30 основных команд, которые сделают из вас мастера Git

1. [Как задать имя пользователя и адрес электронной почты](https://habr.com/ru/companies/ruvds/articles/599929/#1)
2. [Кэширование учётных данных](https://habr.com/ru/companies/ruvds/articles/599929/#2)
3. [Инициализация репозитория](https://habr.com/ru/companies/ruvds/articles/599929/#3)
4. [Добавление отдельных файлов или всех файлов в область подготовленных файлов](https://habr.com/ru/companies/ruvds/articles/599929/#4)
5. [Проверка статуса репозитория](https://habr.com/ru/companies/ruvds/articles/599929/#5)
6. [Внесение изменений однострочным сообщением или через редактор](https://habr.com/ru/companies/ruvds/articles/599929/#6)
7. [Просмотр истории коммитов с изменениями](https://habr.com/ru/companies/ruvds/articles/599929/#7)
8. [Просмотр заданного коммита](https://habr.com/ru/companies/ruvds/articles/599929/#8)
9. [Просмотр изменений до коммита](https://habr.com/ru/companies/ruvds/articles/599929/#9)
10. [Удаление отслеживаемых файлов из текущего рабочего дерева](https://habr.com/ru/companies/ruvds/articles/599929/#10)
11. [Переименование файлов](https://habr.com/ru/companies/ruvds/articles/599929/#11)
12. [Отмена подготовленных и неподготовленных изменений](https://habr.com/ru/companies/ruvds/articles/599929/#12)
13. [Изменение последнего коммита](https://habr.com/ru/companies/ruvds/articles/599929/#13)
14. [Откат последнего коммита](https://habr.com/ru/companies/ruvds/articles/599929/#14)
15. [Откат заданного коммита](https://habr.com/ru/companies/ruvds/articles/599929/#15)
16. [Создание новой ветки и переход в неё](https://habr.com/ru/companies/ruvds/articles/599929/#16)
17. [Просмотр списка веток](https://habr.com/ru/companies/ruvds/articles/599929/#17)
18. [Удаление ветки](https://habr.com/ru/companies/ruvds/articles/599929/#18)
19. [Слияние двух веток](https://habr.com/ru/companies/ruvds/articles/599929/#19)
20. [Отображение журнала фиксации в виде графика для текущей или всех веток](https://habr.com/ru/companies/ruvds/articles/599929/#20)
21. [Прекращение слияния при конфликте](https://habr.com/ru/companies/ruvds/articles/599929/#21)
22. [Добавление удалённого репозитория](https://habr.com/ru/companies/ruvds/articles/599929/#22)
23. [Просмотр удалённых URL-адресов](https://habr.com/ru/companies/ruvds/articles/599929/#23)
24. [Получение дополнительных сведений об удалённом репозитории](https://habr.com/ru/companies/ruvds/articles/599929/#24)
25. [Отправка изменений в удалённый репозиторий](https://habr.com/ru/companies/ruvds/articles/599929/#25)
26. [Получение изменений из удалённого репозитория](https://habr.com/ru/companies/ruvds/articles/599929/#26)
27. [Слияние удалённого репозитория с локальным](https://habr.com/ru/companies/ruvds/articles/599929/#27)
28. [Отправка новой ветки в удалённый репозиторий](https://habr.com/ru/companies/ruvds/articles/599929/#28)
29. [Удаление удалённой ветки](https://habr.com/ru/companies/ruvds/articles/599929/#29)
30. [Использование перебазирования](https://habr.com/ru/companies/ruvds/articles/599929/#30)

1. Как задать имя пользователя и адрес электронной почты

Имя пользователя нужно, чтобы привязывать коммиты к вашему имени. Это не то же самое, что имя пользователя учётной записи GitHub, с помощью которого выполняется вход в профиль на GitHub. Задать или изменить имя пользователя можно с помощью команды git config. Новое имя будет автоматически отображаться в последующих коммитах, отправленных на GitHub через командную строку. Если хотите скрыть своё реальное имя, можно использовать в качестве имени пользователя Git произвольный набор символов.

git config --global user.name "Tara Routray"

Кроме того, командой git config можно изменять адрес электронной почты, привязанный к вашим коммитам Git. Новый адрес электронной почты будет автоматически отображаться во всех дальнейших коммитах, поданных на GitHub через командную строку.

git config --global user.email "dev@tararoutray.com"

2. Кэширование учётных данных

Кэшировать учётные данные можно с помощью параметра config с флагом --global. Так вы избавитесь от необходимости вручную вводить имя пользователя и пароль при создании нового коммита.

git config --global credential.helper cache

3. Инициализация репозитория

Создать пустой репозиторий Git или вновь инициализировать существующий можно параметром init. При инициализации он создаст скрытую папку. В ней содержатся все объекты и ссылки, которые Git использует и создаёт в истории работы над проектом.

git init

4. Добавление отдельных файлов или всех файлов в область подготовленных файлов

Добавить отдельный файл в область подготовленных файлов можно параметром add с указанием имени файла. Просто замените somefile.js на актуальное имя.

git add somefile.js

Кроме того, можно добавить все файлы и папки в эту область, предоставив wildcard . вместо имени файла:

git add .

