу нескольких человек над одним проектом.

Основные возможности Git:

1. **История изменений** — можно вернуться к любой версии кода
2. **Ветвление** — параллельная разработка фич
3. **Совместная работа** — слияние изменений от разных разработчиков
4. **Безопасность** — код хранится не только локально

Подготовка к работе

В работе будем использовать GitHub — создадим удаленный репозиторий и запушим туда код.

Необходимые инструменты:

- Git (установленный на компьютер)
- Аккаунт на GitHub
- Терминал / командная строка
- VS Code (или любой другой редактор)

Проверка установки Git:

Введите в терминале:

```
# Проверяем версию Git
git --version

# Настраиваем имя пользователя и email (один раз)
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Шаг 1. Создание локального репозитория

Создадим папку для проекта и инициализируем Git-репозиторий.

Команды в терминале:

```
# Создаем папку проекта (вместо surname – Ваша фамилия)
mkdir my-git-project-surname
cd my-git-project-surname

# Инициализируем Git-репозиторий
git init
```

Результат:

- Появится скрытая папка `.git` — это служебная директория, где Git хранит всю историю и настройки.
- **Важно:** `.git` нельзя удалять, иначе потеряется история версий.

Шаг 2. Первый коммит (логически завершенный кусок)

Создадим файл `README.md` и сделаем первый коммит.

Создание файла в терминале:

```
# Создаем файл с описанием проекта
echo "# Мой первый Git-проект" > README.md
echo "Это учебный проект для знакомства с Git" >> README.md
```

Создайте еще один файл:

```
echo "# здесь будут мои заметки" > mynotes.txt
```

Работа с Git:

```
# Смотрим состояние репозитория (покажет новый файл)
git status

# Добавляем файл в индекс (подготовка к коммиту)
git add README.md

# Делаем коммит с осмысленным сообщением
git commit -m "Добавлен README.md с описанием проекта"
```

Синтаксис команд (подробно)

Базовые команды Git:

Команда	Описание
git init	Создать новый репозиторий в текущей папке
git status	Показать состояние файлов (измененные, новые)
git add <файл>	Добавить конкретный файл в индекс
git add -A или git add .	Добавить **все** изменения в индекс
git commit -m "сообщение"	Закоммитить изменения с описанием
git log	Посмотреть историю коммитов

Важное правило:

Коммиты должны быть маленькими и частыми. Каждый коммит — логически завершенное изменение.

Шаг 3. Добавляем код и .gitignore

Создадим файл с кодом и настроим игнорирование ненужных файлов.

Создаем файл `script.js`:

```
// script.js
function greet(name) {
    return `Привет, ${name}!`;
}

console.log(greet('Мир'));
```

Создаем файл `.gitignore` (в корне проекта).

Открываем код файла и добавляем:

```
# игнорируем mynotes
mynotes.txt
# или так:
*.txt
```

```
# Игнорируем системные файлы
.DS_Store
Thumbs.db

# Игнорируем зависимости (если бы был Node.js)
node_modules/
.env

# Временные файлы
*.log
*.tmp
```

Шаг 4. Коммитим изменения

Теперь закоммитим новые файлы.

```
# Проверяем статус (увидим новые файлы)
git status

# Добавляем ВСЕ изменения (удобно, когда файлов много)
git add -A

# Или добавляем точечно:
# git add script.js .gitignore

# Делаем коммит
git commit -m "Добавлен скрипт script.js и настроен .gitignore"

# Смотрим историю коммитов
git log --oneline
```

Результат: В истории теперь два коммита.

Шаг 5. Создание репозитория на GitHub

1. Зайдите на GitHub
2. Нажмите зеленую кнопку **"New"** (или "+" в правом верхнем углу)
3. Заполните:
   - Repository name: `my-git-project-surname`
   - Описание (необязательно)
   - Оставьте **Public** (или Private, если хотите скрытый)
   - **НЕ** ставьте галочку `"Initialize this repository with a README"` (у нас уже есть)
4. Нажмите **"Create repository"**

Шаг 6. Связываем локальный и удаленный репозиторий

GitHub покажет инструкции. Нам нужен раздел **"…or push an existing repository from the command line"**.

Выполняем команды:

