1. **GitHub** — один з найбільших [веб-сервісів](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0) для спільної розробки [програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F). Існують безкоштовні та платні тарифні плани користування сайтом.

# **Основные команды**

**git push** — вливание локальных изменений в удаленный репозиторий. git pull — вливание изменений из удаленного репозитория в локальный. Обмен данными обычно происходит с использованием протокола SSH.

**git add**

Команда git add добавляет содержимое рабочей директории в индекс (staging area) для последующего коммита. По умолчанию git commit использует лишь этот индекс, так что вы можете использовать git add для сборки слепка вашего следующего коммита.

**git status**

Команда git status показывает состояния файлов в рабочей директории и индексе: какие файлы изменены, но не добавлены в индекс; какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов.

**git diff**

Команда git diff используется для вычисления разницы между любыми двумя Git деревьями. Это может быть разница между вашей рабочей директорией и индексом (собственно git diff), разница между индексом и последним коммитом (git diff --staged), или между любыми двумя коммитами (git diff master branchB).

**git difftool**

Команда git difftool просто запускает внешнюю утилиту сравнения для показа различий в двух деревьях, на случай если вы хотите использовать что-либо отличное от встроенного просмотрщика git diff.

**git commit**

Команда git commit берёт все данные, добавленные в индекс с помощью git add, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок.

**git reset**

Команда git reset, как можно догадаться из названия, используется в основном для отмены изменений. Она изменяет указатель HEAD и, опционально, состояние индекса. Также эта команда может изменить файлы в рабочей директории при использовании параметра --hard, что может привести к потере наработок при неправильном использовании, так что убедитесь в серьёзности своих намерений прежде чем использовать его.

**git rm**

Команда git rm используется в Git для удаления файлов из индекса и рабочей директории. Она похожа на git add с тем лишь исключением, что она удаляет, а не добавляет файлы для следующего коммита.

**git mv**

Команда git mv — это всего лишь удобный способ переместить файл, а затем выполнить git addдля нового файла и git rm для старого.

**git clean**

Команда git clean используется для удаления мусора из рабочей директории. Это могут быть результаты сборки проекта или файлы конфликтов слияний.

# **Команди Git - Налаштування та конфігурація**

### git config

Git має типовий спосіб, як робити сотні речей. Для більшості з них, ви можете сказати Git працювати типово інакше, або встановити свої вподобання. Це включає все: від надання Git вашого імені до визначення кольорів у терміналі, чи який редактор використовувати. Існує декілька файлів, які ця команда читає та пише, щоб ви могли встановлювати значення як глобально, так і для окремих сховищ.

### git help

Команда git help призначена для відображення документації, що постачається разом з Git для кожної команди. Хоча ми даємо деякий огляд більшості з більш поширених команд у цьому додатку, для повного списку всіх можливих опцій кожної команди, ви завжди можете виконати git help <команда>.

***Всі команди:***

**Создать новый репозиторий**

git init # создать новый проект в текущей папке

git init folder-name # создать новый проект в указанной папке

git status - просмотр текущих изменений

**Клонирование репозитория**

git clone git@github.com:nicothin/web-design.git # клонировать удаленный репозиторий в одноименную папку

git clone git@github.com:nicothin/web-design.git foldername # клонировать удаленный репозиторий в папку «foldername»

git clone git@github.com:nicothin/web-design.git . # клонировать репозиторий в текущую папку

**Добавление файлов к отслеживанию, индексация отслеживаемых**

git add text.txt # добавить к отслеживанию этот существующий файл

git add . # добавить к отслеживанию все новые файлы из текущей папки и её подпапок, индексировать отслеживаемые файлы

git add -i # запуск оболочки интерактивного индексирования для добавления в индекс только выбранных файлов (см. [git-scm.com](http://git-scm.com/book/ru/v1/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-Git-%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5-%D0%B8%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))

git add -p # поочередный просмотр файлов с показом изменений и задаваемым вопросом об отслеживании/индексировании (удобно для добавления в коммит только каких-то конкретных файлов)

**Убирание файла, папки из отслеживания**

git rm --cached readme.txt # удалить файл из отслеживаемых (файл останется на месте)

git rm --cached -r folder # удалить папку из отслеживаемых (папка останется на месте)

git clean - Команда git clean используется для удаления мусора из рабочей директории. Это могут быть результаты сборки проекта или файлы конфликтов слияний.

