**Связывания локального и удаленного репозитория**

git clone <URL репозитория> [<название локальной папки>]

Создает локальную копию удаленного репозитория Git со всеми связями. После этого можно сразу с ним работать.

<URL репозитория> - это URL-адрес удаленного репозитория Git. <название локальной папки> (необязательный параметр) - это название локальной папки, в которую будет склонирован репозиторий. Если <название локальной папки> не указан, Git автоматически создаст папку с названием репозитория.

git remote add origin <url>

Создает первичную настройку для связки репозиториев. Синхронизации при этом никакой не происходит.

origin – это вариант имени удаленного репозитория для использования в командах локального репозитория (чтобы не писать все время url), можно вместо origin использовать любое другое имя.

git push –u orign <branch>

Предназначена для первичной отправки и одновременного связывания **текущей ветки** с веткой <branch> в удаленном репозитории.

orign – это имя удаленного репозитория.

branch – это имя ветки в **удаленном репозитории**, с которой нужно связаться.

git push

Последующяя **отправка** изменений (комитов) **текущей локальной** ветки в удаленный репозиторий. Изменения будут загружаться в ту ветку удаленного репозитория, с которой связали текущую.

git pull

Последующее **получение** изменений (комитов) от удаленного репозитория в **текущую локальную** ветку. Изменения будут загружаться из той ветки удаленного репозитория, с которой связали текущую.

git remote

Просмотр сделанных связей **текущей ветки** с удаленным(и) репозиторием.

git remote -v

Просмотр сделанных связей **текущей ветки** с удаленным(и) репозиторием.

git fetch

Аналогична команде git **pull**, которая только загружает новые объекты, но **не применяет изменения локально в рабочей области**.

git branch -vv

Предназначена для отображения текущей локальной ветки с веткой на удаленном репозитории.

**Текущая работа с репозиторием**

git add

Добавить изменения из рабочей области в индекс

git commit

Добавить новую версию данных (commit) в репозиторий

git push

Отправить новые commit в удаленный репозиторий (GitHub)

git pull

Получить новые commit из удаленного репозитория (GitHub)

git log

Просмотр коммитов.

# Шпаргалка по Git командам

**1. Команда git clone**

Команда **git clone** используется для создания локальной копии удаленного репозитория Git. Это позволяет вам получить копию проекта на вашем локальном компьютере и начать работу с ним.

Синтаксис команды git clone:

**git clone <URL репозитория> [<название локальной папки>]**

*где <URL репозитория> - это URL-адрес удаленного репозитория Git, а <название локальной папки> (необязательный параметр) - это название локальной папки, в которую будет склонирован репозиторий. Если <название локальной папки> не указан, Git автоматически создаст папку с названием репозитория.*

Пример использования команды git clone:

**git clone https://github.com/username/repo.git**

*Эта команда создаст локальную копию удаленного репозитория repo, который находится в аккаунте username на GitHub, в папке с названием repo.*

Если вы хотите указать свое собственное название для локальной папки, вы можете добавить его после URL-адреса репозитория:

**git clone https://github.com/username/repo.git myproject**

*Эта команда создаст локальную копию удаленного репозитория repo в папке myproject. Если папка myproject уже существует, Git выведет сообщение об ошибке и не склонирует репозиторий.*

**2. Команда git init**

Команда **git init** используется для создания нового локального репозитория Git. После выполнения этой команды Git создает в текущей директории новую поддиректорию с именем .git, в которой хранятся все файлы Git, необходимые для управления версиями в этом репозитории.

Синтаксис команды git init:

**git init [--bare] [<directory>]**

***--bare****- определяет репозиторий как "голый", т.е. без рабочей директории. Это используется для создания удаленного репозитория.*

*<directory> - указывает директорию, в которой создается локальный репозиторий. Если параметр не указан, репозиторий создается в текущей директории*.

