

Мануал для начала работы с гитом (часть 2)

(Создан специально для участников группы TechnoFractal)

Итак, что такое Git, я думаю, уже все поняли. Так же поняли, и что такое GitHub.

Git - это базовый набор утилит для контроля версий.

GitHub - это надстройка на гитом, реализующая платформу для хоста репозиториев.

Ещё раз пройдемся по терминам.

Программист: жертва данного мануала.

Git: распределенная система контроля версий.

Repository (Репозиторий): локальная База Данных, хранящая в бинарном виде изменения между версиями.

Working Copy: папка с файлами проекта, над которыми мы непосредственно работаем.

Index (Индекс, Staging Area): временное хранилище изменений для коммита.

Remote: удалённый репозиторий, с которым происходит синхронизация.

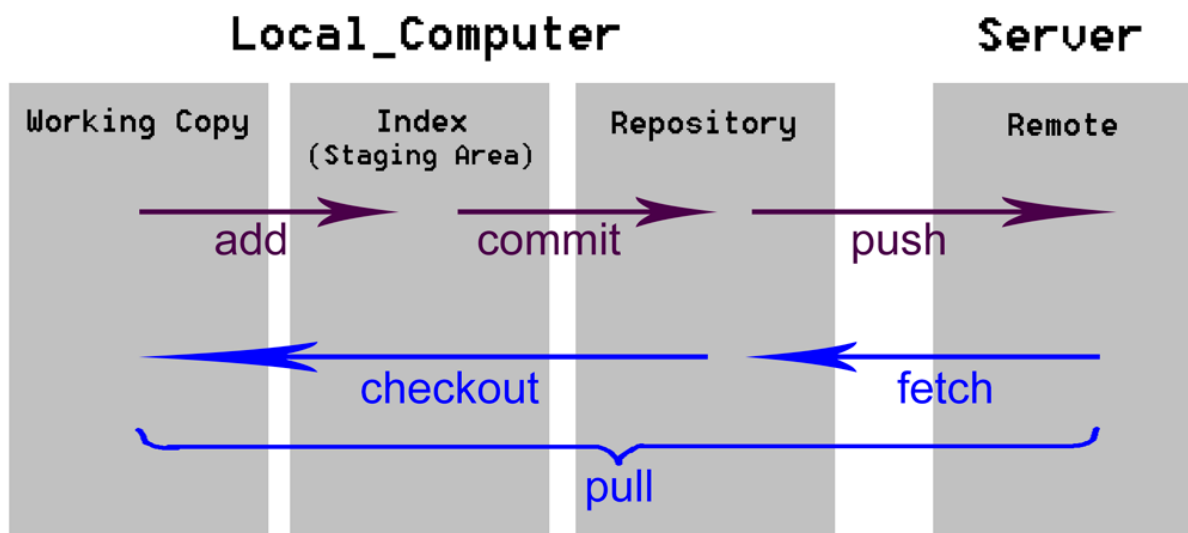
Commit (коммит): запись изменений в локальный репозиторий из индекса, а также объект, представляющий изменения, произошедшие с родительского коммита.

Push (Пуш): запись изменений в удалённый репозиторий из локального.

Fetch: Забор изменений из удалённого репозитория в локальный.

Checkout: применение изменений из локального репозитория в Working Copy.

Pull (Пул): Fetch + Checkout.



Bare repository (Дикий репозиторий): БД гита не в Working Copy.

Merge: слияние файлов, брэнчей или тэгов.

Clone (клонирование): процесс скачивание удалённого репозитория локально.

HEAD: самый верхний коммит главной ветки.

Upstream: дефолтный ремоут.

Branch (ветка): поток связанных комитов.

Tag: именованный коммит.

Fork (вилка): Clone из remote-а одного аккаунта на GitHub-е в другой. По-другому, remote clone.

HTTPS: защищённый протокол для передачи гипертекста.

SSH: защищённый протокол удалённой консоли.

URL: уникальный идентификатор ресурса в сети.

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 Мальвининском аккаунте. Там будет кнопка Fork. Таким образом он клонирует репозиторий на свой аккаунт — но все это пока на сервере гита, удалённо.

Находясь на страничке Мальвины в опции клонирования нам 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. Поэтому 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, чтобы 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 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!!!

