

Практическая работа 2.2. Создание и управление репозиториями на GitHub. Работа с ветками, слияниями, разрешение конфликтов

Цель работы: познакомиться с основными концепциями и командами Git, научиться использовать Git для управления версиями проекта.

Задачи

Создание и настройка репозитория на GitHub.

Работа с ветками (branches).

Слияние веток (merge).

Разрешение конфликтов.

Теоретическая часть

Git - это распределенная система контроля версий, которая позволяет отслеживать изменения в файлах и координировать работу нескольких людей над этими файлами.

GitHub - это веб-сервис для хостинга Git-репозиториях.

Необходимое программное обеспечение

Git (<https://git-scm.com/>).

Аккаунт на GitHub (<https://github.com/>).

Текстовый редактор (VS Code, Sublime Text и т.д.).

Ход работы

1. Создание репозитория

Создать тестовый репозиторий. Добавьте в репозиторий два файла двумя разными коммитами, файлы нужно создать, если их нет:

Файл `mgpu.txt` с текстом `Hello, MGPU!` внутри

Файл `index.html` с текстом `<h1> я коммичу</h1>` внутри

`pwd` – текущая директория.

`cd` – переход в другую директорию.

Для выполнения задания в Git, нужно последовательно создать два файла и добавить их в репозиторий двумя отдельными коммитами. Вот пошаговая инструкция:

1. Инициализация репозитория (если еще не сделано):

```
```bash
git init
```

2. Создание файла `mgpu.txt` с текстом `"Hello, MGPU!"`:

```
```bash
echo " Hello, MGPU!" > mgpu.txt
```

3. Добавление файла в индекс и создание первого коммита:

```
```bash
git add mgpu.txt
git commit -m "Добавлен файл mgpu.txt с текстом 'Hello, MGPU!'"
```

4. Создание файла `index.html` с текстом `"<h1> я коммичу</h1>"`:

```
```bash
echo "<h1> я коммичу</h1>" > index.html
```

5. Добавление файла в индекс и создание второго коммита:

```
```bash
git add index.html
git commit -m "Add index.html with header 'я коммичу'"
```

Теперь в репозитории есть два файла, добавленных двумя разными коммитами.

## 6. Клонирование репо

# Первый параметр «откуда», второй — «куда»

```
$ git clone repos/git-user code-user
```

Перейдите в директорию `code-user`, в которой находится клонированный репозиторий. В репозитории уже есть два файла. Измените их так:

В `mgpu.txt` добавьте второй строчкой текст *I like to change files*

В `index.html` замените текст на `<h1>С помощью Git можно писать книги</h1>`

Выполните один коммит, содержащий сразу эти два изменения. Во время коммита Git попросит ввести электронную почту и имя пользователя.

Добавьте изменения в основной репозиторий с помощью **git push**

После добавления электронной почты и имени пользователя требуется повторить коммит, так как предыдущий коммит из-за отсутствия настроек выполнен не был.

## 7. Создание и настройка репозитория

Создайте аккаунт на GitHub (если еще нет).

Создайте новый репозиторий на GitHub:

Нажмите "New repository"

Укажите имя: "practice-git-basics"

Добавьте README.md

Выберите лицензию (например, MIT)

Клонируйте репозиторий:

```
```bash
git clone https://github.com/ваш-username/practice-git-basics.git
cd practice-git-basics
```

8. Работа с ветками

Создайте новую ветку feature-1:

```
```bash
git checkout -b feature-1
Внесите изменения в README.md:
Practice Git Basics
This is a practice repository for learning Git basics.
Features
- Feature 1: Coming soon
```

Зафиксируйте изменения:

```
```bash
git add README.md
git commit -m "Add feature 1 section to README"
git push origin feature-1
```

9. Слияние веток

Переключитесь на main ветку:

```
```bash
```

```
git checkout main
```

Создайте новую ветку feature-2:

```
```bash
```

```
git checkout -b feature-2
```

Измените README.md в feature-2:

```
# Practice Git Basics
```

```
This is a practice repository for learning Git basics.
```

```
## Features
```

```
- Feature 2: Work in progress
```

