Мануал для начала работы с гитом для Проекта.

Сразу извиняюсь что все так неопрятно, пишу из поезда, без мышки с touchpad-ом очень неудобно. Потом все отформатирую по феншую.

Перед и после прочтения данного мануала, обязательно ознакомтесь с моим постом в группе вк - CodeReview.

И так. Что такое гит я думаю уже все поняли.
Так же и поняли что такое GitHub.
Первое это базовый набор утилит для контроля версий.
Второе это надстройка на гитом реализующая платформу для хоста репозиториев.

Ещё раз пройдёмся по терминам.

Программист: жертва данного мануала.
Гит: распределенная система контроля версий.
Репозиторий: локальная База Данных хранящая в бинарном виде изменения между версиями.
Working Copy: папка с файлами проекта над которыми мы непосредственно работаем.
Remote: удалённый репозиторий с которым происходит синхронизация.
Дикий (bare) репозиторий: БД гита не в воркинг копи.
HTTPS: защищённый протокол для передачи гипертекста.
SSH: защищённый протокол удалённой консоли.
URL: уникальный идентификатор ресурса в сети.
Клонирование (clone): процесс скачивание удалённого репозитория локально.
Индекс: временное хранилище изменений для коммита.
Коммит (commit): запись изменений в локальный репозиторий из индекса а также объект представляющий изменения с прошедшие с родительского коммита.
HEAD: самый верхний коммит главной ветки.
Пуш (push): запись изменений в удалённый репозиторий из локального.
Fetch: Забор изменений из удалённого репозитория в локальный.
Checkout: применение изменений из локального репозитория в воркинг копи.
Пул (pull): Fetch + Checkout.
Upstream: дефолтный ремоут.
Brench (ветка): поток связанных комитов.
Tag: именованный коммит.
Merge: слияние файлов, бренчей или тэгов.
Fork (вилка): Clone из remote-а одного аккаунта на GitHub-е в другой. По другому remote clone.
IDE: Interactive Development Environment - среда разработки, например NetBeans.

PKI: Public/Private Key Infrastructure - криптографическая система открытых/закрытых ключей.
vim: Консольный текстовый редактор через который можно даже дебажить PHP.

Как обычно работают?
У каждого программиста есть репозиторий который он клонирует с remote-а.
Затем он пушит в этот же репозиторий.

Как мы работаем: каждый не только создаёт себе локальный репозиторий, но и remote на GitHub-е. Таким образом получается двух-фазовая синхронизация: между главным remote-ом Мальвины и дочерними в первой фазе и синхронизация между дочерними репозиториями и локальным на второй.
И так в обе стороны.

В первую очередь каждый гражданин должен создать аккаунт на GitHub-е.
Затем со своего аккаунта открывает наш проект который хостится на Мальвинином аккаунте. Там будет кнопочка Fork. Таким образом он клонирует репозиторий на свой аккаунт — но все это пока на сервере гита — удалённо.
Находясь на страничке Мальвины в опции клонирования на доступен URL лишь с префиксом HTTPS. Почему? Потому что пушить по HTTPS можно только с паролем, а по SSH можно по ключу.
В Мальвинин репозиторий мы не будем пушить так как по идее она не обязана давать нам свой пароль. По этому мы делаем fork, и уже из fork-а (удалённого клона) в опции клонирования нам доступен URL с префиксом SSH.
Вот это то что нам надо.
Теперь мы, после установки NetBeans-а или любого другого любимого IDE идём в папку проектов данного IDE и клонируем.
На первом этапе это лучше делать из консоли что бы понять смысл всех наших действий.
Итак:
git clone [ssh://fork`s_url]

Без фигурных скобок подставляя URL.
Важно!!! Он должен быть с префиксом SSH!!!

SSH дефолтный префикс, после клона, при попытке вывести список ремоутов, мы получим следующие:

```
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git remote -v
origin  git@github.com:olgapshen/calculator.git (fetch)
origin  git@github.com:olgapshen/calculator.git (push)
olga@olga-MacBookAir:~/NetBeansProjects/calculator$
```

