**Шпаргалка по Git. Решение основных проблем**

**Восстановление накопленных изменений**

В том случае, если изменения, внесённые пользователем, находятся в режиме накопления, применить их к ветке можно с помощью команды git stashapply. Также можно запустить git diff — эта команда поможет выявить различия. Для того, чтобы затем избавиться от накопленных данных, нужно запустить команду:

git stashdrop

Если существует более одного накопления, найти нужное можно с помощью команды:

git stashlist затем можно применить его, воспользовавшись его индексом:

git stash@{1}

Необходимо учитывать, что отсчёт индексов ведётся от нуля.

**Восстановление удалённого тега**

В том случае, если необходимо восстановить случайно удалённый тег, начать можно с его поиска:

git fsck --unreachable | greptag

После того, как нужный тег найден, его следует восстановить:

git update-ref refs/tags/название-тега

**Восстановление удалённого файла**

Если вы случайно удалили файл, его можно быстро восстановить:

git checkout myfile.txt

Если требуется восстановить файл из конкретной временной точки истории коммитов, следует узнать хеш нужного коммита и запустить команду:

git checkout $commit~1 myfile.txt

**Восстановление удалённой ветки**

С помощью комманды git reflog можно узнать хеш (SHA1) последнего коммита в удалённой ветке. Скопируйте этот хеш и используйте в команде:

git checkout<sha>

После этого восстановить удалённую ветку можно будет вот такой командой:

git checkout -b <название-ветки>

**Изменение сообщения коммита перед его отправкой**

Изменить сообщение коммита можно с помощью команды git commit --amend, она откроет редактор, в котором можно будет внести необходимые поправки в последнее сообщение.

Сообщение можно изменить и напрямую с помощью команды

git commit --amend -m "Новое прекрасное сообщение"

**Изменение сообщения коммита после его отправки**

В данном случае процесс занимает два шага. Сначала нужно изменить сообщение с помощью комманды git commit --amend, а затем перезаписать историю коммитов локальной ветки: git push<remote><branch> --force

**Предупреждение:** подобная насильная перезапись может привести к потерикоммитов из внешней ветки, если с ней давно не было синхронизации, соблюдайте осторожность.

**Использование алиасов команд в командной строке**

Устали каждый раз печатать git status? Этой команде можно присвоить простой алиас, который проще и быстрее вбивать в git.

gitconfig --global alias.st status

— теперь нужно писать только git st

Можно пойти дальше и присвоить алиасы более сложным командам:

gitconfig --global alias.logme 'log -p --author=Rob'

Теперь алиас git logme будет выводить все наши коммиты.

**Коммит в неправильную ветку**

Нужно переключиться на новую ветку, которую вы забыли предварительно создать:

git checkout -b название-новой-ветки.

А затем переключиться к оригинальной ветке:

git checkout название-оригинальной-ветки

...и «откатиться» до последнего коммита, который нужно сохранить.

Чтобы это сделать, можно воспользоваться командой git log и сохранить хеш (SHA1) последнего коммита, который нужно оставить.. Например, это a31a45c.

Теперь его нужно сбросить: git reset --hard a31a45c и отправить получившийся результат.

**Предупреждение:** Убедитесь в том, что никто не отправлял коммиты в оригинальную ветку во время выполнения вышеописанных процедур, в противном случае эти изменения будут потеряны!

**Обновление конкретного подмодуля**

Чтобы обновить конкретный подмодуль в репозитории, нужно добавить путь к подмодулю:

git submodule update --remote --merge <path>

**Откат к конкретному коммиту в истории**

Если вас не очень беспокоят изменения в локальном репозитории, то можно «откатиться» к конкретному коммиту в истории с помощью команды:

git reset --hard HEAD~1

Эта команда установит HEAD на конкретный коммит. Также можно воспользоваться хешемкоммита.

**Отмена коммита до публикации изменений**

Если вы сделали коммит, который впоследствии понадобилось отредактировать или полностью стереть, поможет команда git reset.

git reset HEAD~1 # отменить последний коммит, сохранить изменения

git reset --hard HEAD~1 # отменить последний коммит, стереть изменения

Будьте осторожны используя второй вариант, поскольку изменения ваших локальных файлов будут потеряны.