5. Проверка статуса репозитория

Просмотреть статус нужного репозитория можно по ключевому слову status: его действие распространяется на подготовленные, неподготовленные и неотслеживаемые файлы.

git status

6. Внесение изменений однострочным сообщением или через редактор

При создании коммита в репозитории можно добавить однострочное сообщение с помощью параметра commit с флагом -m. Само сообщение вводится непосредственно после флага, в кавычках.

git commit -m "Your short summary about the commit"

Также можно открыть текстовый редактор в терминале для написания полного сообщения коммита. Оно может состоять из нескольких строк текста, в котором подробно характеризуются изменения, внесённые в репозиторий.

git commit

7. Просмотр истории коммитов с изменениями

Просматривать изменения, внесённые в репозиторий, можно с помощью параметра log. Он отображает список последних коммитов в порядке выполнения. Кроме того, добавив флаг -p, вы можете подробно изучить изменения, внесённые в каждый файл.

git log -p

8. Просмотр заданного коммита

Просмотреть полный список изменений, внесённых конкретным коммитом, можно с помощью параметра show, указав идентификатор или хеш коммита. Значение хеша уникально для каждого коммита, созданного в вашем репозитории.

git show 1af17e73721dbe0c40011b82ed4bb1a7dbe3ce29

Также можно использовать сокращённый хеш.

git show 1af17e

9. Просмотр изменений до коммита

Можно просматривать список изменений, внесённых в репозиторий, используя параметр diff. По умолчанию отображаются только изменения, не подготовленные для фиксации.

git diff

Для просмотра подготовленных изменений необходимо добавить флаг --staged.

git diff --staged

Также можно указать имя файла как параметр и просмотреть изменения, внесённые только в этот файл.

git diff somefile.js

10. Удаление отслеживаемых файлов из текущего рабочего дерева

Удалять файлы из текущего рабочего дерева можно с помощью параметра rm. При этом файлы удаляются и из индекса.

git rm dirname/somefile.js

Можно также использовать маски файлов (например \*.js) для удаления всех файлов, соответствующих критерию.

git rm dirname/\*.html

11. Переименование файлов

Переименовать файл или папку можно параметром mv. Для него указывается источник source и назначение destination. Источник — реально существующий файл или папка, а назначение — существующая папка.

git mv dir1/somefile.js dir2

При выполнении команды файл или папка, указанные как источник, будут перемещены в папку назначения. Индекс будет обновлён соответственно, но изменения нужно записать.

12. Отмена подготовленных и неподготовленных изменений

Восстановить файлы рабочего дерева, не подготовленные к коммиту, можно параметром checkout. Для проведения операции требуется указать путь к файлу. Если путь не указан, параметр git checkout изменит указатель HEAD, чтобы задать указанную ветку как текущую.

git checkout somefile.js

Восстановить подготовленный файл рабочего дерева можно параметром reset. Потребуется указать путь к файлу, чтобы убрать его из области подготовленных файлов. При этом не будет производиться откат никаких изменений или модификаций — однако файл перейдёт в категорию не подготовленных к коммиту.

git reset HEAD somefile.js

Если нужно выполнить это действие для всех подготовленных файлов, путь к ним указывать не надо.

git reset HEAD

13. Изменение последнего коммита

Внести изменения в последний коммит можно параметром commit с флагом --amend. Например, вы записали изменения, внесённые в ряд файлов, и поняли, что допустили ошибку в сообщении коммита. В этом случае можете воспользоваться указанной командой, чтобы отредактировать сообщение предыдущего коммита, не изменяя его снимок.

git commit --amend -m "Updated message for the previous commit"

Также можно вносить изменения в файлы, отправленные ранее. Например, вы изменили несколько файлов в ряде папок и хотите их записать как единый снимок, но забыли добавить в коммит одну из папок. Чтобы исправить такую ошибку, достаточно подготовить для фиксации остальные файлы и папки и создать коммит с флагами --amend и --no-edit.

git add dir1

git commit

# Here you forgot to add dir2 to commit, you can execute the

following command to amend the other files and folders.

git add dir2

git commit --amend --no-edit

Флаг --no-edit позволит внести в коммит поправку без изменения сообщения коммита. В этом случае итоговый коммит заменит неполный, а выглядеть это будет так, как будто мы отправили изменения ко всем файлам в нужных папках как единый снимок.

**Внимание! Не изменяйте публичные коммиты.**  
  
С помощью amend прекрасно исправляются локальные коммиты, а исправления можно передать в общий репозиторий. Однако изменять коммиты, уже доступные другим пользователям, не следует. Помните, что изменённые коммиты являются совершенно новыми, а предыдущий коммит уже не будет доступен в текущей ветке. Последствия будут такими же, как при отмене изменений публичного снимка.

14. Откат последнего коммита

Откатить последний коммит можно с помощью параметра revert. Создастся новый коммит, содержащий обратные преобразования относительно предыдущего, и добавится к истории текущей ветки.

git revert HEAD

▍ Разница между revert и reset

Команда git revert отменяет изменения, записанные только одним коммитом. Она не откатывает проект к более раннему состоянию, удаляя все последующие коммиты, как это делает команда git reset.  
  
У команды revert есть два крупных преимущества по сравнению с reset. Во-первых, она не меняет историю проекта и производит операцию, безопасную для коммитов. Во-вторых, её объектом выступает конкретный коммит, созданный в любой момент истории, а git reset всегда берёт за точку отсчёта текущий коммит. К примеру, если нужно отменить старый коммит с помощью git reset, придётся удалить все коммиты, поданные после целевого, а затем выполнить их повторно. Следовательно, команда git revert — гораздо более удобный и безопасный способ отмены изменений.