10. Зафиксируйте и отправьте изменения:

```
```bash
```

```
git add README.md
```

```
git commit -m "Add feature 2 section to README"
```

```
git push origin feature-2
```

### Критерии оценки

Выполнение задания для самостоятельной работы (100 баллов).

#### Отчетность

- По результатам работы необходимо подготовить отчет, содержащий:
- Титульный лист.
- Цель работы.
- Ход выполнения работы (включая скриншоты).
- Выводы.

### Варианты заданий

#### Для всех вариантов

Работа в Linux подобных ОС не требует установки Git, он идет по умолчанию. Если используете другие ОС(например, Windows), тогда

[воспользоваться инструкцией](https://github.com/BosenkoTM/DevOps-Technologies/blob/main/Lesson%20%20Source%20Code%20Management%20with%20Git%20and%20GitOps/git_source_code_management/git.md)

[\\_ \(https://github.com/BosenkoTM/DevOps-](https://github.com/BosenkoTM/DevOps-Technologies/blob/main/Lesson%20%20Source%20Code%20Management%20with%20Git%20and%20GitOps/git_source_code_management/git.md)

[Technologies/blob/main/Lesson%20%20Source%20Code%20Management%20with%20Git%20and%20GitOps/git\\_source\\_code\\_management/git.md\)](https://github.com/BosenkoTM/DevOps-Technologies/blob/main/Lesson%20%20Source%20Code%20Management%20with%20Git%20and%20GitOps/git_source_code_management/git.md)

**Задание 1.** Создание каталога, перенос его в Git, создания файлов, настройки .gitignore и загрузки в GitHub на Ubuntu 24:

#### 1. Создание каталога:

```
mkdir my_project
```

```
cd my_project
```

#### 2. Инициализация Git репозитория:

```
git init
```

### 3. Создание файлов:

```
touch README.md
```

```
touch main.py
```

### 4. Создание .gitignore:

```
nano .gitignore
```

Добавьте в файл типичные исключения, например:

```
*.log
```

```
__pycache__/
```

```
.env
```

### 5. Добавление файлов в Git:

```
git add .
```

### 6. Создание первого коммита:

```
git commit -m "Initial commit"
```

### 7. Создание репозитория на GitHub:

- Откройте GitHub в браузере

- Нажмите "+" в правом верхнем углу и выберите "New repository"

- Назовите репозиторий и настройте его

### 8. Связывание локального репозитория с GitHub:

```
git remote add origin https://github.com/your_username/your_repository.git
```

### 9. Отправка изменений на GitHub:

```
git push -u origin main
```

Давайте разберемся с этим шаг за шагом. Есть несколько причин, по которым сталкиваемся с трудностями при связывании локального репозитория с GitHub.

### 1. Проверьте, правильно ли настроен удаленный репозиторий:

Выполните команду:

```
git remote -v
```

Эта команда покажет список настроенных удаленных репозиториях.

### 2. Если удаленный репозиторий не настроен, добавьте его:

```
git remote add origin https://github.com/your_username/your_repository.git
```

Замените "your\_username" и "your\_repository" на ваши данные.

### 3. Если возникает ошибка при добавлении удаленного репозитория, возможно, он уже существует. Попробуйте изменить существующий:

```
git remote set-url origin https://github.com/your_username/your_repository.git
```

### 4. Убедитесь, что авторизованы в GitHub. Если используете HTTPS, Git может запросить ваши учетные данные GitHub.

### 5. Если используете SSH, убедитесь, что SSH-ключ правильно настроен в GitHub. Проверьте это командой:

```
ssh -T git@github.com
```

### 6. Попробуйте отправить изменения:

```
git push -u origin main
```

Замените "main" на название основной ветки, если она называется иначе (например, "master").

## Настройка SSH для GitHub.

### 1. Проверьте наличие существующих SSH-ключей:

```
ls -al ~/.ssh
```

```
devops@devopsvm:~/Downloads/my_first_project$ ls -al ~/.ssh
total 24
drwx----- 2 devops devops 4096 Sep 30 22:58 .
drwxr-x--- 23 devops devops 4096 Sep 30 22:40 ..
-rw----- 1 devops devops 0 Aug 17 13:34 authorized_keys
-rw----- 1 devops devops 411 Sep 30 22:53 id_ed25519
-rw-r--r-- 1 devops devops 99 Sep 30 22:53 id_ed25519.pub
-rw----- 1 devops devops 978 Sep 30 22:58 known_hosts
-rw-r--r-- 1 devops devops 142 Sep 30 22:58 known_hosts.old
```

Если видите файлы вроде `id_rsa.pub`, `id_ecdsa.pub` или `id_ed25519.pub`, у вас уже есть SSH-ключ.

### 2. Если у нет SSH-ключа, создайте новый:

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

Замените `"your_email@example.com"` на email, связанный с вашим GitHub аккаунтом.

### 3. Когда попросят ввести файл для сохранения ключа, нажмите Enter для использования расположения по умолчанию.

### 4. Введите пароль для ключа (рекомендуется) или оставьте поле пустым.

### 5. Запустите SSH-агент:

```
eval "$(ssh-agent -s)"
```

### 6. Добавьте ваш SSH-ключ в ssh-agent:

```
ssh-add ~/.ssh/id_ed25519
```

### 7. Скопируйте ваш публичный SSH-ключ в буфер обмена:

```
cat ~/.ssh/id_ed25519.pub | xclip -selection clipboard
```

Если у вас нет `xclip`, можете просто вывести ключ в терминал и скопировать его вручную:

```
cat ~/.ssh/id_ed25519.pub
```

### 8. Перейдите на GitHub:

- Нажмите на ваш профиль в правом верхнем углу.
- Выберите "Settings".
- В левом меню выберите "SSH and GPG keys".
- Нажмите "New SSH key" или "Add SSH key".

### 9. Вставьте ваш ключ в поле "Key".

### 10. Определите название ключа в поле "Title".

### 11. Нажмите "Add SSH key".

### 12. Проверьте подключение:

```
ssh -T git@github.com
```

Должны увидеть приветственное сообщение с вашим именем пользователя GitHub.

```
devops@devopsvm:~/Downloads/my_first_project$ ssh -T git@github.com
Hi BosenkoTM! You've successfully authenticated, but GitHub does not provide shell access.
```

**12. Убедитесь, что находитесь в директории вашего Git-репозитория:**

`cd /path/to/your/repository`

**13. Проверьте статус ваших файлов:**

`git status`

```
devops@devopsvm:~/Downloads/my_first_project$ git status
On branch main
nothing to commit, working tree clean
```

Это покажет, какие файлы изменены или добавлены.

**14. Добавьте файлы, которые хотите отправить, в индекс Git:**

`git add .`

Эта команда добавит все новые и измененные файлы. Если вы хотите добавить конкретные файлы, укажите их имена вместо точки.

**15. Создайте коммит с описанием изменений:**

`git commit -m "Описание ваших изменений"`

**16. Убедитесь, что ваш удаленный репозиторий на GitHub настроен с использованием SSH. Проверьте это командой:**

`git remote -v`

Должны увидеть URL, начинающийся с `git@github.com:`.

**17. Если удаленный репозиторий еще не настроен или настроен на HTTPS, измените его на SSH:**

`git remote set-url origin git@github.com:username/repository.git`

Замените `username` на ваше имя пользователя GitHub, а `repository` на название вашего репозитория.

**18. Отправьте ваши изменения на GitHub:**

`git push origin main`

Замените `main` на название вашей ветки, если она называется иначе.

**19. Если это ваш первый push в репозиторий, используйте команду:**

`git push -u origin main`