Пример использования команды git init:

**$ cd my\_project  
$ git init**  
Initialized empty Git repository in /path/to/my\_project/.git/

*Эта команда создает новый локальный репозиторий Git в текущей директории my\_project. Git создает поддиректорию .git и инициализирует ее файлами, необходимыми для управления версиями.*

Флаг **--bare** используется для создания "голого" удаленного репозитория. Например:

**$ mkdir my\_repo.git  
$ cd my\_repo.git  
$ git init --bare**  
Initialized empty Git repository in /path/to/my\_repo.git/

*Эта команда создает новый удаленный репозиторий Git в директории my\_repo.git. Флаг****--bare****указывает, что это "голый" репозиторий, т.е. без рабочей директории.*

Команда git init также поддерживает другие опции и флаги, которые можно использовать в различных сценариях. Например, флаг **--template** позволяет указать кастомный шаблон инициализации, а опция **--separate-git-dir** позволяет разместить репозиторий в другой директории. Список всех опций и флагов можно посмотреть в документации Git.

**3. Команда git add .**

Команда **git add .** используется для добавления изменений в индекс Git. Индекс - это промежуточный слой между рабочей директорией (где находятся ваши файлы) и репозиторием Git (где сохраняются изменения). Когда вы делаете изменения в файлах в рабочей директории, они не автоматически добавляются в индекс. Для того, чтобы добавить изменения в индекс, необходимо использовать команду git add.

Синтаксис команды git add . :

**git add .**

Эта команда добавляет все измененные файлы в рабочей директории в индекс.

Пример использования команды git add . :

Создайте новый файл с названием file.txt в вашей рабочей директории и добавьте в него какой-то текст.

Выполните команду git status, чтобы увидеть изменения в рабочей директории:

**$ git status**  
On branch master  
Untracked files:  
(use "git add <file>..." to include in what will be committed)  
file.txt  
  
nothing added to commit but untracked files present (use "git add" to track)

Вы видите, что файл file.txt еще не отслеживается Git.

Выполните команду git add ., чтобы добавить все измененные файлы в рабочей директории в индекс:

**$ git add .**

Снова выполните команду git status, чтобы увидеть изменения в индексе:

**$ git status**  
On branch master  
Changes to be committed:  
(use "git restore --staged <file>..." to unstage)  
new file: file.txt

Вы видите, что файл file.txt был добавлен в индекс.

Выполните команду git commit, чтобы сохранить изменения в репозитории Git:

**$ git commit -m "Added file.txt"**  
[master 0e3b0a7] Added file.txt  
1 file changed, 1 insertion(+)  
create mode 100644 file.txt

Вы видите, что файл file.txt был успешно добавлен в репозиторий Git.

Команда git add . очень полезна, когда вам нужно добавить много файлов в индекс одновременно. Однако, будьте осторожны, когда используете эту команду, особенно если вы работаете в большом проекте с множеством файлов. В некоторых случаях вам может потребоваться выбирать файлы для добавления в индекс более тщательно, чтобы избежать случайного добавления ненужных файлов.

**4. Команда git commit**

Команда **git commit**используется для сохранения изменений, сделанных в вашем локальном репозитории, в истории коммитов. Каждый коммит в Git имеет уникальный идентификатор, дату и время коммита, имя автора, электронную почту и сообщение, описывающее изменения.

Синтаксис команды git commit:

**git commit -m "<сообщение коммита>"**

*Где -m означает "message" (сообщение) и <сообщение коммита> - это краткое описание изменений, сделанных в коммите.*

Пример использования команды git commit:

Добавьте изменения в индекс с помощью команды git add (например, git add filename.txt или git add . для добавления всех измененных файлов в индекс). Сделайте коммит с помощью команды git commit и передайте сообщение коммита в кавычках, чтобы описать, что было изменено. Например:

**git commit -m "Добавлен новый файл README.md"**

После выполнения команды, Git сохранит изменения в истории коммитов, и сообщение коммита будет отображаться в списке коммитов.  
Важно отметить, что каждый коммит должен иметь уникальное сообщение, которое описывает изменения в нем. Это помогает вам и другим членам команды понимать, что было изменено и почему.

Кроме того, команду git commit можно дополнительно настраивать, используя различные параметры, например, -a (автоматически добавляет все измененные файлы в индекс) или --amend (используется для изменения последнего коммита). Для получения подробной информации о доступных параметрах команды git commit можно обратиться к документации Git.

