pwd — просмотр вашего текущего местоположения;

ls — список папок и файлов в текущей директории, где была выполнена команда;

ls -a — список открытых и скрытых папок и файлов в текущей директории, где была выполнена команда;

cd ~ — переход в домашнюю директорию текущего пользователя;

cd .. — переход на один уровень вверх в иерархии файловой системы;

cd folder\_name — переход в выбранную папку;

mkdir folder\_name — создать папку с указанным именем.

git config --global user.name "Name Surname"

git config --global user.email "your@email"

git help command\_name

**#Просмотр изменений в компактном виде**

git status -s

git add file\_name

git add file\_name\_1 file\_name\_2 file\_name\_3

git add .

**#Добавляем файлы в историю**

git commit -m "Commit message"

**#Создать новый коммит, минуя индекс.**

git commit -am "Commit message"

**#Если ошиблись в сообщении коммита. Можно ввести опцию --amend и перезаписать сообщение последнего коммита**

git commit --amend -m "New commit message" – редактирует последний коммит, **не создает новый!!!**

**#Сделали коммит слишком рано, например, забыв добавить какие-то файлы (откроет текстовый редактор сообщение оставляем без изм).**

git commit --amend

**#История коммитов**

git log

**# Запрос на вывод истории коммитов в одну строку**

git log –oneline

**## Запрос на просмотр коммита с хешем abc12345**

git show abc12345

**# Смотрим разницу между последним коммитом и текущим состоянием репозитория (что изменили, но пока не проиндексировали)**

git diff

**# Разница между последним коммитом и текущим состоянием файла**

git diff file\_name

**# Разница между последним коммитом и коммитом с указанным хешем**

git diff commit\_hash

**# Разница между последним коммитом и отслеживаемым состоянием репозитория**

git diff –staged

**# Сравнить любые два коммита**

git diff <commit1> <commit2>

**# разница между ветками main и develop**

git diff main develop

**#Любые локальные изменения, внесённые в этот файл, исчезнут**

git restore file\_name (для файлов которые не добавлены в индекс)

**# Вернуть все файлы из индекса к состоянию последнего коммита**

git restore --staged

**# Вернуть указанный файл из индекса к состоянию последнего коммита**

git restore --staged file\_name

**# Удалить файл из индекса и рабочей директории**

git rm file\_name

**#Переименовать файл в Git**

git mv file\_from file\_to

**# Удалить файл из индекса и оставить в папке на компьютере**

git rm --cached file\_name

**# Откатываемся и переводим последующие коммиты в индекс**

git reset --soft commit\_hash

**# Откатываемся и переводим последующие коммиты в неотслеживаемую зону**

git reset --mixed commit\_hash

**# Откатываемся и удаляем все последующие коммиты**

git reset --hard commit\_hash

**#Cоздание новой ветки**

git branch branch\_name

**#Ветку с неудачным названием можно переименовать**

git branch –move или -m master main

**#просмотр списка веток**

git branch

**#Посмотреть коммит на каждой ветке**

git branch -v

**#Переключение между ветками**

git checkout branch\_name

**#создать новую ветку и сразу в неё перейти:**

git checkout -b branch\_name

**#Команда git switch безопасней** и больше подходит новичкам. Перед каждым переключением она автоматически проверяет рабочую директорию и не срабатывает, если переход на выбранную ветку может привести к потере данных.

git switch branch\_name

**#удаление ветки**

git branch -d <branch\_name>:

**# Сливаем изменения из второстепенной ветки в основную**

git merge secondary\_branch

**# Привязка локального репозитория** к удалённому на GitHub (присвоить имя при необходимости, можно получить изменения к себе на локальный компьютер через git fetch name)

git remote add name origin [git@github.com:ваш\_профиль/ваш\_репозиторий.git](mailto:git@github.com:ваш_профиль/ваш_репозиторий.git)

**# Запрашиваем список удалённых** репозиториев, которые связаны с локальным

git remote

**# Запрос списка удалённых репозиториев с URL-адресами**

git remote –v

**# Команда для первой загрузки изменений** в удалённый репозиторий: текущая ветка будет связана с веткой main в удалённом репозитории git push -u origin main