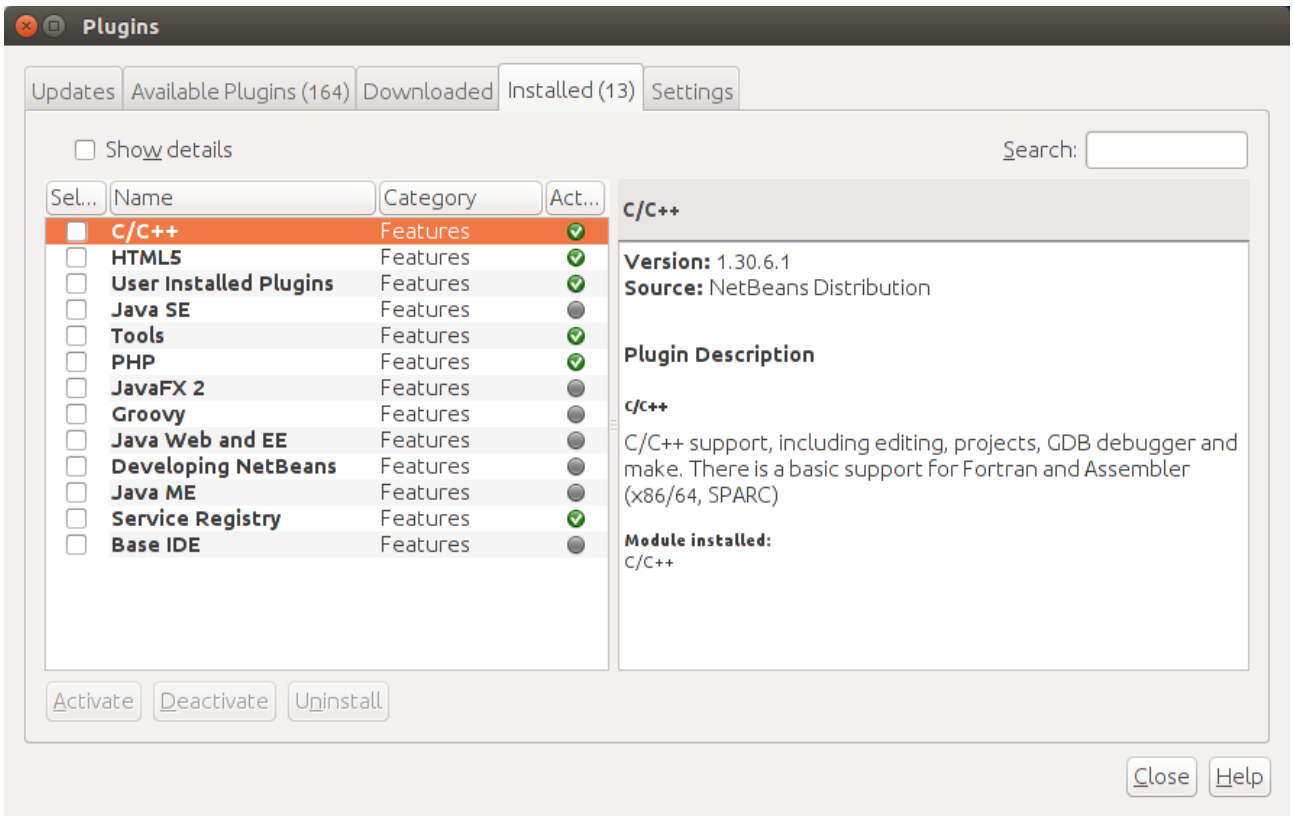
Как вы видите нет HTTPS.
Remote один и тот же фактически, представленный двумя интерфейсам: один для скачивания, один для пуша.

Это сделано для того что бы наш репозиторий мог бы быть звеном в цепи где скачивание идёт с одного места а пуш в другое.
Вспоминаем, что гит работает как разведка — принцип сети (без иронии).

Сразу настраиваем свои позывные как показано на скриншоте ниже:
После «user.» можно нажать tab 2 раза для подсказки.
git config — работа с конфигом гита.
--global — глобально а не только данный репозиторий.
user.name — имя на любом языке.
user.email — ваш email.
-l — вывести конфиг.
Остальное понятно.

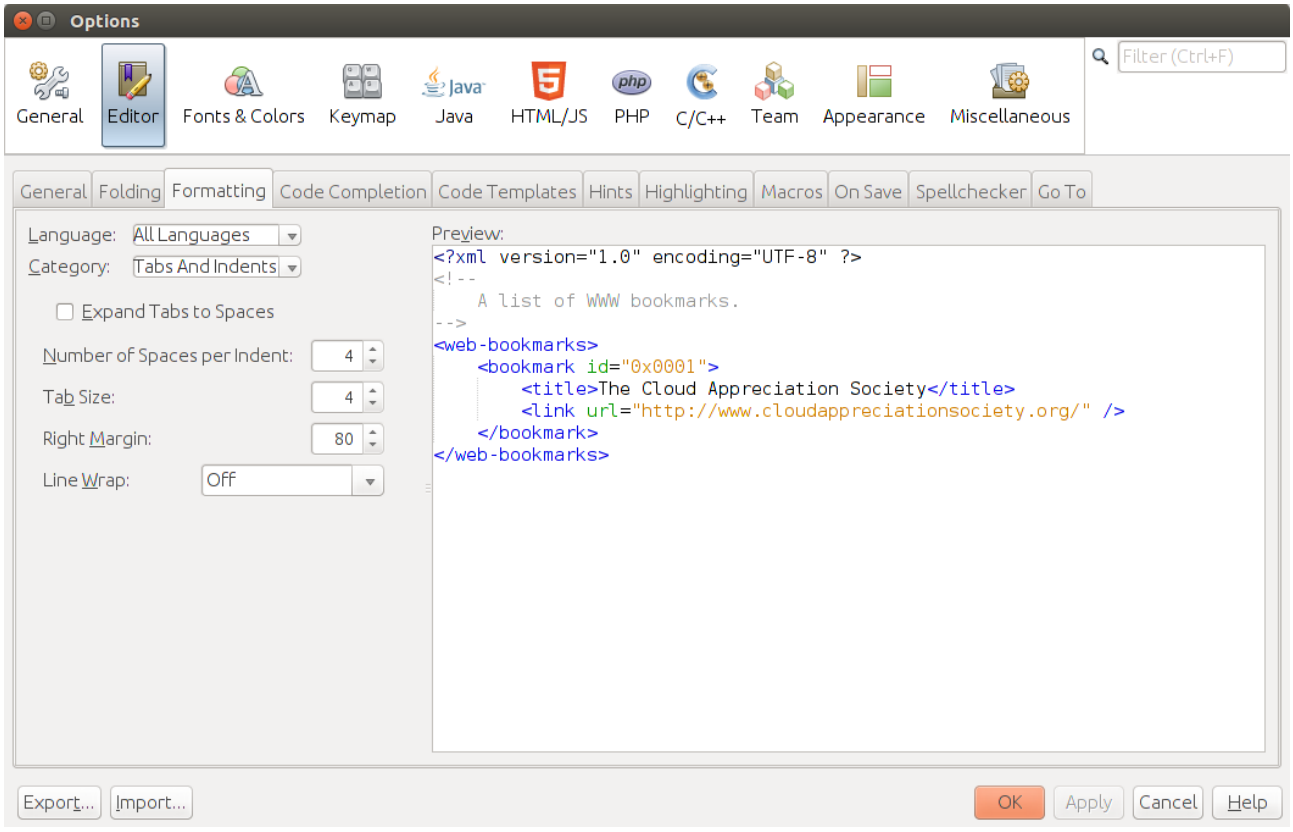
Причём, обязательно прописываем кто мы есть!!!
Что бы можно было знать кто что испортил, или кого хвалить.