**5. Команда git push**

Команда **git push** используется для отправки изменений из вашего локального репозитория в удаленный репозиторий Git. Это позволяет обновить содержимое удаленного репозитория на основе ваших локальных изменений.

Синтаксис команды git push:

**git push <remote> <branch>**

*Где <remote> - это имя удаленного репозитория, куда вы хотите отправить изменения, а <branch> - это название ветки, которую вы хотите отправить.*

Пример использования команды git push:

**git push origin main**

*Эта команда отправляет изменения из вашей локальной ветки main в удаленный репозиторий с именем origin.*

Помимо базового синтаксиса, команда git push имеет несколько флагов, которые можно использовать для дополнительной настройки:

**-u** или **--set-upstream**- устанавливает отслеживание для ветки, что позволяет вам использовать git push и git pull без указания имени удаленного репозитория и названия ветки;  
**-f** или **--force** - заставляет Git принудительно заменить удаленную ветку измененной локальной веткой, даже если это приведет к потере данных;  
**-n** или **--dry-run**- позволяет вам протестировать команду git push, не отправляя реальных изменений в удаленный репозиторий;  
**-v** или **--verbose** - выводит дополнительную информацию о процессе отправки изменений.

Пример использования команды git push с флагом:

**git push -u origin main**

*Эта команда отправляет изменения из вашей локальной ветки main в удаленный репозиторий с именем origin и устанавливает отслеживание для этой ветки.*

**6. Команда git status**

Команда **git status** используется для получения информации о текущем состоянии вашего рабочего пространства Git. Она показывает, какие файлы были изменены, какие из них были добавлены в индекс, и какие из них готовы к коммиту. Кроме того, команда "git status" сообщает о текущей ветке и другой полезной информации.

Синтаксис команды "git status":

**git status [-s] [--long] [--branch] [--porcelain] [--ignore-submodules[=<when>]]**

Основные флаги:

"**-s**" или "**--short**": показывает краткую информацию о состоянии файлов в формате "git diff --shortstat".  
"**--long**": показывает длинный формат состояния файлов, включая информацию о последнем коммите для каждого файла.  
"**--branch**": показывает текущую ветку и ее состояние.  
"**--porcelain**": показывает состояние файлов в машинно-читаемом формате, что полезно для автоматизации.  
"**--ignore-submodules**": позволяет игнорировать изменения в подмодулях.

Пример использования команды "git status":

**$ git status**  
  
На ветке main  
Ваша ветка опережает «origin/main» на 1 коммит.  
  (используйте «git push», чтобы опубликовать локальные коммиты)  
  
изменения, которые будут включены в коммит:  
  (используйте «git restore --staged <file>...», чтобы убрать из индекса)  
  
        изменён:   README.md  
  
изменения, которые не проиндексированы для коммита:  
  (используйте «git add <file>...», чтобы проиндексировать)  
  
        изменён:   index.html  
  
нет изменений добавленных для коммита (используйте «git add» и/или «git commit -a»)

*Эта команда показывает, что находитесь на ветке "main", ваша локальная ветка опережает ветку "origin/main" на один коммит, и что были внесены изменения в файлы "README.md" и "index.html". Файл "README.md" был проиндексирован и готов к коммиту, а файл "index.html" не был проиндексирован. Для того, чтобы проиндексировать файл "index.html", нужно использовать команду "git add index.html".*

**7. Команда git diff**

Команда **git diff** используется для просмотра различий между версиями файлов в репозитории. Она показывает, какие изменения были сделаны в файлах с момента последнего коммита.

Основной синтаксис команды git diff выглядит так:

**git diff [<опции>] [<источник>] [<цель>]**

Где:

*<источник> - определяет начальную точку изменений для отображения различий.  
<цель> - определяет конечную точку изменений для отображения различий.  
<опции> - дополнительные флаги, изменяющие поведение команды.*

Основные флаги git diff:

**--cached**- сравнивает изменения между вашим индексом (т.е. тем, что будет в следующем коммите) и последним коммитом.  
**--stat** - показывает статистику изменений для каждого файла.  
**--color**- отображает различия с использованием цветов для лучшей читаемости.