15. Откат заданного коммита

Откатить проект до заданного коммита можно с помощью параметра revert и идентификатора коммита. Создастся новый коммит — копия коммита с предоставленным идентификатором — и добавится к истории текущей ветки.

git revert 1af17e

16. Создание новой ветки и переход в неё

Создать новую ветку можно с помощью параметра branch, указав имя ветки.

git branch new\_branch\_name

Но Git не переключится на неё автоматически. Для автоматического перехода нужно добавить флаг -b и параметр checkout.

git checkout -b new\_branch\_name

17. Просмотр списка веток

Можно просматривать полный список веток, используя параметр branch. Команда отобразит все ветки, отметит текущую звёздочкой (\*) и выделит её цветом.

git branch

Также можно вывести список удалённых веток с помощью флага -a.

git branch -a

18. Удаление ветки

Удалить ветку можно параметром branch с добавлением флага -d и указанием имени ветки. Если вы завершили работу над веткой и объединили её с основной, можно её удалить без потери истории. Однако, если выполнить команду удаления до слияния — в результате появится сообщение об ошибке. Этот защитный механизм предотвращает потерю доступа к файлам.

git branch -d existing\_branch\_name

Для принудительного удаления ветки используется флаг -D с заглавной буквой. В этом случае ветка будет удалена независимо от текущего статуса, без предупреждений.

git branch -D existing\_branch\_name

Вышеуказанные команды удаляют только локальную копию ветки. В удалённом репозитории она может сохраниться. Если хотите стереть удалённую ветку, выполните следующую команду:

git push origin --delete existing\_branch\_name

19. Слияние двух веток

Объединить две ветки можно параметром merge с указанием имени ветки. Команда объединит указанную ветку с основной.

git merge existing\_branch\_name

Если надо выполнить коммит слияния, выполните команду git merge с флагом --no-ff.

git merge --no-ff existing\_branch\_name

Указанная команда объединит заданную ветку с основной и произведёт коммит слияния. Это необходимо для фиксации всех слияний в вашем репозитории.

20. Отображение журнала фиксации в виде графика для текущей или всех веток

Просмотреть историю коммитов в виде графика для текущей ветки можно с помощью параметра log и флагов --graph --oneline --decorate. Опция --graph выведет график в формате ASCII, отражающий структуру ветвления истории коммитов. В связке с флагами --oneline и --decorate, этот флаг упрощает понимание того, к какой ветке относится каждый коммит.

git log --graph --oneline --decorate

Для просмотра истории коммитов по всем веткам используется флаг --all.

git log --all --graph --oneline --decorate

21. Прекращение слияния при конфликте

Прервать слияние в случае конфликта можно параметром merge с флагом --abort. Он позволяет остановить процесс слияния и вернуть состояние, с которого этот процесс был начат.

git merge --abort

Также при конфликте слияния можно использовать параметр reset, чтобы восстановить конфликтующие файлы до стабильного состояния.

git reset

22. Добавление удалённого репозитория

Добавить удалённый репозиторий можно параметром remote add, указав shortname и url требуемого репозитория.

git remote add awesomeapp https://github.com/someurl..

23. Просмотр удалённых URL-адресов

Просматривать удалённые URL-адреса можно параметром remote с флагом -v. Этот параметр отображает удалённые подключения к другим репозиториям.

git remote -v

Такая команда открывает доступ к интерфейсу управления удалёнными записями, которые хранятся в файле .git/config репозитория.

24. Получение дополнительных сведений об удалённом репозитории

Получить подробные сведения об удалённом репозитории можно с помощью параметра remote show с указанием имени репозитория — например, origin.

git remote show origin

Эта команда отображает список веток, связанных с удалённым репозиторием, а также рабочих станций, подключённых для получения и отправки файлов.

25. Отправка изменений в удалённый репозиторий

Отправлять изменения в удалённый репозиторий можно параметром push с указанием имени репозитория и ветки.

git push origin main

Эта команда передаёт локальные изменения в центральный репозиторий, где с ними могут ознакомиться другие участники проекта.

26. Получение изменений из удалённого репозитория

Для загрузки изменений из удалённого репозитория используется параметр pull. Он скачивает копию текущей ветки с указанного удалённого репозитория и объединяет её с локальной копией.

git pull

Также можно просмотреть подробные сведения о загруженных файлах с помощью флага --verbose.

git pull --verbose

27. Слияние удалённого репозитория с локальным

Слияние удалённого репозитория с локальным выполняется параметром merge с указанием имени удалённого репозитория.

git merge origin

28. Отправка новой ветки в удалённый репозиторий

Передать новую ветку в удалённый репозиторий можно параметром push с флагом -u, указав имя репозитория и имя ветки.

git push -u origin new\_branch

29. Удаление удалённой ветки

Чтобы избавиться от удалённой ветки, используйте параметр push с флагом --delete, указав имя удалённого репозитория и имя ветки.

git push --delete origin existing\_branch

30. Использование перебазирования

Для доступа к этой функции используйте параметр rebase с указанием имени ветки. Перебазирование — это процесс объединения или перемещения последовательности коммитов на новый родительский снимок.

git rebase branch\_name

Эта команда изменит основу ветки с одного коммита на другой, как если бы вы начали ветку с другого коммита. В Git это достигается за счёт создания новых коммитов и применения их к указанному базовому коммиту. Необходимо понимать, что, хотя ветка и выглядит такой же, она состоит из совершенно новых коммитов.

