

Лабораторная работа

Исправление ошибок в Git

Цель работы

Освоить продвинутые техники работы с историей коммитов в Git: изменение последнего коммита (**commit --amend**), откат изменений (**reset**), а также взаимодействие локального и удалённого репозиториев при переписывании истории.

Предварительные требования

- Установленный Git
- Аккаунт на GitHub
- Базовое понимание команд add, commit, push, pull

Часть 1. Подготовка рабочего окружения

Шаг 1.1. Создание репозитория

1. Создайте новый публичный репозиторий на GitHub с именем **git-history-lab**.
2. Не добавляйте README при создании.
3. Клонируйте пустой репозиторий на локальную машину:

```
git clone https://github.com/<ваш-username>/git-history-lab.git  
cd git-history-lab
```

Шаг 1.2. Настройка идентификации

```
git config user.name "Ваше Имя"  
git config user.email "ваш@email.com"
```

Часть 2. Работа с commit --amend

Теоретическая справка

Команда **git commit --amend** позволяет изменить **последний коммит**:

- Исправить сообщение коммита
- Добавить забытые файлы
- Изменить содержимое файлов

⚠ Важно: --amend создаёт новый коммит с новым хешем, заменяя предыдущий. Это переписывание истории!

Задание 2.1. Исправление сообщения коммита

Сценарий: Вы сделали коммит с опечаткой в сообщении.

1. Создайте файл calculator.py:

```
def add(a, b):
    return a + b
```

2. Сделайте коммит с **намеренной опечаткой**:

```
git add calculator.py
git commit -m "Добавлена фнукция сложения"
```

3. Проверьте историю:

```
git log --oneline
```

4. Исправьте сообщение коммита:

```
git commit --amend -m "Добавлена функция сложения"
```

5. Снова проверьте историю и убедитесь, что сообщение исправлено, хеш коммита изменился, количество коммитов осталось прежним.

 **Что сдать:** Скриншот вывода git log --oneline до и после --amend.

Задание 2.2. Добавление забытого файла в коммит

Сценарий: Вы забыли добавить файл в коммит.

1. Создайте файл tests.py:

```
from calculator import add

def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
```

2. Создайте файл README.md:

```
# Calculator
Простой калькулятор на Python.
```

3. Добавьте **только** tests.py и сделайте коммит:

```
git add tests.py
git commit -m "Добавлены тесты и документация"
```

4. Осознайте, что забыли добавить README.md. Проверьте статус: git status

5. Добавьте забытый файл в последний коммит:

```
git add README.md
git commit --amend --no-edit
```

Флаг --no-edit сохраняет прежнее сообщение коммита.

6. Проверьте, что оба файла теперь в одном коммите:

```
git show --name-only HEAD
```

 **Что сдать:** Скриншот вывода git show --name-only HEAD, показывающий оба файла.

Задание 2.3. Изменение содержимого файла в последнем коммите

Сценарий: Вы нашли баг сразу после коммита.

- Добавьте функцию деления в calculator.py:

```
def divide(a, b):
    return a / b
```

- Сделайте коммит:

```
git add calculator.py
git commit -m "Добавлена функция деления"
```

- Осознайте, что забыли обработку деления на ноль. Исправьте функцию:

```
def divide(a, b):
    if b == 0:
        raise ValueError("Деление на ноль невозможно")
    return a / b
```

- Внесите исправление в последний коммит:

```
git add calculator.py
git commit --amend --no-edit
```

 **Что сдать:** Содержимое файла calculator.py и вывод git log --oneline.

Часть 3. Работа с git reset

Теоретическая справка

git reset перемещает указатель ветки (HEAD) на указанный коммит. Существует три режима:

Режим	Индекс (staging)	Рабочая директория	Использование
--soft	Сохранён	Сохранена	Переделать коммит
--mixed	Сброшен	Сохранена	Переделать add + commit
--hard	Сброшен	Сброшена	Полностью откатить

 **Внимание:** --hard удаляет изменения безвозвратно!

Задание 3.1. Мягкий сброс (--soft)

Сценарий: Вы сделали коммит, но хотите объединить его с предыдущим.

- Добавьте функцию умножения и сделайте коммит:

```
def multiply(a, b):
    return a * b
git add calculator.py
git commit -m "Добавлено умножение"
```

- Добавьте функцию вычитания и сделайте коммит:

```
def subtract(a, b):
    return a - b
```

```
git add calculator.py
git commit -m "Добавлено вычитание"
```

3. Посмотрите историю: `git log --oneline -5`. Запишите хеши двух последних коммитов.
4. Откатитесь на 2 коммита назад с сохранением изменений в индексе:

```
git reset --soft HEAD~2
```

5. Проверьте статус: `git status`. Изменения должны быть в staged состоянии.
6. Сделайте один общий коммит:

```
git commit -m "Добавлены функции умножения и вычитания"
```

 **Что сдать:** Скриншоты `git log --oneline` до `reset`, `git status` после `reset`, `git log --oneline` после нового коммита.

Задание 3.2. Смешанный сброс (--mixed)

Сценарий: Вы случайно добавили в индекс файл, который не должен попасть в коммит.