Примеры использования:

**git diff** - простой запуск команды покажет различия между вашей рабочей копией и последним коммитом.  
**git diff --cached**- покажет различия между вашим индексом и последним коммитом.  
**git diff HEAD** - покажет различия между вашей рабочей копией и последним коммитом.  
**git diff HEAD~2 HEAD** - покажет различия между двумя последними коммитами.  
**git diff --stat**- покажет статистику изменений для каждого файла.  
**git diff --color**- покажет различия с использованием цветов.

Кроме того, можно использовать git diff для сравнения любых двух коммитов, веток или тэгов в репозитории. Например, git diff branch1..branch2 покажет различия между двумя ветками branch1 и branch2.

**8. Команда git log**

Команда **git log** используется для просмотра истории коммитов в репозитории Git. Она отображает список коммитов в обратном хронологическом порядке, начиная с последнего.

Пример использования команды git log:

**git log**

Эта команда показывает список всех коммитов в репозитории, от самых последних до самых старых. Каждый коммит включает SHA-1 хеш, автора коммита, дату и время коммита, и комментарий.

Основные флаги команды git log:

**--oneline**: показывает каждый коммит в одной строке, содержащей только его SHA-1 хеш и комментарий.

**--graph**: показывает историю коммитов в виде графа, что позволяет легко визуализировать разветвленную историю ветвлений и слияний.

**--author=<имя>**: показывает только коммиты, сделанные указанным автором.

**--since=<дата>**: показывает только коммиты, сделанные после указанной даты.

**--until=<дата>**: показывает только коммиты, сделанные до указанной даты.

**-n <количество>**: показывает только указанное количество последних коммитов.

**--grep=<строка>**: показывает только коммиты, содержащие указанную строку в своих комментариях.

Пример использования флага --since:

**git log --since=2022-01-01**

*Эта команда показывает список всех коммитов, сделанных после 1 января 2022 года.*

Пример использования флага --author:

**git log --author="John Doe"**

*Эта команда показывает список всех коммитов, сделанных автором с именем "John Doe".*

Пример использования флага --grep:

**git log --grep="bug"**

*Эта команда показывает список всех коммитов, содержащих слово "bug" в своих комментариях.*

**9. Команда git checkout**

Команда **git checkout** в Git используется для переключения между ветками, проверки коммитов и отката изменений. Она позволяет переключаться между различными состояниями вашего репозитория.

Основные флаги:

**-b**: создать новую ветку и переключиться на нее. Например, git checkout -b new-branch создаст новую ветку с названием "new-branch" и переключится на нее.  
**-f**: принудительно перезаписывает локальные изменения, которые не были зафиксированы. Этот флаг используется только в крайних случаях.  
**-p**: позволяет просмотреть и выбрать конкретные изменения для восстановления.  
**<commit/branch>**: позволяет переключиться на определенный коммит или ветку.

Примеры использования:

Переключение на ветку:

**git checkout main**

*Эта команда переключает вас на ветку "main".*

Создание новой ветки и переключение на нее:

**git checkout -b new-branch**

*Эта команда создает новую ветку с названием "new-branch" и переключает вас на нее.*

Переключение на определенный коммит:

**git checkout 2a3e8c9**

*Эта команда переключает вас на коммит с указанным идентификатором.*

Отмена изменений в файле:

**git checkout myfile.txt**

*Эта команда отменяет изменения в файле "myfile.txt" и восстанавливает его до последней зафиксированной версии.*

Восстановление определенных изменений:

**git checkout -p myfile.txt**

*Эта команда позволяет вам просмотреть изменения в файле "myfile.txt" и выбрать, какие из них восстановить.*

**10. Команда git branch**

Команда **git branch**используется для просмотра, создания и удаления веток в репозитории Git. Ветка - это отдельная линия разработки, которая может включать в себя свой собственный набор коммитов.