**Отмена индексации**

git reset HEAD # убрать из индекса все индексированные файлы

git reset HEAD text.txt # убрать из индекса указанный файл

**Просмотр изменений**

git diff # посмотреть непроиндексированные изменения (если есть, иначе ничего не выведет)

git diff --staged # посмотреть проиндексированные изменения (если есть, иначе ничего не выведет)

git difftool

Команда git difftool просто запускает внешнюю утилиту сравнения для показа различий в двух деревьях, на случай если вы хотите использовать что-либо отличное от встроенного просмотрщика git diff.

**Отмена изменений**

git checkout -- text.txt # ОПАСНО: отменить все изменения, внесенные в отслеживаемый файл со времени предыдущего коммита (файл не добавлен в индекс)

git checkout -- . # ОПАСНО: отменить изменения во всех непроиндексированных отслеживаемых файлах

git checkout text.txt # ОПАСНО: отменить изменения в непроиндексированном файле

**Коммиты**

git commit -m "Name of commit" # закоммитить отслеживаемые индексированные файлы (указано название коммита)

git commit -a -m "Name of commit" # закоммитить отслеживаемые индексированные файлы (указано название коммита, не требует git add, не добавит в коммит неотслеживаемые файлы)

git commit # закоммитить отслеживаемые индексированные файлы (откроется редактор для введения названия коммита)

git commit --amend # изменить последний коммит (Insert — режим ввода, : — командный режим; в командном режиме: :wq — сохранить и выйти)

git commit --amend -m "Новое название" # переименовать последний коммит (только если ещё не был отправлен в удалённый репозиторий)

**Отмена коммитов**

git revert HEAD --no-edit # создать новый коммит, отменяющий изменения последнего коммита без запуска редактора сообщения

git revert b9533bb --no-edit # создать новый коммит, отменяющий изменения указанного (b9533bb) коммита без запуска редактора сообщения (указаны первые 7 символов хеша коммита)

git reset --hard 75e2d51 # вернуть репозиторий в состояние коммита с указанным хешем ОПАСНО! пропадет вся работа, сделанная после этого коммита

**Временно переключиться на другой коммит**

git checkout b9533bb # временно переключиться на коммит с указанным хешем

git checkout master # вернуться к последнему коммиту в указанной ветке

**Переключиться на другой коммит и продолжить работу с него**

Потребуется создание новой ветки, начинающейся с указанного коммита.

git checkout -b new-branch 5589877 # создать ветку new-branch, начинающуюся с коммита 5589877

**Удаление файла (просто удалить отслеживаемый файл из папки недостаточно, нужно сделать его неотслеживаемым и отправить коммит)**

git rm text.txt # удалить из отслеживаемых неиндексированный файл (файл будет удален из папки)

git rm -f text.txt # удалить из отслеживаемых индексированный файл (файл будет удален из папки)

git rm -r log/ # удалить из отслеживаемых всё содержимое папки log/ (папка будет удалена)

git rm ind\* # удалить из отслеживаемых все файлы с именем, начинающимся на «ind» в текущей папке (файлы будут удалены из папки)

git rm --cached readme.txt # удалить из отслеживаемых индексированный файл (файл останется на месте)

**Перемещение/переименование файлов (Git не отслеживает перемещения/переименование, но пытается его угадать)**

git mv text.txt test\_new.txt # переименовать файл «text.txt» в «test\_new.txt»

git mv readme\_new.md folder/ # переместить файл readme\_new.md в папку folder/ (должна существовать)

**История изменений**

git log -p index.html # показать историю изменений файла index.html (выход из длинного лога: Q)

git log -p -5 index.html # показать историю изменений файла index.html (последние 5 коммитов, выход из длинного лога: Q)

git log -2 # показать последние 2 коммита

git log -2 --stat # показать последние 2 коммита и статистику внесенных ими изменений

git log -p -22 # показать последние 22 коммита и внесенную ими разницу на уровне строк (выход из длинного лога: Q)

git log --pretty=format:"%h - %an, %ar : %s" -4 # показать последние 4 коммита с форматированием выводимых данных

git log --graph -10 # показать последние 10 коммитов с ASCII-представлением ветвления

git log --since=2.weeks # показать коммиты за последние 2 недели

git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short # мой формат вывода, висящий на алиасе оболочки

git log master..branch\_99 # показать коммиты из ветки branch\_99, которые не влиты в master