▍ Спасибо за внимание!

Вы узнали 30 основных команд интерфейса командной строки Git. Теперь, при условии регулярной практики, у вас есть всё необходимое, чтобы достичь мастерства в управлении репозиториями GitHub.

# Шпаргалка по консольным командам Git

### Добавляйте свои команды и остальные полезности через Pull request.

## Общее

Git — система контроля версий (файлов). Что-то вроде возможности сохраняться в компьютерных играх (в Git эквивалент игрового сохранения — коммит). **Важно**: добавление файлов к «сохранению» двухступенчатое: сначала добавляем файл в индекс (git add), потом «сохраняем» (git commit).

Любой файл в директории существующего репозитория может находиться или не находиться под версионным контролем (отслеживаемые и неотслеживаемые).

Отслеживаемые файлы могут быть в 3-х состояниях: неизменённые, изменённые, проиндексированные (готовые к коммиту).

### Ключ к пониманию

Ключ к пониманию концепции git — знание о «трех деревьях»:

* Рабочая директория — файловая система проекта (те файлы, с которыми вы работаете).
* Индекс — список отслеживаемых git-ом файлов и директорий, промежуточное хранилище изменений (редактирование, удаление отслеживаемых файлов).
* Директория .git/ — все данные контроля версий этого проекта (вся история разработки: коммиты, ветки, теги и пр.).

Коммит — «сохранение» (хранит набор изменений, сделанный в рабочей директории с момента предыдущего коммита). Коммит неизменен, его нельзя отредактировать.

У всех коммитов (кроме самого первого) есть один или более родительских коммитов, поскольку коммиты хранят изменения от предыдущих состояний.

### Простейший цикл работ

* Редактирование, добавление, удаление файлов (собственно, работа).
* Индексация/добавление файлов в индекс (указание для git какие изменения нужно будет закоммитить).
* Коммит (фиксация изменений).
* Возврат к шагу 1 или отход ко сну.

### Указатели

* HEAD — указатель на текущий коммит или на текущую ветку (то есть, в любом случае, на коммит). Указывает на родителя коммита, который будет создан следующим.
* ORIG\_HEAD — указатель на коммит, с которого вы только что переместили HEAD (командой git reset ..., например).
* Ветка (master, develop etc.) — указатель на коммит. При добавлении коммита, указатель ветки перемещается с родительского коммита на новый.
* Теги — простые указатели на коммиты. Не перемещаются.

### Настройки

Перед началом работы нужно выполнить некоторые настройки:

git config --global user.name "Your Name" # указать имя, которым будут подписаны коммиты

git config --global user.email "e@w.com" # указать электропочту, которая будет в описании коммитера

Если вы в Windows:

git config --global core.autocrlf true # включить преобразование окончаний строк из CRLF в LF

### Указание неотслеживаемых файлов

Файлы и директории, которые не нужно включать в репозиторий, указываются в файле .gitignore. Обычно это устанавливаемые зависимости (node\_modules/, bower\_components/), готовая сборка build/ или dist/ и подобные, создаваемые при установке или запуске. Каждый файл или директория указываются с новой строки, [возможно использование шаблонов](http://git-scm.com/book/ru/v2/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git-%D0%97%D0%B0%D0%BF%D0%B8%D1%81%D1%8C-%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B9-%D0%B2-%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%B9#%D0%98%D0%B3%D0%BD%D0%BE%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2).

### Консоль

[Как использовать консоль Bash в Windows, основные команды](https://github.com/cyberspacedk/BASH-Commands).

### Длинный вывод в консоли: Vim

Вызов некоторых консольных команд приводит к необходимости очень длинного вывода в консоль (пример: вывод истории всех изменений в файле командой git log -p fileName.txt). При этом прямо в консоли запускается редактор [Vim](https://ru.wikipedia.org/wiki/Vim). Он работает в нескольких режимах, из которых Вас заинтересуют режим вставки (редактирование текста) и нормальный (командный) режим. Чтобы попасть из Vim обратно в консоль, нужно в командном режиме ввести :q. Переход в командный режим из любого другого: Esc.

Если нужно что-то написать, нажмите i — это переход в режим вставки текста. Если нужно сохранить изменения, перейдите в командный режим и наберите :w.