Основные флаги команды git branch:

**-a** - показывает все ветки (включая удаленные)  
**-d**- удаляет ветку (данная команда удаляет только те ветки, которые были слиты в текущую ветку)  
**-D** - удаляет ветку без проверки, были ли все её изменения слиты с текущей веткой  
**-m** - переименовывает текущую ветку  
**-r** - показывает все удаленные ветки

Примеры использования команды git branch:

Просмотр списка веток:

**git branch**

Создание новой ветки:

**git branch new-branch**

Переименование текущей ветки:

**git branch -m new-branch-name**

Удаление ветки:

**git branch -d branch-to-delete**

Просмотр удаленных веток:

**git branch -r**

Команда git branch позволяет управлять ветками в репозитории и помогает организовать работу в команде. Она также помогает контролировать историю изменений в проекте.

**11. Команда git fetch**

Команда **git fetch** используется для получения изменений из удаленного репозитория Git, но не вносит изменения в локальную ветку. Эта команда позволяет вам получить информацию о ветках и коммитах, которых еще нет в локальном репозитории.

Синтаксис команды git fetch:

**git fetch [<remote>] [<refspec>...]**

Основные флаги:

**<remote>**: имя удаленного репозитория Git (например, origin). Если не указан, Git использует имя origin по умолчанию.  
**<refspec>**: имена веток и тегов, которые вы хотите получить из удаленного репозитория. Если не указан, Git получает все ветки и теги.

Пример использования команды git fetch:

**git fetch**

*Эта команда получает все изменения из удаленного репозитория Git, который связан с вашим локальным репозиторием. После выполнения этой команды вы можете выполнить команду git log origin/master для просмотра истории изменений в удаленной ветке master.*

**git fetch origin feature-branch**

*Эта команда получает все изменения из удаленной ветки feature-branch в удаленном репозитории Git, связанном с вашим локальным репозиторием. После выполнения этой команды вы можете выполнить команду git log origin/feature-branch для просмотра истории изменений в удаленной ветке feature-branch.*

**12. Команда git pull**

Команда **git pull**используется для получения изменений из удаленного репозитория и объединения их с вашей локальной веткой.

Основной синтаксис команды git pull выглядит следующим образом:

**git pull [<options>] [<repository> [<refspec>...]]**

Основные флаги git pull:

**--rebase**: выполняет перебазирование текущей ветки на самый новый коммит вместо создания слияния (merge). Рекомендуется использовать этот флаг, если вы хотите, чтобы история коммитов оставалась простой и понятной.  
**--no-rebase**: отменяет перебазирование и выполняет слияние вместо этого.  
**--no-commit**: предотвращает создание автоматического коммита после выполнения слияния. Это дает возможность внести дополнительные изменения перед фиксацией изменений.  
**--ff-only**: выполнить слияние только в том случае, если это можно сделать быстрым перемещением ветки вперед (fast-forward).  
**--no-ff**: выполнить слияние только как коммит слияния (merge commit), даже если это можно сделать быстрым перемещением ветки вперед.

***Примеры использования команды git pull:***

Простой git pull, который получает изменения из удаленной ветки и автоматически объединяет их с локальной веткой:

**git pull**

git pull с использованием флага --rebase, который перебазирует текущую ветку на самый новый коммит из удаленной ветки вместо создания слияния:

**git pull --rebase**

git pull с использованием флага --no-commit, который предотвращает создание автоматического коммита после выполнения слияния:

**git pull --no-commit**

git pull с использованием флага --ff-only, который выполнит слияние только в том случае, если это можно сделать быстрым перемещением ветки вперед (fast-forward):

**git pull --ff-only**

git pull с использованием флага --no-ff, который выполнит слияние только как коммит слияния (merge commit), даже если это можно сделать быстрым перемещением ветки вперед:

**git pull --no-ff**

**13. Команда git merge**

Команда **git merge** используется для объединения изменений из одной ветки в другую. Она применяется для слияния изменений, которые были внесены в отдельной ветке, обычно для объединения фиксации ошибок или добавления функциональности в проект.