```
devops@devopsvm:~/Downloads/my_first_project$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 298 bytes | 298.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:BosenkoTM/my_project.git
 * [new branch] main -> main
```

Это установит связь между локальной и удаленной веткой. Если все настроено правильно, не должны получить запрос на ввод пароля. GitHub будет использовать ваш SSH-ключ для аутентификации.

Если получаете ошибки или запрос на ввод пароля, убедитесь, что:

- SSH-ключ добавлен в SSH-агент (`ssh-add ~/.ssh/id_ed25519`).
- публичный ключ добавлен в настройках аккаунта на GitHub.

- URL удаленного репозитория использует SSH формат.

**Задание 2.** Создать новый файл и отправить изменения на удаленный репозиторий в GitHub. Вот пошаговая инструкция:

**1. Перейдите в директорию вашего локального Git-репозитория:**

`cd /путь/к/вашему/репозиторию`

**2. Создайте новый файл (например, test.txt):**

`touch test.txt`

**3. Добавьте содержимое в файл (например, с помощью редактора nano):**

`nano test.txt`

Введите какой-нибудь текст, сохраните файл (Ctrl+O, затем Enter) и выйдите из редактора (Ctrl+X).

**4. Проверьте статус репозитория:**

`git status`

Вы должны увидеть ваш новый файл в списке неотслеживаемых файлов.

```
devops@devopsvm:~/Downloads/my_first_project$ git status
```

```
On branch main
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
test.txt
```

**5. Добавьте файл в индекс Git:**

`git add test.txt`