**# Команда для второй и последующих загрузок изменений в удалённый репозиторий**

git push

**#отправить коммиты принудительно (осторожно: может переписать историю у других)**

git push --force

**#Скачиваем изменения из удалённого репозитория** и добавляем их в локальную ветку (+ происходит слияние merge или rebase в зависимости от настроек)

git pull

**#Скачиваем изменения из удалённого репозитория** и добавляем их в локальную ветку (без слияния изменений)

git fetch

**#отменить эффект выбранного коммита (создаёт новый коммит-отмену)**

git revert <commit>

git revert f3c1a2e

**# удалить неотслеживаемые файлы/папки (очень осторожно!)**

git clean -n # показать, что будет удалено

git clean -f # удалить неотслеживаемые файлы

git clean -fd # + папки

**#это команда Git, позволяет «перенести» один или несколько отдельных коммитов** из одной ветки в другую. В отличие от merge или rebase, cherry-pick не связывает всю ветку, а выбирает только конкретные коммиты, которые вы укажете.

git cherry-pick a1b2c3d

git cherry-pick a1b2c3d^..d4e5f6g

применит все коммиты с `a1b2c3d` до `d4e5f6g` включительно.

-e — открывает редактор для редактирования сообщения коммита

-x — добавляет строку в сообщение: «cherry picked from commit

- --no-commit (-n) — применяет изменения, но не создаёт коммит, позволяет внести правки перед коммитом

Примеры:

git cherry-pick -x a1b2c3d

git cherry-pick -n a1b2c3d

Когда применять cherry-pick

- Нужно перенести конкретный багфикс или небольшое улучшение между ветками без полного слияния.

- Хочется избежать попадания лишних изменений из других коммитов ветки-источника.

Конфликты могут возникать при слиянии веток (merge), переносе изменений (cherry-pick), перемотке истории (rebase) и даже при pull, если в удалённой и локальной версиях файла одни и те же строки были изменены по-разному.

**1. Как узнать, что возник конфликт?**

- После команды git merge, git cherry-pick, git rebase или git pull, Git может остановить выполнение и показать сообщение, что есть конфликты.

- В команде git status будет написано «Unmerged paths» или «both modified», а также список файлов с конфликтами.

**2. Найти конфликтные файлы**

git status

Git выведет список файлов с конфликтами (обычно помечены как both modified).

**3. Открыть конфликтный файл**

В файле вместо обычного текста появятся специальные метки:

<<<<<<< HEAD

Текст из вашей текущей ветки

=======

Текст из сливаемой/переносимой ветки

>>>>>>> feature-branch

- Всё, что между <<<<<<< HEAD и ======= — это ваш вариант (локальный).

- Между ======= и >>>>>>> имя-ветки или хеш — это другой вариант, который пытались применить.

**4. Разрешить конфликт вручную**

- Откройте файл в любом редакторе

- Найдите эти метки <<<<<<<, =======, >>>>>>>.

- Оставьте нужный вариант, удалите все метки. Можно также отредактировать текст, объединив оба варианта, если это нужно.

Пример:

<<<<<<< HEAD

print("Добро пожаловать!")

=======

print("Welcome!")

>>>>>>> feature-branch

Решение (оставляем один вариант или объединяем):

print("Добро пожаловать!")

print("Welcome!")

**5. Отметить конфликт как решённый**

После ручного редактирования каждого конфликтного файла сохраните его и выполните:

git add <имя\_файла>

Повторите для всех файлов с конфликтами.

**6. Завершить операцию**

- Если это был **git merge или git cherry-pick**, после добавления файла(ов) завершите процесс коммитом (если не был создан автоматически):

git commit

- Если это был **git rebase,** продолжите процесс:

git rebase --continue

- Если использовали **cherry-pick** и решали конфликт:

git cherry-pick --continue

**7. Если что-то пошло не так — отменить процесс**

- Для merge:

git merge --abort

- Для rebase:

git rebase --abort

- Для cherry-pick:

git cherry-pick --abort

**8. Проверить результат**

- После завершения процесса рекомендуется проверить свои изменения и убедиться, что код работает правильно.

- Затем можно продолжить работу: пушить изменения, выполнять новые коммиты и т.д.

Советы для упрощения работы с конфликтами

- Используйте современные редакторы кода (VSCode, WebStorm, PyCharm): они визуально подсвечивают и часто предлагают удобный интерфейс для выбора вариантов.

- Команда `git diff` помогает видеть различия между вашей и «чужой» версией файла.

- Перед слиянием актуализируйте свою ветку командой `git fetch` и `git pull`, чтобы уменьшить шанс конфликтов.