Далее открываем проект. Если он открывается в NetBeans с сообщением о не установленных плагинах — это значит, что вы скачали NetBeans без поддержки PHP.
Скачайте среду со всеми опциями или доустановите плагин.
Tools → Plugins
Должно быть как на снимке ниже.

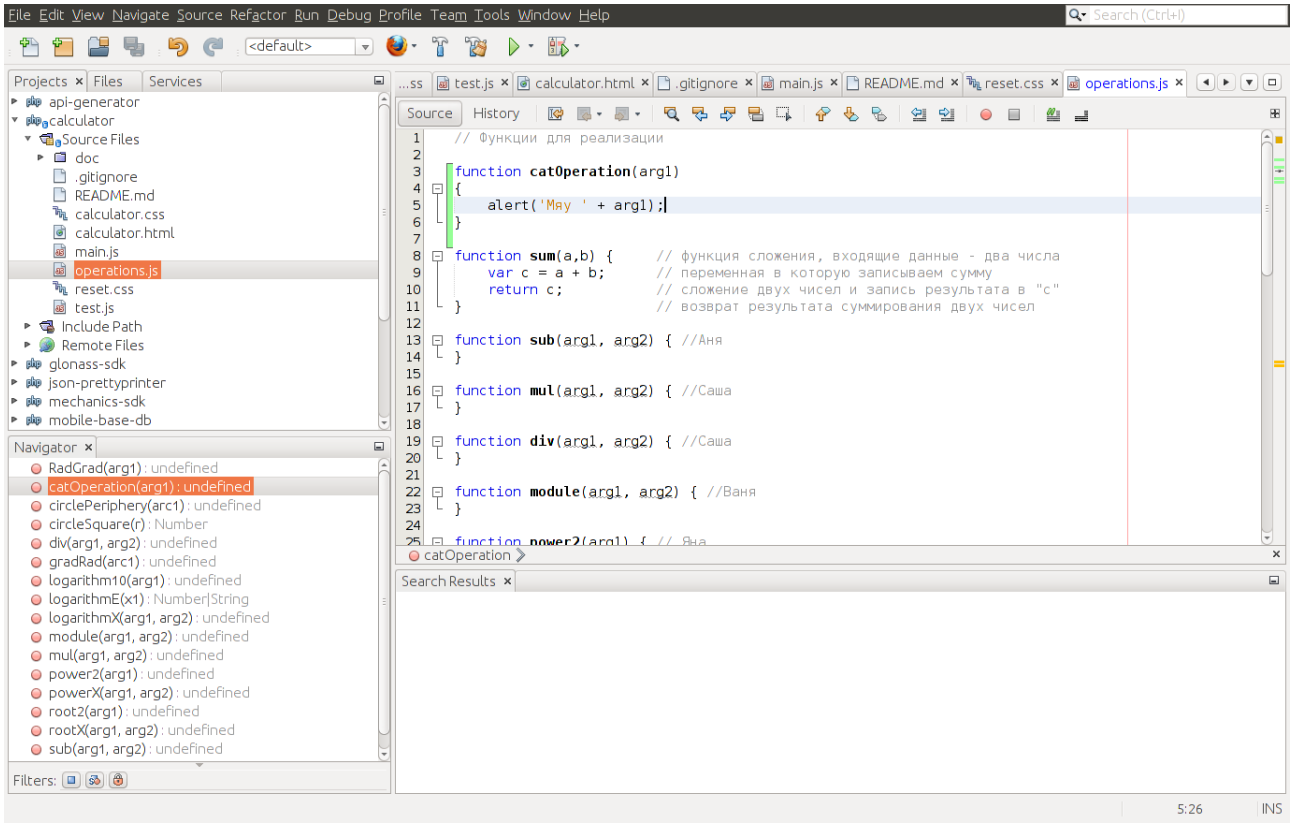


Далее мы пишем код, тестируем и доходим до момента что нам надо запушить.