```
devops@devopsvm:~/Downloads/my_first_project$ git status
```

```
On branch main
```

```
Changes to be committed:
```

```
(use "git restore --staged <file>..." to unstage)
```

```
new file: test.txt
```

**6. Создайте коммит:**

`git commit -m "Добавлен новый файл test.txt"`

**7. Отправьте изменения на GitHub:**

`git push origin main`

(Замените 'main' на название вашей ветки, если она называется иначе)

## Варианты заданий

### **Вариант 1.** Рабочий процесс и интеграция с GitHub.

Создайте новый репозиторий на GitHub и склонируйте его на свой компьютер. Добавьте несколько файлов, сделайте коммиты и отправьте изменения на GitHub. Затем создайте новую ветку, внесите изменения и создайте pull request.

### **Вариант 2.** Рабочая директория и анализ сделанных изменений.

Создайте локальный репозиторий, добавьте несколько файлов и сделайте первый коммит. Внесите изменения в файлы и используйте команды `git status` и `git diff` для анализа сделанных изменений. Опишите, что показывают эти команды.

### **Вариант 3.** Анализ истории изменений и отмена изменений в рабочей директории:

Создайте репозиторий с несколькими коммитами. Используйте `git log` для просмотра истории изменений. Затем внесите изменения в файл, но не коммитьте их. Используйте `git checkout` для отмены этих изменений.

### **Вариант 4.** Отмена коммитов и изменение последнего коммита:

Создайте репозиторий и сделайте несколько коммитов. Используйте `git revert` для отмены одного из предыдущих коммитов. Затем сделайте небольшое изменение и используйте `git commit --amend` для изменения последнего коммита.

### **Вариант 5.** Индекс и перемещение по истории:

Создайте репозиторий с несколькими файлами и коммитами. Используйте `git add` для добавления изменений в индекс, но не коммитьте их. Затем используйте `git checkout` для перемещения на предыдущий коммит и обратно.

### **Вариант 6.** Понимание Git и игнорирование файлов:

Создайте репозиторий и добавьте файл `.gitignore`. Настройте игнорирование определенных типов файлов и директорий. Затем создайте файлы разных типов и продемонстрируйте, как работает игнорирование.

### **Вариант 7.** Stash и рабочая директория:

Начните работу над новой функцией в репозитории. Не завершив работу, получите срочную задачу исправить баг. Используйте `git stash` для сохранения текущих изменений, исправьте баг в новой ветке, затем вернитесь к работе над функцией, применив `stash`.

### **Вариант 8.** Интеграция с GitHub и анализ истории изменений:

Форкните открытый проект на GitHub. Клонировать свой форк, внесите изменения и создайте pull request в оригинальный репозиторий. Используйте `git log` и `git show` для анализа истории изменений в проекте.

### **Вариант 9.** Отмена коммитов и работа с ветками:

Создайте репозиторий с основной и дополнительной ветками. Внесите изменения в обе ветки. Затем используйте `git reset` для отмены коммита в основной ветке и выполните слияние веток, разрешив возникшие конфликты.

### **Вариант 10.** Рабочий процесс и Stash:

Смоделируйте рабочий процесс, где вы работаете над несколькими задачами параллельно. Используйте ветки для разных задач и `git stash` для быстрого переключения между ними. Продемонстрируйте, как управлять несколькими stash'ами и применять их выборочно.



**Вариант 11. Работа с удаленными репозиториями:**

Создайте локальный репозиторий и свяжите его с новым пустым репозиторием на GitHub. Добавьте несколько коммитов локально, затем настройте еще один удаленный репозиторий (например, на GitLab). Продемонстрируйте, как push и pull изменяют с обоими удаленными репозиториями.

**Вариант 12. Продвинутое использование индекса:**

Создайте репозиторий с несколькими файлами. Внесите изменения в разные части каждого файла. Используйте `git add -p` для интерактивного добавления только определенных изменений в индекс. Создайте коммит из этих выборочных изменений.

**Вариант 13. Работа с тегами и релизами:**

Создайте репозиторий, имитирующий разработку программного продукта. Добавьте несколько коммитов, представляющих различные этапы разработки. Используйте `git tag` для маркировки важных точек (например, версий). Создайте релиз на GitHub, используя один из тегов.

**Вариант 14. Работа с подмодулями:**

Создайте два репозитория: основной проект и библиотеку. Добавьте библиотеку как подмодуль в основной проект. Продемонстрируйте, как обновлять подмодуль и коммитить изменения в обоих репозиториях.

**Вариант 15. Использование git bisect:**

Создайте репозиторий с рядом коммитов, один из которых "ломает" функциональность. Напишите скрипт для автоматической проверки работоспособности. Используйте `git bisect` для автоматического нахождения коммита, вызвавшего проблему.

**Вариант 16. Работа с большими файлами:**

Настройте Git LFS (Large File Storage) в репозитории. Добавьте несколько больших файлов (например, изображений или архивов) и продемонстрируйте, как Git LFS управляет ими. Покажите разницу между обычными файлами и файлами, управляемыми через LFS.

**Вариант 17. Настройка Git hooks:**

Создайте репозиторий и настройте несколько Git hooks (например, `pre-commit` для проверки стиля кода и `post-receive` для автоматического деплоя). Продемонстрируйте, как эти hooks влияют на рабочий процесс.

**Вариант 18. Работа с патчами:**

Создайте два репозитория с похожим кодом. В одном репозитории внесите изменения и создайте патч с помощью `git format-patch`. Затем примените этот патч ко второму репозиторию, используя `git apply`.

**Вариант 19. Использование reflog:**

Создайте репозиторий с несколькими ветками и коммитами. Выполните ряд операций, включая слияние веток и `reset`. Используйте `git reflog` для восстановления "потерянных" коммитов и состояний репозитория.

**Вариант 20. Работа с cherry-pick:**

Создайте репозиторий с несколькими ветками разработки. Выберите конкретные коммиты из разных веток и примените их к основной ветке, используя git cherry-pick. Разрешите возникающие конфликты.

**Вариант 21. Использование rebase:**

Создайте основную ветку и несколько тематических веток с коммитами. Используйте git rebase для актуализации тематических веток относительно основной. Продемонстрируйте интерактивный rebase для изменения истории коммитов.

**Вариант 22. Работа с gitflow:**

Настройте репозиторий, следуя модели gitflow. Создайте ветки feature, release и hotfix. Продемонстрируйте полный цикл разработки функции и выпуска версии согласно этой модели.

**Вариант 23. Анализ и оптимизация репозитория:**

Создайте репозиторий с большим количеством коммитов и файлов. Используйте git gc для оптимизации репозитория. Проанализируйте размер репозитория до и после оптимизации. Используйте git count-objects для подробной статистики.

**Вариант 24. Работа с subversion через git:**

Создайте репозиторий Subversion и клонируйте его с помощью git svn. Внесите изменения, сделайте коммиты в Git и отправьте их обратно в Subversion. Продемонстрируйте основные операции работы с SVN через Git.

**Вариант 25. Настройка и использование Git alias:**

Создайте несколько полезных Git alias для упрощения часто используемых команд. Например, создайте alias для просмотра красивого лога, быстрого коммита с сообщением, или для сложной последовательности команд. Продемонстрируйте использование этих alias в рабочем процессе.

**Вариант 26. Работа с Git worktree:**

