**https://msdn.microsoft.com/en-us/library/y740d9d3.aspx**

**https://www.edx.org/course/introduction-linux-linuxfoundationx-lfs101x-1**

**https://git-scm.com/book/ru/v1/Введение**

**https://github.com/nicothin/web-development/blob/master/git/readme.md#Шпаргалка-по-консольным-командам-git**

**Шпаргалка по консольным командам Git**

* [Общее](https://github.com/nicothin/web-development/tree/master/git#%D0%9E%D0%B1%D1%89%D0%B5%D0%B5)
* [Консольные команды](https://github.com/nicothin/web-development/tree/master/git#%D0%9A%D0%BE%D0%BD%D1%81%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5-%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D1%8B)
* [Примеры реальной работы](https://github.com/nicothin/web-development/tree/master/git#%D0%9F%D1%80%D0%B8%D0%BC%D0%B5%D1%80%D1%8B)

**Общее**

Git — система контроля версий (файлов). Что-то вроде возможности сохраняться в компьютерных играх (в Git эквивалент игрового сохранения — коммит).

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

Неотслеживаемые файлы можно в любой момент сделать отслеживаемыми и наоборот.

Отслеживаемые файлы могут быть в 3-х состояниях: неизменённые, изменённые, готовые к коммиту. Сразу после клонирования репозитория все файлы — отслеживаемые неизменённые.

После редактирования отслеживаемого файла он становится изменённым. Изменения файла (или нескольких) можно индексировать (подготовить к коммиту) и закоммитить. После коммита такие файлы становятся отслеживаемыми неизменёнными.

Проверить состояние репозитория можно командой git status. Она покажет ветку, изменённые файлы (если есть) с пометкой «modified», неотслеживаемые файлы в списке «Untracked files».

**Указание неотслеживаемых файлов**

Файлы и папки, которые не нужно включать в репозиторий, указываются в файле .gitignore. Обычно это устанавливаемые зависимости в папку node\_modules, компоненты в папке bower\_components, готовая сборка в папке build или dist и подобные, создаваемые при установке или запуске. Каждый файл или папка 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).

Для прочих файлов, которые не имеют непосредственного отношения к репозиторию (временные, личные или тестовые папки и файлы и т.п.), игнорирование можно указывать в файле .git/info/exclude. ВНИМАНИЕ: этот файл хранится только локально и не попадает в индексацию, его не будет в удалённом репозитории.

**Консоль**

[Как использовать консоль в Windows](http://nicothin.pro/page/console-windows) (на OSX и Linux сразу есть хорошая консоль (терминал)). [Как комфортно работать с GitHub в консоли Windows](http://nicothin.pro/page/kak-komfortno-rabotat-s-github-v-konsoli-windows).

**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.

[Чуть-чуть больше команд](https://github.com/nicothin/web-development/blob/master/git/vim.md).

**Консольные команды**

**Создать новый репозиторий**

git init # создать новый проект в текущей папке

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

**Клонирование репозитория**

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

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

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

**Добавление файлов к отслеживанию, индексация отслеживаемых**

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

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

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

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

**Убирание файла, папки из отслеживания**

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

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

**Отмена индексации**

git reset HEAD # убрать из индекса все индексированные файлы

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

**Просмотр изменений**

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

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

**Отмена изменений**

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

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

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

**Коммиты**

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

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

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

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

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

**Отмена коммитов**

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

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

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

**Временно переключиться на другой коммит**

git checkout b9533bb # временно переключиться на коммит с указанным хешем

git checkout master # вернуться к последнему коммиту в указанной ветке

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

Потребуется создание новой ветки, начинающейся с указанного коммита.

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

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

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

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

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

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

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

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

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

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

**История изменений**

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

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

git log -2 # показать последние 2 коммита

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

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

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

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

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

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

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

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

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

git show HEAD^ # показать данные о предыдущем коммите

**Ветки**

git branch # показать список веток

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

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

git checkout new\_branch # перейти в указанную ветку

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

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

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

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

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

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

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

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

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

git branch --unset-upstream # завершить процесс переименования

**Удалённые репозитории**

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

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

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

git remote rm origin # удалить привязку удалённого репозитория

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

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

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

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

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

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

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

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

**Разное**

git clean -f -d # удалить из репозитория все неотслеживаемые папки и файлы (папки и файлы, добавленные в .gitignore останутся на месте)

**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:

git init # инициируем гит в этой папке

touch readme.md # создаем файл readme.md

git add readme.md # делаем этот файл отслеживаемым

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

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

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

**Обычный рабочий процесс**

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

# указана последовательность действий:

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

# в консоли попадаем в свою папку для всех проектов

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

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

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

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

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

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

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

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

Не обязательно после каждого коммита отправлять изменения в удаленный репозиторий, можно сделать это один раз в конце работы.

**Внесение изменений в коммит**

# 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

# вносим изменения

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

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

**Работа с ветками**

Есть master (публичная версия сайта), хотим масштабно что-то поменять (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 и сохраняем разметку «шапки»

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

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

git checkout master # возвращаемся к ветке master

git checkout -b footer\_hotfix # создаём ветку (основанную на master) для решения проблемы

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

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

git checkout master # переключаемся в ветку master

git merge footer\_hotfix # вливаем в master изменения из ветки footer\_hotfix

git branch -d footer\_hotfix # удаляем ветку footer\_hotfix

git checkout new\_page\_header # переключаемся 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, конфликт слияния**

Есть 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 (теперь имеем 1 коммит в ветке branch\_1)

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 (теперь имеем 1 коммит в ветке branch\_2)

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 git@github.com:address.git # добавляем удаленный репозиторий: сокр. имя — upstream, URL мастер-репозитория

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

git checkout master # переключаемся на ветку master своего репозитория

git merge upstream/master # вливаем 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 # жОско сбрасываем состояние master

git checkout new-branch # переключаемся обратно на новую ветку

**Нужно вернуть содержимое файла к состоянию, бывшему в каком-либо коммите (известна SHA коммита)**

# указана последовательность действий:

git checkout f26ed88 -- index.html # указана SHA коммита, к состоянию которого нужно вернуть файл и имя файла

git status # изменения внесены в файл, файл сразу проиндексирован

git diff --staged # показать изменения в файле

git commit -am "Navigation fixs" # коммит

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

Речь именно о запросе пары логин + пароля, 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/). Решение посложнее: [использовать протокол SSH](https://help.github.com/articles/connecting-to-github-with-ssh/): сгенерировать SSH-ключ, сообщить публичную часть ключа удаленному сервису (github-у), привязать удаленный репозиторий по SSH:

# указана последовательность действий:

git remote -v # показать список удалённых репозиториев с адресами (у проблемного будет адрес по https), предположим, это origin

git remote add origin git@github.com:address.git # добавляем удаленный репозиторий, сокр. имя — origin

# если возникает ошибка добавления с сообщением о том, что origin «уже задан», то:

git remote rm origin # удаляем привязанный удалённый репозиторий

git remote add origin git@github.com:address.git # добавляем удаленный репозиторий, сокр. имя — origin

**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

# Ввод в командном режиме

:q! — выйти без сохранения

:wq — сохранить файл и выйти

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