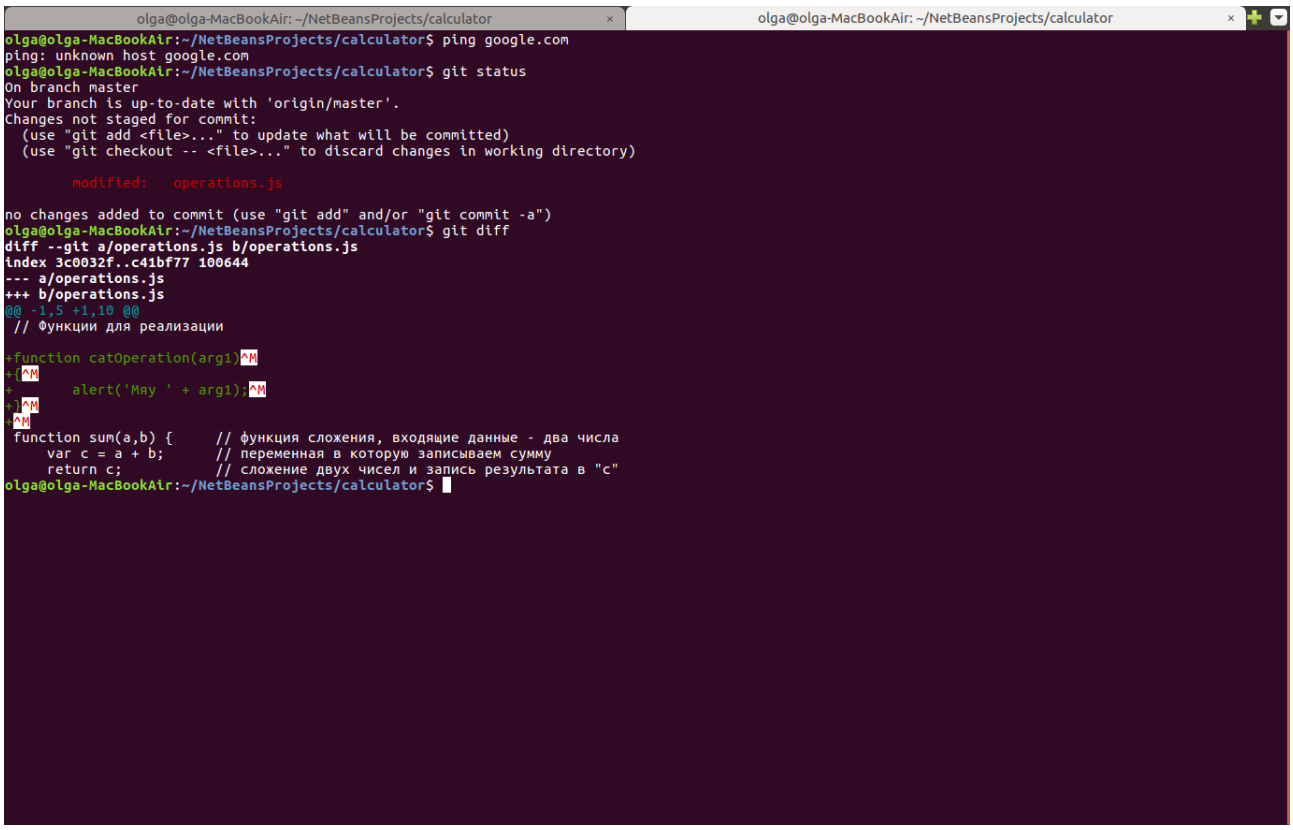
Обязательно в настройках для ВСЕХ языков прописываем использовать канонический таб равный 4 пробелам:



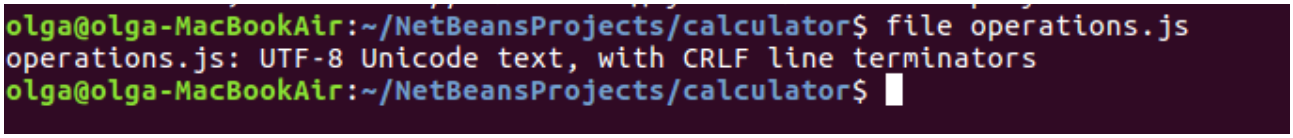
Далее, допустим я добавила функцию catOperation(arg1):



Конечно моя функция не по ТЗ которое нам написала Ненси, по этому потом её удалим.
Перед любой фиксацией изменений делаем git status что бы узнать что у нас идёт на фиксацию. Далее с помощью git diff получаем бриф изменений:

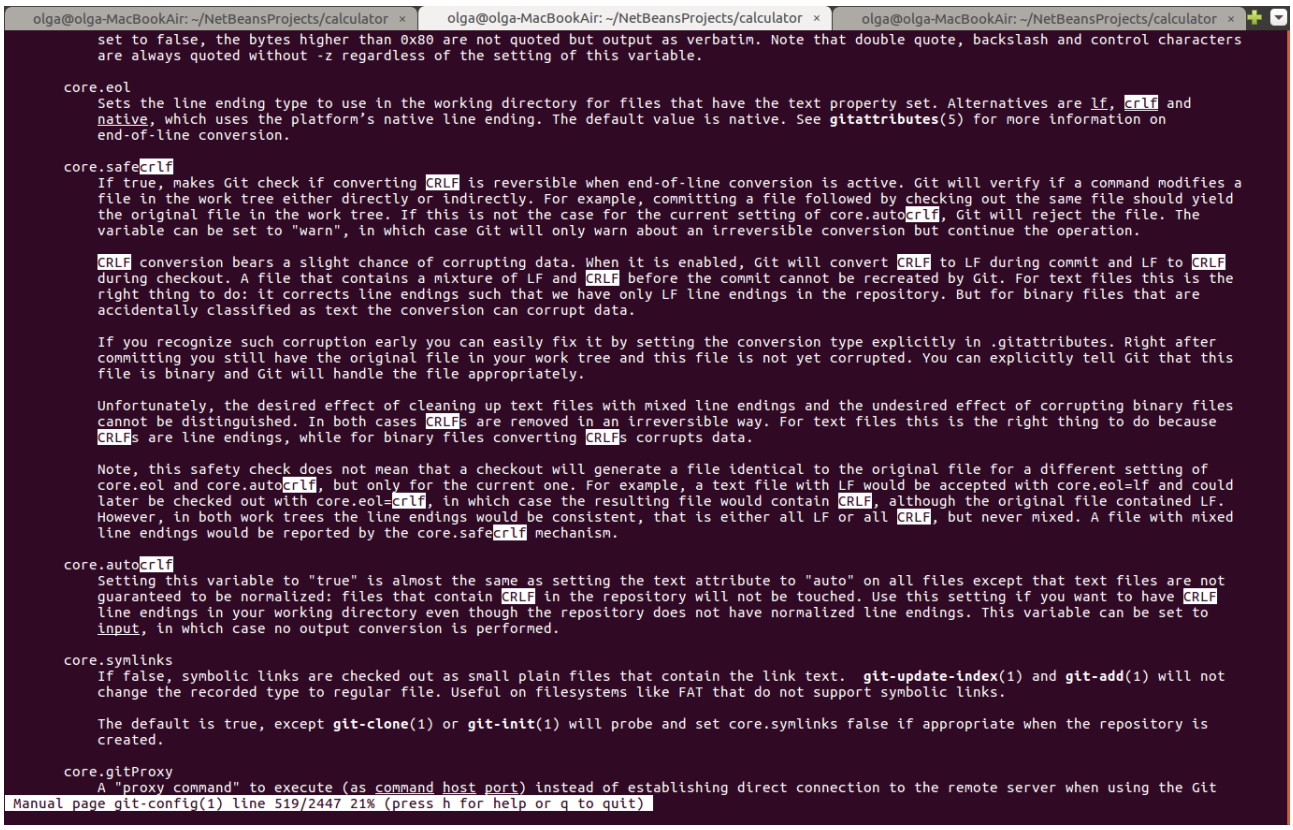


Обратите внимание на ^M — это свидетельствует о виндовском стиле окончания строк.
Подробнее об этом мы сможем узнать из сверх важной команды file:



Эта команда выводит всю инфу о формате файла.
CRLF это две комманды из потока символов таблицы ASCII (ОБЯЗАТЕЛЬНО гуглить по поводу ASCII) — Carret Return, Line Feed.
Что значит — каретку назад, спуститься.
Это в компьютерах осталось ещё с печатных машин.
Винда использует обе команды, а Линукс только LF.
В коде это зачастую выглядет как: «\r\n».
Как с этим бороться?
1. Изменить формат файла.
2. Сказать гиту — пуш с Линусовским окончанием строк.

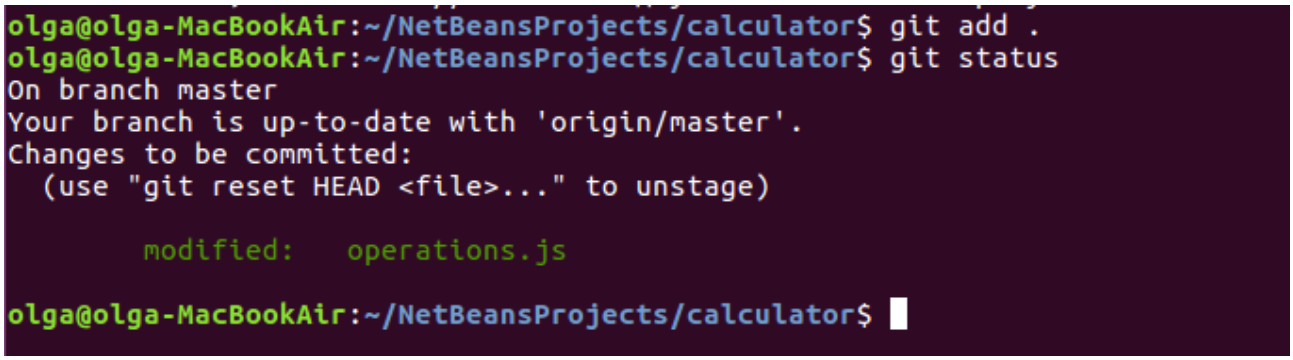
Вот, что мы видим по man git-config используя поиск по crlf:



Прочитав все это мы понимаем что нужна следующая конфигурация:
git config --global core.eol «lf»
git config --global core.savecrlf false

Выходим из комманды man с помощью клавиши q. Поиск: / и текст что мы ищем, enter. Далее - n (к следующему результату), ctrl + n к предыдущему. Все это мы усвоим лучше когда ознакомимся с vim-ом.