```
```bash
```

```
# Создайте новый репозиторий и добавьте несколько коммитов
```

```
git init git-worktree-demo
```

```
cd git-worktree-demo
```

```
echo "Initial content" > file.txt
```

```
git add file.txt
```

```
git commit -m "Initial commit"
```

```
# Создайте новую ветку и добавьте в нее коммит
```

```
git branch feature-branch
```

```
git checkout feature-branch
```

```
echo "Feature content" >> file.txt
```

```
git commit -am "Add feature content"
```

```
# Вернитесь в main и создайте worktree для feature-branch
```

```
git checkout main
```

```
git worktree add ../feature-worktree feature-branch
```

```
# Внесите изменения в обоих worktree и создайте коммиты
```

```
echo "Main content" >> file.txt
```

```
git commit -am "Add main content"
```

```
cd ../feature-worktree
```

```
echo "More feature content" >> file.txt
```

```
git commit -am "Add more feature content"
```

```
# Вернитесь в основной worktree и выполните слияние
```

```
cd ../git-worktree-demo
```

```
git merge feature-branch
```

```
# Удалите дополнительный worktree
```

```
git worktree remove ../feature-worktree
```

```
...
```

Задание. Выполните указанные в скрипте шаги для работы с Git worktree. После каждого шага используйте ``git log --all --graph --oneline`` для визуализации структуры репозитория. Опишите, как использование worktree помогает в параллельной работе над разными ветками.

Вариант 27. Использование Git blame и Git annotate:

Создайте репозиторий с файлом, содержащим не менее 50 строк кода. Внесите изменения в этот файл в нескольких коммитах, выполненных разными авторами (можно имитировать разных авторов, меняя Git config). Используйте ``git blame`` и ``git annotate`` для анализа истории изменений файла. Объясните различия между этими командами и их практическое применение в процессе разработки.

Вариант 28. Работа с Git filter-branch:

```
```bash
```

```
Создайте репозиторий с несколькими коммитами, содержащими
конфиденциальную информацию
```

```
git init filter-branch-demo
```

```
cd filter-branch-demo
```

```
echo "Public info" > public.txt
```

```
echo "Secret: password123" > secret.txt
```

```
git add .
```

```
git commit -m "Initial commit with secret"
```

```
echo "More public info" >> public.txt
```

```
git commit -am "Add more public info"
```

```
echo "Another secret: api_key456" >> secret.txt
```

```
git commit -am "Add another secret"
```

```
Используйте filter-branch для удаления secret.txt из истории
```

```
git filter-branch --force --index-filter \
"git rm --cached --ignore-unmatch secret.txt" \
--prune-empty --tag-name-filter cat -- --all
```

```
Принудительно обновите все ссылки и выполните сборку мусора
git for-each-ref --format="% (refname)" refs/original/ | xargs -n 1 git update-ref -d
git reflog expire --expire=now --all
git gc --prune=now
```

...

**Задание.** Выполните указанные шаги для создания репозитория с конфиденциальной информацией и ее последующего удаления из истории. Объясните, почему простое удаление файла новым коммитом недостаточно для защиты конфиденциальной информации. Обсудите этические аспекты изменения истории репозитория.

**Вариант 29.** Использование Git rerere (Reuse Recorded Resolution):

Создайте репозиторий с двумя ветками, которые имеют повторяющиеся конфликты при слиянии. Включите функцию rerere с помощью `git config --local rerere.enabled true`. Выполните слияние веток, разрешая конфликты вручную. Затем отмените слияние и повторите его снова. Продемонстрируйте, как Git автоматически разрешает конфликты, используя ранее записанные решения. Обсудите преимущества и потенциальные риски использования этой функции.

**Вариант 30.** Работа с Git refsPEC:

Создайте локальный репозиторий и свяжите его с удаленным. Создайте несколько веток локально и удаленно. Используйте различные refsPEC для выборочной синхронизации определенных веток между локальным и удаленным репозиториями. Например, настройте push только для веток с определенным префиксом или pull только для определенных тегов. Продемонстрируйте, как refsPEC влияет на стандартные операции fetch, pull и push.