git log branch\_99..master # показать коммиты из ветки master, которые не влиты в branch\_99

git show 60d6582 # показать изменения из коммита с указанным хешем

git show HEAD^ # показать данные о предыдущем коммите

**Ветки**

git branch # показать список веток

git branch -v # показать список веток и последний коммит в каждой

git branch new\_branch # создать новую ветку с указанным именем

git checkout new\_branch # перейти в указанную ветку

git checkout -b new\_branch # создать новую ветку с указанным именем и перейти в неё

git merge hotfix # влить в ветку, в которой находимся, данные из ветки hotfix

git branch -d hotfix # удалить ветку hotfix (если её изменения уже влиты в главную ветку)

git branch --merged # показать ветки, уже слитые с активной (их можно удалять)

git branch --no-merged # показать ветки, не слитые с активной

git branch -a # показать все имеющиеся ветки (в т.ч. на удаленных репозиториях)

git branch -m old\_branch\_name new\_branch\_name # переименовать локально ветку old\_branch\_name в new\_branch\_name

git branch -m new\_branch\_name # переименовать локально ТЕКУЩУЮ ветку в new\_branch\_name

git push origin :old\_branch\_name new\_branch\_name # применить переименование в удаленном репозитории

git branch --unset-upstream # завершить процесс переименования

**Удалённые репозитории**

git remote -v # показать список удалённых репозиториев, связанных с этим

git remote remove origin # убрать привязку удалённого репозитория с сокр. именем origin

git remote add origin git@github.com:nicothin/test.git # добавить удалённый репозиторий (с сокр. именем origin) с указанным URL

git remote rm origin # удалить привязку удалённого репозитория

git remote show origin # получить данные об удалённом репозитории с сокращенным именем origin

git fetch origin # скачать все ветки с удаленного репозитория (с сокр. именем origin), но не сливать со своими ветками

git fetch origin master # то же, но скачивается только указанная ветка

git checkout origin/github\_branch # посмотреть ветку, скачанную с удалённого репозитория (локальной редактируемой копии не создаётся! если нужно редактировать, придётся влить)

git checkout --track origin/github\_branch # создать локальную ветку github\_branch (данные взять из удалённого репозитория с сокр. именем origin, ветка github\_branch) и переключиться на неё

git push origin master # отправить в удалённый репозиторий (с сокр. именем origin) данные своей ветки master

git pull origin # влить изменения с удалённого репозитория (все ветки)

git pull origin master # влить изменения с удалённого репозитория (только указанная ветка)

**3. Приклад застосування**

### Начало работы

Создание нового репозитория, первый коммит, привязка удалённого репозитория с gthub.com, отправка изменений в удалённый репозиторий.

# указана последовательность действий:

git init # инициируем гит в этой папке

touch readme.md # создаем файл readme.md

git add readme.md # делаем этот файл отслеживаемым

git commit -m "Первый коммит" # создаем первый коммит с вменяемым названием

git remote add origin « nicothin/test.git # добавляем предварительно созданный пустой удаленный репозиторий

git push origin master # отправляем данные из локального репозитория в удаленный (в ветку master)

### Обычный рабочий процесс

Создание нового репозитория на github.com, клонирование к себе, работа, периодическая «синхронизация с github.com».

# указана последовательность действий:

# создаём на github.com репозиторий, копируем его URL клонирования (SSH)

# в консоли попадаем в свою папку для всех проектов

git clone АДРЕС\_РЕПОЗИТОРИЯ ПАПКА\_ПРОЕКТА # клонируем удалённый репозиторий к себе на компьютер (если не указать ПАПУ\_ПРОЕКТА, будет создана папка, совпадающая по имени с названием репозитория)

cd ПАПКА\_ПРОЕКТА # переходим в папку проекта (указана команда для git bash)

# редактируем файлы, добавляем файлы и/или папки (если удаляем файлы — см. секцию про удаление файлов)

git add . # делаем все новые файлы в этой папке отслеживаемыми и готовыми к коммиту

git commit -m "НАЗВАНИЕ\_КОММИТА" # создаем коммит с вменяемым названием

git push origin master # отправляем данные из локального репозитория в удаленный (в ветку master)

# снова вносим какие-то изменения (если удаляем файлы — см. секцию про удаление файлов)

# возвращаемся к шагу с git add . и проходим цикл заново