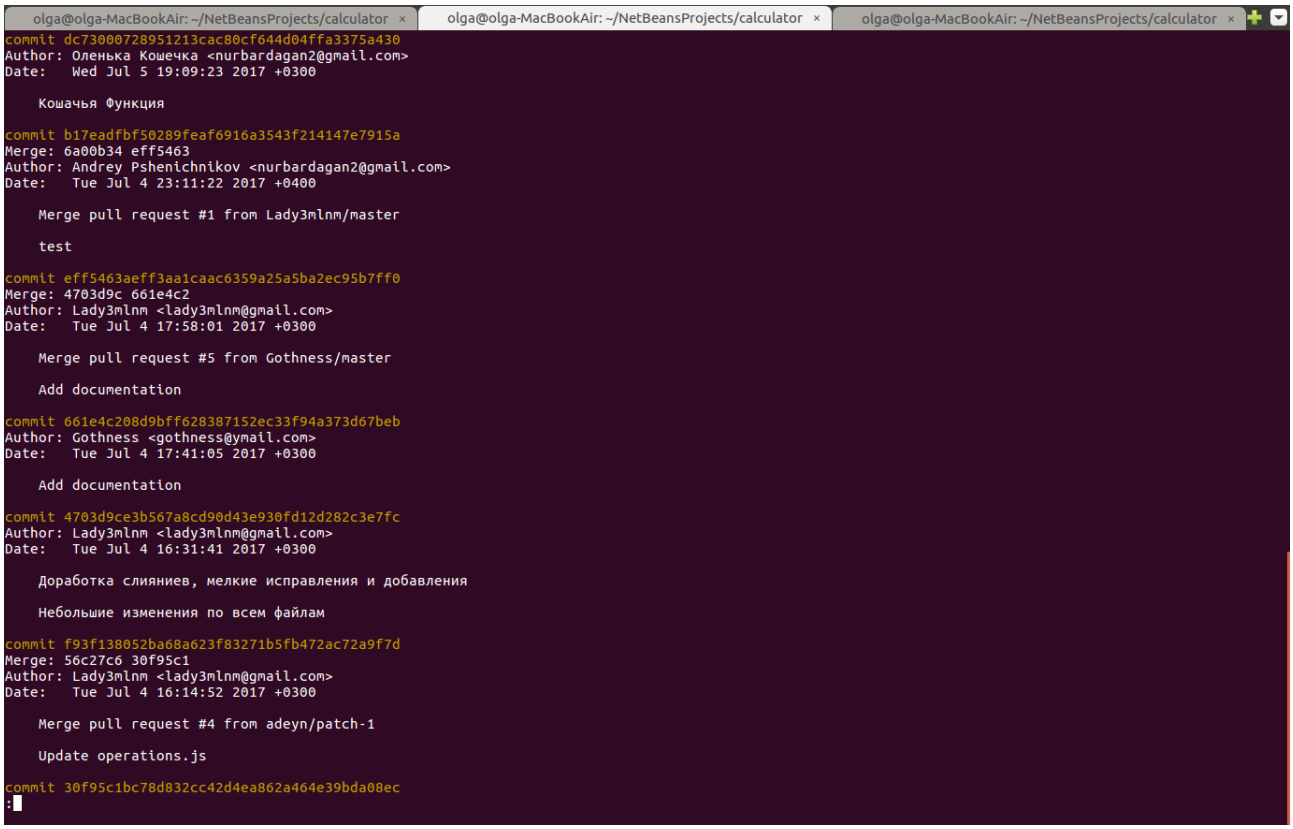
Далее делаем add что бы занести в индекс:



Точка значит - начиная с данной папки. И начинать надо с корневой папки проекта.
Далее статус говорит нам что мы таки занесли изменения в индекс.
Далее коммитим локально.

```
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git commit -m 'Кошачья Функция'
[master dc73000] Кошачья Функция
 1 file changed, 5 insertions(+)
olga@olga-MacBookAir:~/NetBeansProjects/calculator$
```

В -m пишем (ОБЯЗАТЕЛЬНО) комментарий в коммит.
Теперь в git log можно посмотреть историю коммитов включая наш последний:

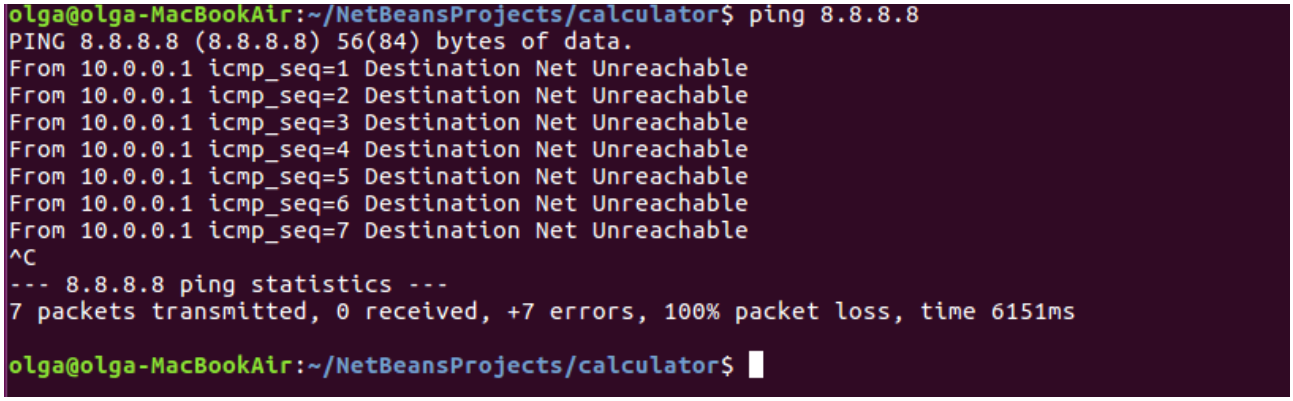


Мерзавец GitHub сохранил моё старое имя для моего предыдущего коммита который смерджила Мальвина после Pull Request-а.
Стрелочки вверх/вних на клаве дают нам возможность навигации, а выйти как и из man-а можно с помощью q.

