Git

Система контроля версий

Занятие 1

Frontend Course 2025

Что такое Git?

> **Вопрос:** У вас есть файлы "курсовая_финал.docx", "курсовая_финал_2.docx", "курсовая_РЕАЛЬНО_финал.docx". Знакомо?

Git решает эту проблему. Он отслеживает все изменения в файлах, позволяет вернуться к любой версии и работать в команде без конфликтов.

Git vs обычное копирование

> **Вопрос:** В чем разница между Git и просто копированием папок?

- Git хранит только изменения, а не полные копии
- Показывает что именно изменилось и кто это сделал
- Позволяет нескольким людям работать одновременно
- Есть "машина времени" для кода

Основные понятия

> **Вопрос:** Что означают слова: репозиторий, коммит, ветка?

Репозиторий — папка с проектом под контролем Git
Коммит — снимок состояния проекта в момент времени
Ветка — параллельная линия разработки
Staging area — область подготовки файлов к коммиту

Настройка Git

```
# Представиться Git
git config --global user.name "Ваше Имя"
git config --global user.email "your@email.com"

# Проверить настройки
git config --list
```

> **Вопрос:** Зачем Git знать мое имя?

Каждый коммит подписывается автором. Так команда знает, кто и когда делал изменения.

Базовый workflow

```
# Создать репозиторий
git init

# Добавить файлы в staging area
git add index.html
git add .

# Создать коммит
git commit -m "Add homepage"

# Посмотреть статус
git status
```

Просмотр истории

> **Вопрос:** Как узнать, что происходило в проекте?

```
git log                    # полная история
git log --oneline          # компактный вид
git log --graph --all      # с визуализацией веток
git status                 # текущее состояние
git diff                   # что изменилось
```

Работа с ветками

> **Вопрос:** Зачем нужны ветки?

- Экспериментировать без страха сломать основной код
- Работать над разными функциями параллельно
- Легко переключаться между задачами

```
git branch feature-login     # создать ветку
git checkout feature-login   # переключиться
git checkout -b new-feature  # создать и переключиться
```

Слияние веток

```
git checkout main
git merge feature-login
```

> **Вопрос:** Что если в двух ветках изменили один файл?

Конфликт! Git покажет где проблема, вы решите вручную, и всё будет хорошо.

GitHub

> **Вопрос:** GitHub это тот же Git?

Нет. Git — инструмент на компьютере, GitHub — облачное хранилище + социальная сеть для программистов.

```
git remote add origin https://github.com/user/repo.git
git push -u origin main    # отправить код
git pull origin main       # скачать изменения
```

SSH ключи

> **Вопрос:** Надоело вводить пароль каждый раз. Есть решение?

SSH ключи. Создаете пару ключей, публичный загружаете на GitHub, работаете без паролей.

```
ssh-keygen -t ed25519 -C "your@email.com"
cat ~/.ssh/id_ed25519.pub  # скопировать в GitHub
```

Stash

> **Вопрос:** Работаю над функцией, но нужно срочно переключиться на багфикс. Что делать?

Stash временно "прячет" изменения, позволяет переключиться, а потом вернуть всё обратно.

```
git stash      # спрятать изменения
git stash pop  # вернуть изменения
```

Хорошие commit messages

> **Вопрос:** Как писать сообщения коммитов?

```
# Плохо
git commit -m "fix"

# Хорошо
git commit -m "fix: correct email validation in contact form"
git commit -m "feat: add user authentication"
git commit -m "docs: update README"
```

Conventional Commits — стандарт для читаемых сообщений.

Commitlint — инструмент для автоматической проверки формата коммитов.

Стратегии ветвления

> **Вопрос:** Как организовать ветки в команде?

Git Flow (в наших командах):
• master — стабильная версия
• features/{PROJECT_NAME}-{TASK_NUMBER} — ветки с разрабатываемым функционалом, например, features/EC-2345.
• bugfixes/{PROJECT_NAME}-{TASK_NUMBER} — ветки с багами, например, bugfixes/EC-2334.
• releases/{VERSION} — ветки, которые выливаются на сервера, например, releases/1.2.133.
• Pull Request для внесения изменений в master ветку.

Восстановление

> **Вопрос:** Сделал git reset --hard и потерял код!

Не паникуйте! Git почти ничего не удаляет навсегда.

```
git reflog                    # история всех действий
git reset --hard HEAD@{2}     # восстановить состояние

# Забыли файл в коммите?
git add forgotten.js
git commit --amend --no-edit
```

Практика

Создаем первый репозиторий:

1. `git init`
2. Создать README.md
3. `git add README.md`
4. `git commit -m "Initial commit"`
5. Создать репозиторий на GitHub
6. `git remote add origin <URL>`
7. `git push -u origin main`

Главные правила

- Коммитьте часто, но логично
- Пишите понятные сообщения
- Не пушьте сломанный код в main
- Используйте ветки для экспериментов
- `git status` — ваш лучший друг

Домашнее задание

- Создать репозиторий `frontend-course-homework`
- Настроить Git с вашими данными
- Сделать первые коммиты
- Попрактиковаться с ветками
- Подключить к GitHub

Следующее занятие: HTML

Вопросы?

Git теперь ваш друг.
Главное — не бояться экспериментировать.