# Vim (некоторые команды)

# Нажатия кнопок

ESC — переход в командный режим

i — переход в режим редактирования текста

ZQ (зажат Shift, поочередное нажатие) — выход без сохранения

ZZ (зажат Shift, поочередное нажатие) — сохранить и выйти

```bash

# Нажатия кнопок

ESC — переход в командный режим

i — переход в режим редактирования текста

ZQ (зажат Shift, поочередное нажатие) — выход без сохранения

ZZ (зажат Shift, поочередное нажатие) — сохранить и выйти

# Ввод в командном режиме

:q! — выйти без сохранения

:wq — сохранить файл и выйти

:w filename.txt — сохранить файл как filename.txt

## Консольные команды

### Создать новый репозиторий

git init # создать новый проект в текущей директории

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

### Клонирование репозитория

# клонировать удаленный репозиторий в одноименную директорию

git clone https://github.com/cyberspacedk/Git-commands.git

# клонировать удаленный репозиторий в директорию «FolderName»

git clone https://github.com/cyberspacedk/Git-commands.git FolderName

# клонировать репозиторий в текущую директорию

git clone https://github.com:nicothin/web-design.git .

### Просмотр изменений

git status # показать состояние репозитория (отслеживаемые, изменённые, новые файлы и пр.)

git diff # сравнить рабочую директорию и индекс (неотслеживаемые файлы ИГНОРИРУЮТСЯ)

git diff --color-words # сравнить рабочую директорию и индекс, показать отличия в словах (неотслеживаемые файлы ИГНОРИРУЮТСЯ)

git diff index.html # сравнить файл из рабочей директории и индекс

git diff HEAD # сравнить рабочую директорию и коммит, на который указывает HEAD (неотслеживаемые файлы ИГНОРИРУЮТСЯ)

git diff --staged # сравнить индекс и коммит с HEAD

git diff master feature # посмотреть что сделано в ветке feature по сравнению с веткой master

git diff --name-only master feature # посмотреть что сделано в ветке feature по сравнению с веткой master, показать только имена файлов

git diff master...feature # посмотреть что сделано в ветке feature с момента (коммита) расхождения с master

### Добавление изменений в индекс

git add . # добавить в индекс все новые, изменённые, удалённые файлы из текущей директории и её поддиректорий

git add text.txt # добавить в индекс указанный файл (был изменён, был удалён или это новый файл)

git add -i # запустить интерактивную оболочку для добавления в индекс только выбранных файлов

git add -p # показать новые/изменённые файлы по очереди с указанием их изменений и вопросом об отслеживании/индексировании

### Удаление изменений из индекса

git reset # убрать из индекса все добавленные в него изменения (в рабочей директории все изменения сохранятся), антипод git add

git reset readme.txt # убрать из индекса изменения указанного файла (в рабочей директории изменения сохранятся)

### Отмена изменений

git checkout text.txt # ОПАСНО: отменить изменения в файле, вернуть состояние файла, имеющееся в индексе

git reset --hard # ОПАСНО: отменить изменения; вернуть то, что в коммите, на который указывает HEAD (незакомиченные изменения удалены из индекса и из рабочей директории, неотслеживаемые файлы останутся на месте)

git clean -df # удалить неотслеживаемые файлы и директории

### Коммиты

git commit -m "Name of commit" # зафиксировать в коммите проиндексированные изменения (закоммитить), добавить сообщение

git commit -a -m "Name of commit" # проиндексировать отслеживаемые файлы (ТОЛЬКО отслеживаемые, но НЕ новые файлы) и закоммитить, добавить сообщение

### Отмена коммитов и перемещение по истории

Все коммиты, которые уже были отправлены в удалённый репозиторий, должны отменяться новыми коммитами (git revert), дабы избежать проблем с историей разработки у других участников проекта.

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

git revert b9533bb --no-edit # то же, но отменяются изменения, внесённые коммитом с указанным хешем (b9533bb)

**Все команды, приведённые ниже можно выполнять ТОЛЬКО если коммиты еще не были отправлены в удалённый репозиторий.**

# ВНИМАНИЕ! Опасные команды, можно потерять незакоммиченные изменения

git commit --amend -m "Название" # «перекоммитить» изменения последнего коммита, заменить его новым коммитом с другим сообщением (сдвинуть текущую ветку на один коммит назад, сохранив рабочую директорию и индекс «как есть», создать новый коммит с данными из «отменяемого» коммита, но новым сообщением)

git reset --hard @~ # передвинуть HEAD (и ветку) на предыдущий коммит, рабочую директорию и индекс сделать такими, какими они были в момент предыдущего коммита

git reset --hard 75e2d51 # передвинуть HEAD (и ветку) на коммит с указанным хешем, рабочую директорию и индекс сделать такими, какими они были в момент указанного коммита

git reset --soft @~ # передвинуть HEAD (и ветку) на предыдущий коммит, но в рабочей директории и индексе оставить все изменения

git reset --soft @~2 # то же, но передвинуть HEAD (и ветку) на 2 коммита назад

git reset @~ # передвинуть HEAD (и ветку) на предыдущий коммит, рабочую директорию оставить как есть, индекс сделать таким, каким он был в момент предыдущего коммита (удобнее, чем git reset --soft @~, если индекс нужно задать заново)

# Почти как git reset --hard, но безопаснее: не получится потерять изменения в рабочей директории

git reset --keep @~ # передвинуть HEAD (и ветку) на предыдущий коммит, сбросить индекс, но в рабочей директории оставить изменения, если возможно (если файл с изменениями между коммитами менялся, будет выдана ошибка и переключение не произойдёт)

### Временно переключиться на другой коммит

git checkout b9533bb # переключиться на коммит с указанным хешем (переместить HEAD на указанный коммит, рабочую директорию вернуть к состоянию, на момент этого коммита)

git checkout master # переключиться на коммит, на который указывает master (переместить HEAD на коммит, на который указывает master, рабочую директорию вернуть к состоянию на момент этого коммита)