Синтаксис команды git merge:

**git merge <имя ветки>**

*где <имя ветки> - это имя ветки, изменения из которой вы хотите объединить в текущую ветку.*

***Пример использования команды git merge:***

Сначала, вам нужно переключиться на ветку, в которую вы хотите объединить изменения:

**git checkout master**

Затем вы можете выполнить команду git merge, указав имя ветки, из которой вы хотите получить изменения:

**git merge feature-branch**

В этом примере мы объединяем изменения из ветки feature-branch в текущую ветку master.

Основные флаги команды git merge:

**--no-ff**: Отключает быстрое слияние, что создает коммит с объединением, даже если изменения можно было бы просто быстро применить.  
**--abort**: Отменяет текущее слияние и возвращает репозиторий в предыдущее состояние.

Пример использования флага --no-ff:

**git merge --no-ff feature-branch**

*В этом примере мы отключаем быстрое слияние и создаем новый коммит с объединением.*

Пример использования флага --abort:

**git merge --abort**

*В этом примере мы отменяем текущее слияние и возвращаем репозиторий в предыдущее состояние.*

**14. Команда git rebase**

Команда **git rebase**используется для перебазирования текущей ветки на другую ветку или на определенный коммит. Это означает, что изменения из исходной ветки будут применены поверх коммитов в целевой ветке или коммите.

Основные флаги:

**-i (или --interactive)**: запускает интерактивный режим, который позволяет изменять порядок, объединять или отбрасывать коммиты.  
**-m (или --merge)**: используется, когда нужно перебазировать ветку слияния.  
**--onto <branch>**: перебазирует текущую ветку на указанную ветку.

Примеры использования команды git rebase:

Перебазирование текущей ветки на другую ветку:

**git checkout feature-branch  
git rebase main**

*В этом примере текущая ветка feature-branch перебазируется на ветку main. Это означает, что изменения из ветки main будут применены поверх коммитов в ветке feature-branch.*

Перебазирование текущей ветки на определенный коммит:

**git checkout feature-branch  
git rebase abc123**

*В этом примере текущая ветка feature-branch перебазируется на коммит с хеш-кодом abc123. Это означает, что изменения из этого коммита и всех коммитов после него будут применены поверх коммитов в ветке feature-branch.*

Использование интерактивного режима:

**git checkout feature-branch  
git rebase -i main**

*В этом примере запускается интерактивный режим, который позволяет изменять порядок, объединять или отбрасывать коммиты при перебазировании ветки feature-branch на ветку main.*

Перебазирование ветки слияния:

**git checkout merge-branch  
git rebase -m main**

*В этом примере ветка merge-branch, являющаяся веткой слияния, перебазируется на ветку main. Опция -m используется для корректной обработки коммитов слияния.*

**15. Команда git revert**

Команда **git revert**используется для отмены определенного коммита и создания нового коммита, который отменяет изменения предыдущего коммита. Это делает команду безопасной для использования в общем репозитории, поскольку она не изменяет историю коммитов.

Синтаксис команды git revert:

**git revert <commit>**

*где <commit> - это хэш-идентификатор коммита, который нужно отменить.*

Пример использования команды git revert:

**$ git log --oneline**  
d3b3d43 Commit C  
786f7f8 Commit B  
12556fa Commit A  
  
**$ git revert d3b3d43**

*Эта команда создаст новый коммит, который отменяет изменения, внесенные коммитом с хэш-идентификатором d3b3d43.*

Флаги:

**--no-commit:** применить изменения, но не создавать новый коммит. Это позволяет вам проверить изменения перед тем, как закоммитить их.  
**-m parent-number:** используется, если коммит имеет несколько родительских коммитов (как в случае с объединением веток). Он указывает, какой из родительских коммитов следует использовать при выполнении операции отмены.

Пример использования команды git revert с флагом --no-commit:

**$ git revert --no-commit d3b3d43  
$ git status**  
On branch master  
Changes to be committed:  
  (use "git reset HEAD <file>..." to unstage)  
        modified:   file.txt

*Эта команда отменяет изменения, внесенные коммитом с хэш-идентификатором d3b3d43, и добавляет их в индекс. Но новый коммит не создается, пока не будет выполнена команда git commit.*