Чтобы сохранить сообщение коммита, наберите: :

git commit -i ORIG\_HEAD

**Отмена коммита после отправки его в master-репозиторий**

Рассмотрим процедуру возврата одного или нескольких коммитов, которые нужно стереть из удалённой ветки. Обозначить конкретный коммит можно с помощью его хеша:

git revert b712c3c

Отмена только коммита, который является вторым после последнего:

git revert HEAD^

Простая отмена последнего коммита:

git revert -n HEAD

**Отмена локальных изменений файлов**

Простейшим способом избавиться от нежелательных изменений для файлов и папок является восстановление состояния последнего коммита. Сделать это можно с помощью специальной команды:

git checkout myfile.txt

Кроме того, можно восстановить конкретный путь к файлу:

git checkout -- путь-до-файла

**Отображение всех коммитов одного файла**

Если вы хотите просмотреть все коммиты с изменениями конкретного файла, воспользуйтесь командой git log --follow -p -- myfile

Аргумент —follow позволяет вывести все изменения над файлом, даже если в процессе работы он был переименован.

Если опустить опцию -p, то система выведет только сообщения коммитов, но не их содержимое.

**Отображения числа коммитов от каждого участника**

Хотите узнать, сколько коммитов сделал каждый участник команды?

Эта команда выведет список, отсортированный в порядке убывания количества коммитов: git shortlog -s -n

**Отобразить коммиты, содержащие удалённые файлы**

Узнать, в каких коммитах содержатся удалённые файлы, можно с помощью команды:

git log --diff-filter=D --summary

Она покажет список коммитов, в которых удалялись файлы.

**Отсортировать коммиты по автору**

Чтобы вывести список коммитов, отфильтрованных по автору, следует воспользоваться следующей командой:

git log --author="Имяавтора"

**Очистка всех скрытых состояний**

Очистить все скрытые состояния можно следующей командой:

git stashclear

**Переименование локальной и удалённой ветки**

Предложим, у вас есть ветка *«fix-bug25»*, которую вы захотели переименовать в *«hotfix-users»*. Прежде всего, нужно будет изменить локальную ветку:

git branch -m fix-bug25 hotfix-users

А затем — удалённую ветку: переименовать её напрямую нельзя, поэтому нужно будет её удалить, и затем опубликовать заново уже с новым именем. Прежде чем приступать к этим процедурам, следует убедиться, что никто из членов команды не работает с этой веткой! Удаляем ветку: git pushorigin :fix-bug25

А теперь заново публикуем её с новым именем: git pushoriginhotfix-users

**Переименование тега**

Чтобы переименовать существующий тег:

git tag новое-название-тега старое-название-тега

git tag -d старое-название-тега

git push origin :refs/tags/старое-название-тега

git push --tags

**Перестать отслеживать существующие файлы**

Если вы хотите перестать отслеживать файлы, которые уже есть в репозитории, но при этом желаете сохранить его локально, осуществите коммит изменений и запустите команду:

git rm -r --cached

Она удалит изменённые файлы из зоны подготовленных файлов (stagingarea). Затем нужно запустить команду:

git add .

и отправить изменения.

**Подготовка удалённых файлов**

Чтобы подготовить к коммиту файлы и папки, которые были удалены локально, можно использовать специальную команду:

git add -u

Если требуется подготовить только используемый в данный момент путь, воспользуйтесь командой

git add -u .

**Поиск конкретного сообщения во всех коммитах**

Чтобы найти конкретный текст сообщения коммита, соответствующий регулярному выражению, нужно воспользоваться командой

git log --grep<запрос>

**Пометить конфликтующий файл, как разрешённый**

Чтобы пометить один или несколько конфликтующих файлов, как разрешённые, чтобы их можно было нормально изменять, воспользуйтесь командой:

git add название-файла

Затем можно запустить git commit, чтобы разрешить конфликты и опубликовать изменения.

**Просмотр всех неотправленных коммитов**

Чтобы просмотреть все коммиты, которые ещё не были отправлены в соответствующие ветки, воспользуйтесь следующей командой:

git log --branches --not --remotes

Крометого, можноиспользовать:

git log origin/master..HEAD

**Просмотр старой ревизии файла**

Существует возможность просмотреть содержимое файла в конкретный момент времени в прошлом. Для этого нужно использовать команду:

git showcommitHash:myfile.txt

**Публикация локальной ветки для удалённого редактирования**

Если вы создали локальную ветку, и хотите, чтобы другие пользователи могли с ней работать, воспользуйтесь командой:

git push -u origin название-моей-новой-ветки

Теперь они тоже смогут вносить изменения в эту ветку.

**Сброс локальной ветки до состояния удалённой**

В том случае, если отсутствуют изменения, которые необходимо сохранить, сбросить локальную ветку до состояния удалённой можно с помощью двух простых команд.

Прежде всего нужно получить свежие обновления из удалённой ветки:

git fetch название-удалённой-ветки.