### 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 c хешем 5589877 (переместить HEAD на указанный коммит, рабочую директорию вернуть к состоянию, на момент этого коммита, создать указатель на этот коммит (ветку) с указанным именем)

### Восстановление изменений

git checkout 5589877 index.html # восстановить в рабочей директории указанный файл на момент указанного коммита (и добавить это изменение в индекс) (git reset index.html для удаления из индекса, но сохранения изменений в файле)

### Копирование коммита (перенос коммитов)

git cherry-pick 5589877 # скопировать на активную ветку изменения из указанного коммита, закоммитить эти изменения

git cherry-pick master~2..master # скопировать на активную ветку изменения из master (2 последних коммита)

git cherry-pick -n 5589877 # скопировать на активную ветку изменения из указанного коммита, но НЕ КОММИТИТЬ (подразумевается, что мы сами потом закоммитим)

git cherry-pick master..feature # скопировать на активную ветку изменения из всех коммитов ветки feature с момента её расхождения с master (похоже на слияние веток, но это копирование изменений, а не слияние), закоммитить эти изменения; это может вызвать конфликт

git cherry-pick --abort # прервать конфликтный перенос коммитов

git cherry-pick --continue # продолжить конфликтный перенос коммитов (сработает только после решения конфликта)

### Удаление файла

git rm text.txt # удалить отслеживаемый неизменённый файл и проиндексировать это изменение

git rm -f text.txt # удалить отслеживаемый изменённый файл и проиндексировать это изменение

git rm -r log/ # удалить всё содержимое отслеживаемой директории log/ и проиндексировать это изменение

git rm ind\* # удалить все отслеживаемые файлы с именем, начинающимся на «ind» в текущей директории и проиндексировать это изменение

git rm --cached readme.txt # удалить из отслеживаемых индексированный файл (ФАЙЛ ОСТАНЕТСЯ НА МЕСТЕ) (часто используется для нечаянно добавленных в отслеживаемые файлов)

### Перемещение/переименование файлов

Для git не существует переименования. Переименование воспринимается как удаление старого файла и создание нового. Факт переименования может быть определен только после индексации изменения.

git mv text.txt test\_new.txt # переименовать файл «text.txt» в «test\_new.txt» и проиндексировать это изменение

git mv readme\_new.md folder/ # переместить файл readme\_new.md в директорию folder/ (должна существовать) и проиндексировать это изменение

### История коммитов

Выход из длинного лога вывода: q.

git log master # показать коммиты в указанной ветке

git log -2 # показать последние 2 коммита в активной ветке

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

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 --after '2018-06-30' # показать коммиты, сделанные после указанной даты

git log index.html # показать историю изменений файла index.html (только коммиты)

git log -5 index.html # показать историю изменений файла index.html, последние 5 коммитов (только коммиты)

git log -p index.html # показать историю изменений файла index.html (коммиты и изменения)

git log -G'myFunction' -p # показать все коммиты, в которых менялись строки с myFunction (в кавычках регулярное выражение)

git log -L '/<head>/','/<\/head>/':index.html # показать изменения от указанного до указанного регулярных выражений в указанном файле

git log --grep fix # показать коммиты, в описании которых есть буквосочетание fix (регистрозависимо, только коммиты текущей ветки)

git log --grep fix -i # показать коммиты, в описании которых есть буквосочетание fix (регистроНЕзависимо, только коммиты текущей ветки)

git log --grep 'fix(ing|me)' -P # показать коммиты, в описании которых есть совпадения для регулярного выражения (только коммиты текущей ветки)

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

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 log master...branch\_99 --boundary -- graph # показать коммиты из указанных веток, начиная с их расхождения (коммит расхождения будет показан)

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

git show HEAD~ # показать данные о предыдущем коммите в активной ветке

git show @~ # аналогично предыдущему

git show HEAD~3 # показать данные о коммите, который был 3 коммита назад

git show my\_branch~2 # показать данные о коммите, который был 2 коммита назад в указанной ветке

git show @~:index.html # показать контент указанного файла на момент предыдущего (от HEAD) коммита

git show :/"подвал" # показать самый новый коммит, в описании которого есть указанное слово (из любой ветки)

### Кто написал строку

git blame README.md --date=short -L 5,8 # показать строки 5-8 указанного файла и коммиты, в которых строки были добавлены

### История изменений указателей (веток, HEAD)

git reflog -20 # показать последние 20 изменений положения указателя HEAD

git reflog --format='%C(auto)%h %<|(20)%gd %C(blue)%cr%C(reset) %gs (%s)' -20 # то же, но с указанием давности действий

### 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 на текущем коммите

git branch new\_branch 5589877 # создать новую ветку с указанным именем на указанном коммите

git branch -f master 5589877 # переместить ветку master на указанный коммит

git branch -f master master~2 # переместить ветку master на 2 коммита назад

git checkout new\_branch # перейти в указанную ветку

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

git checkout -B master 5589877 # переместить ветку с указанным именем на указанный коммит и перейти в неё

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

git merge hotfix -m "Горячая правка" # влить в ветку, в которой находимся, данные из ветки hotfix (указано сообщение коммита слияния)

