

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.

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.

#### 2. 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 нужно повторить коммит, так как предыдущий коммит из-за отсутствия настроек выполнен не был.

### Варианты заданий

#### Для всех вариантов

- Установите Git по [инструкции](#)
- Добавьте файл *PEOPLE.md* в репозиторий. После добавления команда `git status` покажет такой вывод:  
*git status*  
*On branch main*  
*nothing to commit, working tree clean*
- Создайте и добавьте ssh-ключи на GitHub [инструкции](#).
- Добавьте новый файл *NEW.md* с произвольным содержимым в репозиторий. Для этого нужно выполнить коммит.
- Залейте изменения на **GitHub** с помощью `git push`.
- Обновите страницу репозитория на **GitHub**. Там должен появиться последний коммит — те изменения, которые были совершены.
- Удалите файл *NEW.md* и сделайте коммит
- Добавьте файл *INFO.md* с текстом *git is awesome!* и сделайте коммит
- Залейте изменения на GitHub с помощью `git push`
- Обновите страницу репозитория на GitHub и изучите произошедшие изменения
- Сделайте коммит с сообщением *add new content*
- Залейте изменения на GitHub
- Изучите историю репозитория [самостоятельно подобрать](#)
- Измените добавленный текст на *No code No pain*
- Закоммитьте изменения с сообщением *update README.md*
- Залейте изменения на GitHub
- Залейте изменения на GitHub
- Добавьте файл *.gitignore* в проект
- Добавьте файл *INFO.md* в список игнорируемых файлов
- Удалите файл *INFO.md* из репозитория
- Создайте файл *INFO.md* и убедитесь в том, что `git status` его не отображает
- Загрузите изменения на GitHub

**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 изменения 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 `git 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.