А затем нужно сообщить git, что локальную ветку следует «откатить» до состояния удалённой:

git reset --hardorigin/название-локальной-ветки.

При наличии коммита, который нужно сохранить, перед сбросом нужно создать новую ветку и произвести коммит: git commit -m «Обновление»

git branch название-новой-ветки

**Синхронизировать ветку с master-репозиторием**

Чтобы синхронизировать последние изменения в репозиторииmaster (или с любой другой веткой, с которой вы работали) необходимо «перебазировать» локальную ветку. Предположим, вы работаете над веткой foobar:

git checkoutfoobar

А затем осуществляете «перебазирование»:

git rebasemaster

После этого будут применены коммитыorigin из master. После разрешения конфликтов процесс можно продолжить с помощью команды git rebase —continue. Теперь можно продолжать работу над своей веткой или осуществить её слияние (merge) с главным репозиторием.

**Слияние локальных изменений с другой веткой**

Это можно сделать прямо в процессе стандартного слияния (merge). Вам стоит *сохранить историю слияний* используя флаг —no-ff, что означает *no fastforward*.

Перейдите в ветку, в которую будут вливаться изменения, убедитесь в её актуальности и запустите процесс:

git merge <другая-ветка> --no-ff

Затем появится сообщение о коммитеmerge X into Y branch, после чего вы можете смело запушить ваше слияние.>

**Совмещение двух и более коммитов**

Здесь нам понадобится произвести интерактивное перебазирование. Если перебазирование происходит относительно master-ветки, то начать следует с команды git rebase -i master. Однако, если перебазирование происходит не относительно ветки, то нужно будет перебазироваться относительно HEAD.

Если есть необходимость в совмещении двух последних коммитов, можно использовать команду

git rebase -i HEAD~2.

После её ввода появятся инструкции по выбору коммитов. В том случае, если необходимо совместить все коммиты с первым *старейшим* коммитов, то в первой строке нужно написать pick, а для всех остальных коммитов изменить букву на f. Подробнее[здесь](https://github.com/k88hudson/git-flight-rules" \l "i-need-to-combine-commits)

**Совмещение коммитов по конкретной функции для добавления в ветку релиза**

Если вы решите совместить и опубликовать коммиты, то возникнет новый коммит в ветке релиза, поэтому история ветки конкретной функции останется неизменной.

Ниже представлен пример того, как достичь подобного эффекта:

git fetch origin

git checkout [release-branch]

git rebase origin/[release-branch]

git merge —squash —no-commit [feature-branch]

git commit -m 'Merge X into Y'

В конечном итоге останется только один коммит в ветке релиза, а история изменений в ветке разработки конкретной функции останется нетронутой.

[Подробнее о ветках функций](https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow)

**Создание новой ветки с изменениями текущей**

Часто возникает ситуация, при которой пользователи начинают изменять файлы в ветке, чтобы что-то исправить, и лишь позднее вспоминают, что предварительно не создали новую ветку. К счастью, есть способ сделать это уже в процессе:

git checkout -b название-моей-новой-ветки

Эта команда перенесёт файлы из текущей ветки в новую, которую потом уже можно «закоммитить».

**Убрать файл из буфера**

Чтобы убрать добавленный по ошибке файл из буфера, нужно воспользоваться простой командой:

git reset HEAD unlovedfile.txt

**Удаление внешней ветки**

Если вы хотите удалить ветку, введите команду:

git push origin --delete название-ветки

**Удаление неотслеживаемых файлов и папок**

Чтобы удалить неотслеживаемые файлы и папки из рабочей копии наберите следующую команду:

git clean -f

Чтобы в принципе удалить их:

git clean -fd

Подсказка: чтобы увидеть, какие файлы являются лишними, перед их непосредственным удалением, наберите:

git clean -n

**Удаление старых веток, стёртых из внешнего репозитория**

Если ветка удалена из внешнего репозитория, её также можно стереть из локального репозитория с помощью команды

git-remoteprune название-удалённой-ветки.

Она удалит старую ветку под названием *название-удалённой-ветки*, которая уже была стёрта из внешнего репозитория, но всё ещё доступна локально в remotes/название-удалённой-ветки.

**Удаление файла из git с сохранением его локальной копии**

Для того, чтобы удалить файл из git, но сохранить его локально нужно использовать следующую команду:

git rm --cached myfile.txt

Git — Удалить Локальную Ветку

Чтобы удалить ветку из локального Git-репозитория, выполните:

$ git branch -d <имя\_ветки>

Git — Удалить Удаленную Ветку

Чтобы удалить ветку из удаленного Git-репозитория, выполните:

$ git push origin --delete **<**имя\_ветки>