git merge hotfix --log # влить в ветку, в которой находимся, данные из ветки hotfix, показать редактор описания коммита, добавить в него сообщения вливаемых коммитов

git merge hotfix --no-ff # влить в ветку, в которой находимся, данные из ветки hotfix, запретить простой сдвиг указателя, изменения из hotfix «останутся» в ней, а в активной ветке появится только коммит слияния

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

git branch --merged # показать ветки, уже слитые с активной

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

### Теги

git tag v1.0.0 # создать тег с указанным именем на коммите, на который указывает HEAD

git tag -a -m 'В продакшен!' v1.0.1 master # создать тег с описанием на том коммите, на который смотрит ветка master

git tag -d v1.0.0 # удалить тег с указанным именем(ами)

git tag -n # показать все теги, и по 1 строке сообщения коммитов, на которые они указывают

git tag -n -l 'v1.\*' # показать все теги, которые начинаются с 'v1.\*'

### Временное сохранение изменений без коммита

git stash # временно сохранить незакоммиченные изменения и убрать их из рабочей директории

git stash pop # вернуть сохраненные командой git stash изменения в рабочую директорию

### Удалённые репозитории

Есть два распространённых способа привязать удалённый репозиторий к локальному: по HTTPS и по SSH. Если SSH у вас не настроен (или вы не знаете что это), привязывайте удалённый репозиторий по HTTPS (адрес привязываемого репозитория должен начинаться с https://).

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

git branch -r # показать удаленные ветки

git branch -a # показать все ветки(локальные и удаленные)

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

git remote add origin https://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 --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)

### Конфликт слияния

Предполагается ситуация: есть ветка master и есть ветка feature. В обеих ветках есть коммиты, сделанные после расхождения веток. В ветку master пытаемся влить ветку feature (git merge feature), получаем конфликт, т.к. в обеих ветках есть изменения одной и той же строки в файле index.html.

При возникновении конфликта, репозиторий находится в состоянии прерванного слияния. Нужно оставить в конфликтующих местах файлов только нужный код, проиндексировать изменения и закоммитить.

git merge feature # влить в активную ветку изменения из ветки feature

git merge-base master feature # показать хеш последнего общего коммита для двух указанных веток

git checkout --ours index.html # оставить в конфликтном файле (index.html) состояние ветки, В КОТОРУЮ мы вливаем (в примере — из ветки master)

git checkout --theirs index.html # оставить в конфликтном файле (index.html) состояние ветки, ИЗ КОТОРОЙ мы вливаем (в примере — из ветки feature)

git checkout --merge index.html # показать в конфликтном файле (index.html) сравнение содержимого сливаемых веток (для ручного редактирования)

git checkout --conflict=diff3 --merge index.html # показать в конфликтном файле (index.html) сравнение содержимого сливаемых веток плюс то, что было в месте конфликта в коммите, на котором разошлись сливаемые ветки

git reset --hard # прекратить это прерванное слияние, вернуть рабочую директорию и индекс как было в момент коммита, на который указывает HEAD, а я пойду немного поплачу

git reset --merge # прекратить это прерванное слияние, но оставить изменения, не закоммиченные до слияния (для случая, когда слияние делается не на чистом статусе)

git reset --abort # то же, что и строкой выше

### «Перенос» ветки

Можно «переместить» ответвление какой-либо ветки от основной на произвольный коммит. Это нужно для того, чтобы в «переносимой» ветке появились какие-либо изменения, внесённые в основной ветке (уже после ответвления переносимой).

Нельзя «переносить» ветку, если она уже отправлена на удалённый репозиторий.

git rebase master # перенести все коммиты (создать их копии) активной ветки так, будто активная ветка ответвилась от master на нынешней вершине master (часто вызывает конфликты)

git rebase --onto master feature # перенести коммиты активной ветки на master, начиная с того места, в котором активная ветка отделилась от ветки feature

git rebase --abort # прервать конфликтный rebase, вернуть рабочую директорию и индекс к состоянию до начала rebase

git rebase --continue # продолжить конфликтный rebase (сработает только после разрешения конфликта и индексации такого разрешения)

#### Как отменить rebase

git reflog feature -2 # смотрим лог перемещений ветки, которой делали rebase (в этом примере — feature), видим последний коммит ПЕРЕД rebase, на него и нужно перенести указатель ветки

git reset --hard feature@{1} # переместить указатель ветки feature на один коммит назад, обновить рабочую директорию и индекс

### Разное

git archive -o ./project.zip HEAD # создать архив с файловой структурой проекта по указанному пути (состояние репозитория, соответствующее указателю HEAD)

## Примеры

Собираем коллекцию простых и сложных примеров работы.

### Начало работы

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

# указана последовательность действий:

# создана директория проекта, мы в ней

git init # создаём репозиторий в этой директории

touch readme.md # создаем файл readme.md

git add readme.md # добавляем файл в индекс

git commit -m "Старт" # создаем коммит

git remote add origin https://github.com:nicothin/test.git # добавляем предварительно созданный пустой удаленный репозиторий

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

### «Внесение изменений» в коммит

Только если коммит ещё не был отправлен в удалённые репозиторий.

# указана последовательность действий:

subl inc/header.html # редактируем и сохраняем разметку «шапки»

git add inc/header.html # индексируем измененный файл

git commit -m "Убрал телефон из шапки" # делаем коммит