Далее пушим.
Для успешного пуша можно проверить что есть ping на remote (гуглим что такое ping и icmp!!!):

```
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git remote -v
origin  git@github.com:olgapshen/calculator.git (fetch)
origin  git@github.com:olgapshen/calculator.git (push)
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ ping github.com
```

Команда висит, так как нет сети а я в поезде (((.
Ждём когда появится сеть.
А вот таким образом можно убедиться что действительно нет сети:



8.8.8.8 - это IP гугловских DNS-ов (гуглим что такое DNS!!!)

Я не буду ждать сети а перейду дальше.
Далее для успешного пуша вам понадобится кое что супер важное! Это пара ключей.))) Это очень важная тема которую я сейчас опишу.

Итак, PKI.
PKI расшифровывается как Public/Private Key Infrastruction - инфраструктура публичных и первичных ключей.
Для чего? Для безопасности.
Допустим мы куда то часто подключаемся, обычно это требует пароль. PKI избавляет нас от этого.
Второе, в общем, это шифрация и подпись и сверка подписи данных.
Итак, PKI позволяет нам 2 вещи:
1. Шифровка/Расшифровка
2. Подписать/Проверить

PKI подоразумевает наличие пары ключей - личный и публичный.
Личный только у субъекта использующего PKI, публичный он поставляет в публичный доступ.
С помощью личного ключа можно подписать данные так, что сверить подпись можно только с помощью публичного ключа из данной пары.
Зашифровать можно с помощью публичного ключа так что расшифровать можно только с помощью первичного из данной пары.

Допустим я отсылаю Андрею пакет данных, и хочу зашифровать его не передавая ключ шифровки.
Для этого я использую его публичный ключ в одностороннем алгоритме шифра.
Расшифровать с помощью этого же ключа нельзя!
Андрей получает пакет и спокойно расшифровывает его с помощью своего личного ключа.
Теперь я получила пакет данных от Андрея, и хочу убедиться что это именно от него и данные не изменились.

Для этого Андрей одностороннем образом хеширует данные, например с помощью алгоритма хеширования md5 (читаем что такое md5 в Википедии!!!).
Далее он шифрует хеш с помощью своего первичного ключа односторонним алгоритмом шифра (все алгоритмы тут одностороние).
Далее я получаю пакет с незашифрованными данными и зашифрованным хешом именуемым - дигитальной подписью.
Я так же хеширую данные.
Далее расшифровать я могу его зашифрованный хеш только с помощью его публичного ключа.
Далее я сверяю полученный мной хеш с хешом полученным из расшифровки подписи.
Если совпадают, значит пакет от Андрея и не был подделан.
Вот основы PKI.
Мы используем его в SSH-ных ремоутах что бы не надо было постоянно вводить пароль.

В первую очередь получим пару.
Пара по умолчанию будет в папке ~/.ssh где ~ это домашний каталог.
У меня там уже есть ключик и я его бекапю от греха подальше:

```
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ cd ~/.ssh
olga@olga-MacBookAir:~/.ssh$ ls -la
total 16
drwx------  2 olga olga 4096 июн 30 00:56 .
drwxr-xr-x 38 olga olga 4096 июл  5 19:06 ..
-rwx------  1 olga olga 1679 июн 30 00:55 id_rsa
-rwx------  1 olga olga 3275 июл  4 22:06 known_hosts
olga@olga-MacBookAir:~/.ssh$ mv id_rsa id_rsa_bak
olga@olga-MacBookAir:~/.ssh$ ls -la
total 16
drwx------  2 olga olga 4096 июл  5 19:31 .
drwxr-xr-x 38 olga olga 4096 июл  5 19:06 ..
-rwx------  1 olga olga 1679 июн 30 00:55 id_rsa_bak
-rwx------  1 olga olga 3275 июл  4 22:06 known_hosts
olga@olga-MacBookAir:~/.ssh$
```

Далее я генерирую пару ключей простой коммандой ssh-keygen:

```
olga@olga-MacBookAir:~/.ssh$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/olga/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/olga/.ssh/id_rsa.
Your public key has been saved in /home/olga/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:H/QOr1UWttrQQCA3CTnRzVnQl3HDj/3niGSVLuQnzT4 olga@olga-MacBookAir
The key's randomart image is:
+---[RSA 2048]----+
|        +=+=o=oo+|
|        oooo+ .o+|
|         .. . o=.|
|         . ..++oo|
|        S ooo== .|
|         . =**= o|
|          .o=*.o.|
|           o. E .|
|          .    . |
+----[SHA256]-----+
olga@olga-MacBookAir:~/.ssh$
```

Соглашаемся со всем что нам говорят, постоянно нажимая enter.

Помните, мы на первом занятии говорили о псевдослучайных числах и проблемах детерминизма и рандомальности. Тут это понимание нам пригодится.
Картинка это графическое представление приближенного математического хаоса использованного для генерации ключа.
На программе putty надо рандомально двигать мышкой))) А производители кредитных карт используют изменчивую погоду.
(Читаем про putty те кто ещё на винде!!!)