1. Создайте файл с конфиденциальными данными:

```
echo "DB_PASSWORD=secret123" > config.env
```

2. Создайте файл `.gitignore` с содержимым `config.env`
3. Случайно добавьте оба файла:

```
git add .
```

4. Проверьте статус — оба файла в индексе: `git status`
5. Уберите `config.env` из индекса, но сохраните файл:

```
git reset HEAD config.env
```

6. Проверьте статус. Теперь в индексе только `.gitignore`.
7. Сделайте коммит:

```
git add .gitignore
```

```
git commit -m "Добавлен .gitignore"
```

 **Что сдать:** Скриншоты `git status` до и после `reset`.

Задание 3.3. Жёсткий сброс (--hard)

Сценарий: Вы сделали несколько неудачных коммитов и хотите полностью от них избавиться.

 **Предупреждение:** Эта операция необратима! Все незакоммиченные изменения будут потеряны.

1. Создайте «плохой» файл и закоммитьте:

```
echo "Это неправильный код" > bad_code.py
```

```
git add bad_code.py
```

```
git commit -m "Добавлен плохой код"
```

2. Внесите ещё изменения и закоммитьте:

```
echo "Ещё хуже" >> bad_code.py
```

```
git add bad_code.py
git commit -m "Сделано ещё хуже"
```

3. Посмотрите историю: `git log --oneline -5`
4. Выполните жёсткий сброс на 2 коммита назад:

```
git reset --hard HEAD~2
```

5. Проверьте: `git log --oneline` (коммитов нет), `ls` (файла `bad_code.py` нет).

 **Что сдать:** Скриншоты до и после `reset --hard`.

Часть 4. Взаимодействие с удалённым репозиторием

Задание 4.1. Проблема расхождения историй

Это критически важная часть! Поймите, что происходит при переписывании истории.

1. Отправьте текущее состояние на GitHub:

```
git push origin main
```

2. Сделайте новый коммит и отправьте:

```
echo "print('Hello')" > hello.py
git add hello.py
git commit -m "Приветствие"
git push origin main
```

3. Теперь измените этот коммит локально:

```
git commit --amend -m "Добавлено приветствие"
```

4. Попробуйте отправить изменения:

```
git push origin main
```

5. **Вы получите ошибку!** Объясните в отчёте: почему возникла ошибка?
Что означает сообщение об ошибке?

6. **Принудительная отправка** (используйте с осторожностью!):

```
git push --force origin main
```

 **Что сдать:** Скриншот ошибки при обычном `push`, объяснение причины (2–3 предложения), скриншот успешного `push --force`.

Задание 4.2. Безопасная альтернатива: --force-with-lease

1. Сделайте ещё один коммит и отправьте его.
2. Измените коммит через `--amend`.
3. Вместо `--force` используйте более безопасный вариант:

```
git push --force-with-lease origin main
```

 **Объясните в отчёте:** В чём преимущество `--force-with-lease` перед `--force`?

Часть 5. Комплексное задание

Сценарий

Вы работаете над проектом и сделали серию ошибок. Исправьте их, используя изученные команды.

Начальное состояние: У вас есть три последних коммита:

```
abc1234 Добавлен модуль авторизации
def5678 Исправлены тесты (содержит пароль в открытом виде!)
ghi9012 Обновлена документация (опечатка: "документация")
```

Требуется:

- Исправить опечатку в сообщении последнего коммита
- Удалить коммит с паролем из истории, сохранив исправления тестов без пароля
- Объединить исправленные тесты с коммитом авторизации

Для выполнения:

1. Создайте файлы (пример для Windows), имитирующие эту ситуацию:

```
# Коммит 1
echo "class Auth: pass" > auth.py
git add auth.py
git commit -m "Добавлен модуль авторизации"

# Коммит 2 (с паролем)
echo "PASSWORD = 'secret123'" > tests.py
echo "def test_auth(): assert True" >> tests.py
git add tests.py
git commit -m "Исправлены тесты"

# Коммит 3 (с опечаткой)
echo "# Auth Module" > docs.md
git add docs.md
git commit -m "Обновлена документация"
```

2. Теперь исправьте все проблемы, используя `--amend` и `reset`.
3. В итоге должно быть два чистых коммита:
 - «Добавлен модуль авторизации и тесты» (без пароля в `tests.py`)
 - «Обновлена документация»

 **Что сдать:** Последовательность команд (`git reflog`), финальный вывод `git log --oneline`, содержимое `tests.py` без пароля.

Контрольные вопросы

Ответьте письменно в отчёте:

1. В чём разница между `git reset --soft`, `--mixed` и `--hard`?

2. Почему нельзя использовать `--amend` для коммитов, которые уже отправлены в общий репозиторий?
3. Что произойдёт, если два разработчика работают с одной веткой и один из них сделает `push --force`?
4. Как восстановить коммит после `reset --hard`, если вы передумали? (подсказка: `git reflog`)
5. В каких случаях допустимо использовать `push --force`?

Полезные команды для отладки

```
git log --oneline --graph --all      # Визуализация истории  
git reflog                           # История всех действий  
git diff HEAD~1                      # Разница с предыдущим коммитом  
git show <hash>                      # Содержимое коммита  
git status                            # Текущее состояние
```