# ВНИМАНИЕ: коммит пока не был отправлен в удалённый репозиторий

# сознаём, что нужно было еще что-то сделать в этом коммите.

subl inc/header.html # вносим изменения

git add inc/header.html # индексируем измененный файл (можно git add .)

git commit --amend -m "«Шапка»: выполнена задача №34" # заново делаем коммит

### Работа с ветками

Есть master (публичная версия сайта), выполняем масштабную задачу (переверстать «шапку»), но по ходу работ возникает необходимость подправить критичный баг (неправильно указан контакт в «подвале»).

# указана последовательность действий:

git checkout -b new-page-header # создадим новую ветку для задачи изменения «шапки» и перейдём в неё

subl inc/header.html # редактируем разметку «шапки»

git commit -a -m "Новая шапка: смена логотипа" # делаем коммит (работа еще не завершена)

# тут выясняется, что есть баг с контактом в «подвале»

git checkout master # возвращаемся к ветке master

subl inc/footer.html # устраняем баг и сохраняем разметку «подвала»

git commit -a -m "Исправление контакта в подвале" # делаем коммит

git push # отправляем коммит с быстрым критическим изменением в master в удалённом репозитории

git checkout new-page-header # переключаемся обратно в ветку new-page-header для продолжения работ над «шапкой»

subl inc/header.html # редактируем и сохраняем разметку «шапки»

git commit -a -m "Новая шапка: смена навигации" # делаем коммит (работа над «шапкой» завершена)

git checkout master # переключаемся в ветку master

git merge new-page-header # вливаем в master изменения из ветки new-page-header

git branch -d new-page-header # удаляем ветку new\_page\_header

### Работа с ветками, слияние и откат к состоянию до слияния

Была ветка fix, в которой исправляли баг. Исправили, влили fix в master. но тут выяснилось, что это исправление ломает какую-то функциональность, Нужно откатить master к состоянию без слияния (наличие бага менее критично, чем порча функциональности).

# находимся в ветке fix, баг уже «исправлен»

git checkout master # переключаемся на master

git merge fix # вливаем изменения из fix в master

# видим проблему: часть функциональности сломалась

git checkout fix # переключаемся на fix (пока мы в master, git не даст ее двигать)

git branch -f master ORIG\_HEAD # передвигаем ветку master на коммит, указанный в ORIG\_HEAD (тот, на который указывала master до вливания fix)

### Работа с ветками, конфликт слияния

Есть ветка master (публичная версия сайта), в двух параллельных ветках (branch-1 и branch-2) было отредактировано одно и то же место одного и того же файла, первую ветку (branch-1) влили в master, попытка влить вторую вызывает конфликт.

# указана последовательность действий:

git checkout master # переключаемся на ветку master

git checkout -b branch-1 # создаём ветку branch-1, основанную на ветке master

subl . # редактируем и сохраняем файлы

git commit -a -m "Правка 1" # коммитим

git checkout master # возвращаемся к ветке master

git checkout -b branch-2 # создаём ветку branch-2, основанную на ветке master

subl . # редактируем и сохраняем файлы

git commit -a -m "Правка 2" # коммитим

git checkout master # возвращаемся к ветке master

git merge branch-1 # вливаем изменения из ветки branch-1 в текущую ветку (master), удача (автослияние)

git merge branch-2 # вливаем изменения из ветки branch-2 в текущую ветку (master), КОНФЛИКТ автослияния

# Automatic merge failed; fix conflicts and then commit the result.

subl . # выбираем в конфликтных файлах те участки, которые нужно оставить, сохраняем

git commit -a -m "Устранение конфликта" # коммитим результат устранения конфликта

### Синхронизация репозитория-форка с мастер-репозиторием

Есть некий репозиторий на github.com, он него нами был сделан форк, добавлены какие-то изменения. Оригинальный (мастер-)репозиторий был как-то обновлён. Задача: стянуть с мастер-репозитория изменения (которые там внесены уже после того, как мы его форкнули).

# указана последовательность действий:

git remote add upstream https://github.com:address.git # добавляем удаленный репозиторий: сокр. имя — upstream, URL мастер-репозитория

git fetch upstream # стягиваем все ветки мастер-репозитория, но пока не сливаем со своими

git checkout master # переключаемся на ветку master своего репозитория

git merge upstream/master # вливаем стянутую ветку master удалённого репозитория upstream в свою ветку master

### Ошибка в работе: закоммитили в мастер, но поняли, что нужно было коммитить в новую ветку

**ВАЖНО: это сработает только если коммит еще не отправлен в удалённый репозиторий.**

# указана последовательность действий:

# сделали изменения, проиндексировали их, закоммитили в master, но ЕЩЁ НЕ ОТПРАВИЛИ (не делали git push)

git checkout -b new-branch # создаём новую ветку из master

git checkout master # переключаемся на master

git reset HEAD~ --hard # сдвигаем указатель (ветку) master на 1 коммит назад

git checkout new-branch # переключаемся обратно на новую ветку для продолжения работы

### Нужно вернуть содержимое файла к состоянию, бывшему в каком-либо коммите (известен хеш коммита)

# указана последовательность действий:

git checkout f26ed88 -- index.html # восстановить в рабочей директории состояние указанного файла на момент указанного коммита, добавить это изменение в индекс

git commit -am "Navigation fixs" # сделать коммит

### При любом действии с github (или другим удалённым сервисом) запрашивается логин и пароль

Речь именно о запросе пары логин + пароль, а не ключевой фразы. Происходит это потому, что git по умолчанию не сохранит пароль для доступа к репозиторию по HTTPS.

Простое решение: [указать git кешировать ваш пароль](https://help.github.com/articles/caching-your-github-password-in-git/).

## .gitattributes

\* text=auto

\*.html diff=html

\*.css diff=css

\*.scss diff=css