```
# Добавляем ссылку на удаленный репозиторий
# origin — стандартное имя для главного удаленного репозитория
git remote add origin https://github.com/ВАШ-АККАУНТ/my-git-project-surname.git

# Пушим в первый раз (флаг -u связывает локальную и удаленную ветки)
git push -u origin master
```

Примечание: Вместо *master* может быть *main* (зависит от настроек GitHub).

Шаг 7. Проверяем на GitHub

Обновите страницу вашего репозитория на GitHub.

Вы должны увидеть:

- Все файлы (`README.md`, `script.js`, `.gitignore`)
- Сообщения коммитов
- Время последнего изменения

Поздравляю! Код успешно опубликован в интернете.

Шаг 8. Вносим изменения и пушим снова

Теперь изменим код и отправим новую версию.

Изменяем script.js**:**

```
// script.js - добавляем новую функцию
function greet(name) {
   return `Привет, ${name}!`;
}
function farewell(name) {
   return `Пока, ${name}!`;
}
console.log(greet('Мир'));
console.log(farewell('Друг'));
```

Отправляем изменения:

```
# Проверяем изменения
git status
# Добавляем измененный файл
git add script.js
# Коммитим (маленькое, логически завершенное изменение!)
git commit -m "Добавлена функция farewell"
# Пушим (теперь без -u, связь уже настроена)
git push
```

Работа с коммитами в VS Code (встроенная поддержка Git)

VS Code имеет встроенную поддержку Git, но важно понимать, какие команды выполняются под капотом.

В VS Code можно:

1. Открыть вкладку **Source Control** (Ctrl+Shift+G)
2. Видеть измененные файлы (буква **M**)
3. Вводить сообщение коммита и нажимать галочку (это = git commit)
4. Нажимать кнопку "..." и выбирать Push/Pull

Но в рамках работы мы учимся работать в консоли, чтобы понимать процессы!

Шаг 9. Имитация работы в команде (git pull)

Представим, что другой разработчик изменил код на GitHub. Научимся забирать изменения.

Создадим изменение прямо на GitHub:

- Зайдите в репозиторий
- Откройте README.md
- Нажмите карандаш (Edit)
- Добавьте строку (Ваша фамилия): Автор: Студент Иванов
- Нажмите "Commit changes"

Забираем изменения локально:

```
# Скачиваем изменения с удаленного репозитория
git pull
```

Результат: Файл README.md обновится локально.

Сводная таблица команд

Команда	Описание	Когда использовать
git init	Инициализация	Только один раз, в начале проекта
git status	Проверка статуса	Постоянно, чтобы видеть изменения
git add <файл>	Добавить файл	Перед каждым коммитом
git commit -m "..."	Создать коммит	После каждого логического блока
git push	Отправить на GitHub	После коммита, чтобы сохранить в облаке
git pull	Забрать с GitHub	Перед началом работы, после работы команды
git log	История коммитов	Чтобы вспомнить, что делали

Задание для самостоятельной работы

1. Создайте новый репозиторий на GitHub (не забудьте .gitignore для Node.js).
2. Склонируйте его к себе (команда git clone <url>).
3. Добавьте файл *index.html* с простой страницей.
4. Сделайте коммит и пуш.
5. Внесите изменения в *index.html* через интерфейс GitHub (в браузере).
6. Сделайте *git pull* локально и убедитесь, что изменения пришли.
7. Добавьте локально в проект папку *temp* и создайте в ней текстовый файл с каким-то текстом.
8. Попробуйте добавить в .gitignore папку *temp/* и проверьте, что файлы в ней не отслеживаются.

Вопросы для самопроверки:

- Что будет, если сделать git add -A, а потом изменить файл до коммита?
- Зачем нужен флаг -u в первом git push?
- Чем Git отличается от GitHub?