У нас появились ключи. Обязательно присваиваем всей папке права 700, иначе ничего не будет работать (гуглим про chmod!!!):

```
olga@olga-MacBookAir:~/.ssh$ ls -la
total 24
drwx------  2 olga olga 4096 июл  5 19:32 .
drwxr-xr-x 38 olga olga 4096 июл  5 19:06 ..
-rw-------  1 olga olga 1675 июл  5 19:32 id_rsa
-rwx------  1 olga olga 1679 июн 30 00:55 id_rsa_bak
-rw-r--r--  1 olga olga  402 июл  5 19:32 id_rsa.pub
-rwx------  1 olga olga 3275 июл  4 22:06 known_hosts
olga@olga-MacBookAir:~/.ssh$ sudo chmod -R 700 .
[sudo] password for olga:
olga@olga-MacBookAir:~/.ssh$ ls -la
total 24
drwx------  2 olga olga 4096 июл  5 19:32 .
drwxr-xr-x 38 olga olga 4096 июл  5 19:06 ..
-rwx------  1 olga olga 1675 июл  5 19:32 id_rsa
-rwx------  1 olga olga 1679 июн 30 00:55 id_rsa_bak
-rwx------  1 olga olga  402 июл  5 19:32 id_rsa.pub
-rwx------  1 olga olga 3275 июл  4 22:06 known_hosts
olga@olga-MacBookAir:~/.ssh$
```

Первичная часть в id_rsa - можно посмотреть содержимое коммандой
cat id_rsa

Публичная в id_rsa.pub

Видим что созданы они одновременно.
В known_hosts отпечатки пальцев серверов куда мы разрешаем системе подключатся по ssh без запроса внести отпечаток пальцев данного сервера в этот самый файл.
Просматриваем все файлы.

Теперь в независимости есть ли у нас публичная часть ключа или нет генерируем её из личной:

```
olga@olga-MacBookAir:~/.ssh$ ssh-keygen -y -f id_rsa
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCfFeU+WqVpW3u/m4oHAkgZq9T2Z38otC79wxd4OzpSqWhrM5LQjOrLt6HE+h/cUtz94cbR2Fhtz9OlJRni3vAbrulxSoBAyZooyQWHSIciWslZ8FP
RQ2cAAGBzqbDAzrgZ8FGPPH4ZdoJmdjzE8s9T09f7YNGUSZi+Y8+/psyVShcsjXXXQEQ9orAbDGma+OuAFLC4T28o6AqcfK4PgO68etV+bPB4K0uvKLEfALsVjocfNiyBMlGz7yh+fsp7BAgvVe1c3p
Tzfp1MEwUKS7SrAHNBqTiJOz9c7qiJrsNv6t3KgIFPvTQcFgXeBnJDe/foEKgApVF+9GD/nopAoVOX
olga@olga-MacBookAir:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCfFeU+WqVpW3u/m4oHAkgZq9T2Z38otC79wxd4OzpSqWhrM5LQjOrLt6HE+h/cUtz94cbR2Fhtz9OlJRni3vAbrulxSoBAyZooyQWHSIciWslZ8FP
RQ2cAAGBzqbDAzrgZ8FGPPH4ZdoJmdjzE8s9T09f7YNGUSZi+Y8+/psyVShcsjXXXQEQ9orAbDGma+OuAFLC4T28o6AqcfK4PgO68etV+bPB4K0uvKLEfALsVjocfNiyBMlGz7yh+fsp7BAgvVe1c3p
Tzfp1MEwUKS7SrAHNBqTiJOz9c7qiJrsNv6t3KgIFPvTQcFgXeBnJDe/foEKgApVF+9GD/nopAoVOX olga@olga-MacBookAir
olga@olga-MacBookAir:~/.ssh$
```

Можно распечатать id_rsa.pub и убедиться что они в своей базовой части (до комментария после пробела) идентичны.

Прописываем его в GitHub-е (это было в CodeReview).

Сети до сих пор нет))) так что продолжаю без скриншотов с GitHub-а.
Потом вставлю.
И так, на GitHub-е заходим в настройки профиля, ищем ключи (ещё раз, смотрим галлерею - CodeReview) и добавляем наш публичный ключ.

Далее просто делаем git push.
Git скорее всего попросит у нас указать дефолнтый ремоут (upstream),
по этому запускаем первый раз команду таким образом:

git push --set-upstream origin master

В следующие разы делаем просто git push.

Таким образом, если всё пройдёт без эксцессов, на нашем форке Мальвининого репозитория (на нашем аккаунте) окажутся нужные нам изменения.
Далее делаем pull request к Мальвине.
Так же пишем комментарий.
Мальвина получает об этом уведомление. И великодушно мерджит ваши изменения к ней - в наиосновнейший репозиторий проекта.
Далее когда много людей ещё туда запушат с помощью своих pull request-ов, мы делаем pull request от Мальвины к самим себе, одобряя это сами же затем.

При чем по умолчанию GitHub думает что мы хотим смерджить что то к Мальвине от нас, по этому когда он поймёт что у Мальвины ветка на пару коммитов вперёд, он предложит нам сделать reverse base.
Соглашаемся и мерджим от Мальвины к нам.
Далее одобряем инициированный нами же pull request к нам же.
Далее идём в локальный репозиторий и просто делаем git pull.
Если первый раз и даже не делали push то: git pull origin master.

Вот так)))
И это вечное колесо Сансары в котором мы будем крутиться, разве что в дальнейшим будем сразу коммитеть в главный Мальвинин ремоут.
Но это потом)

Муррр.