**Шпаргалка по консольным командам Git**

**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 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 вертку 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