Чтобы можно было знать, кто что испортил или кого хвалить.

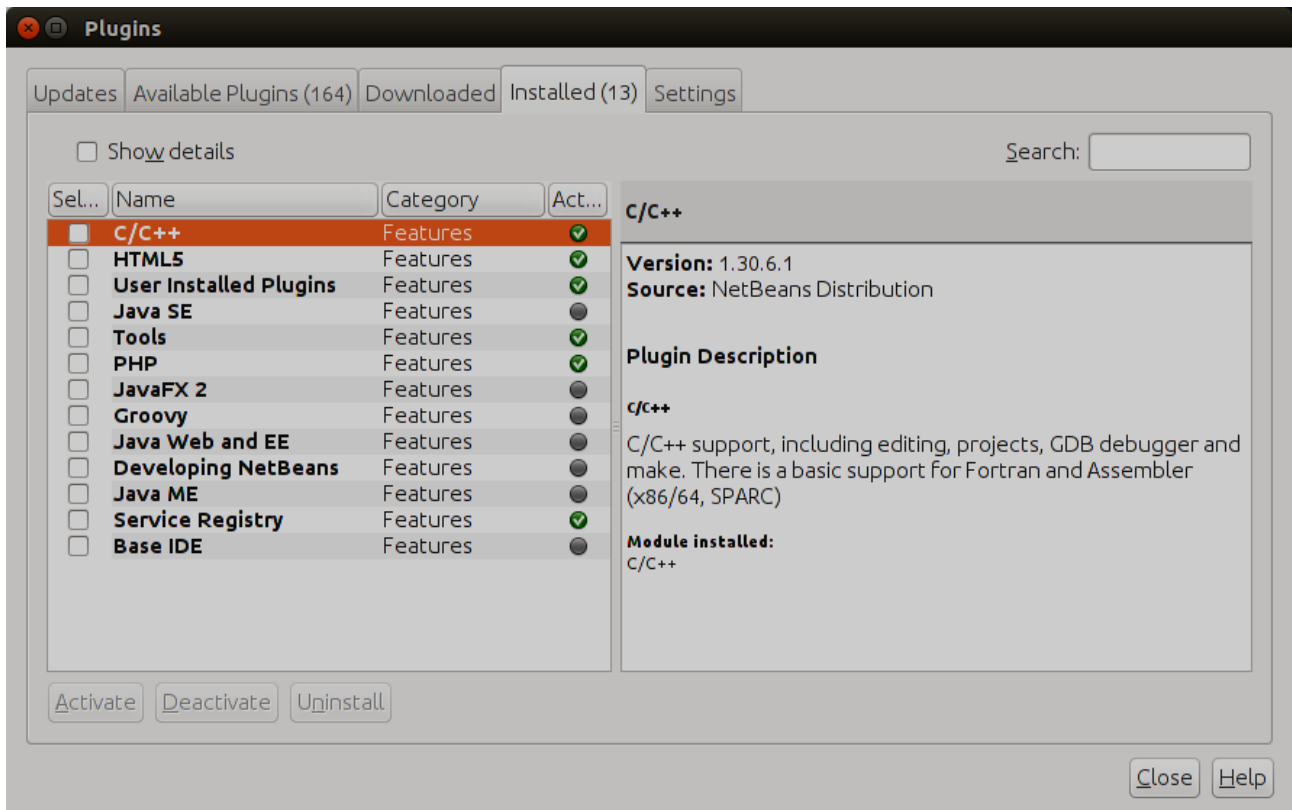
```
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ man git-config
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git config --global user.
user.email      user.name      user.signingkey
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git config --global user.name "Оленька Кошечка"
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git config --global user.email nurbardagan2@gmail.com
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git config -l
user.email=nurbardagan2@gmail.com
user.name=Оленька Кошечка
push.default=simple
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=git@github.com:olgapshen/calculator.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
olga@olga-MacBookAir:~/NetBeansProjects/calculator$
```

Далее открываем проект. Если он открывается в NetBeans с сообщением о неустановленных плагинах — это значит, что вы скачали NetBeans без поддержки PHP.

Скачайте среду со всеми опциями или доустановите плагин.

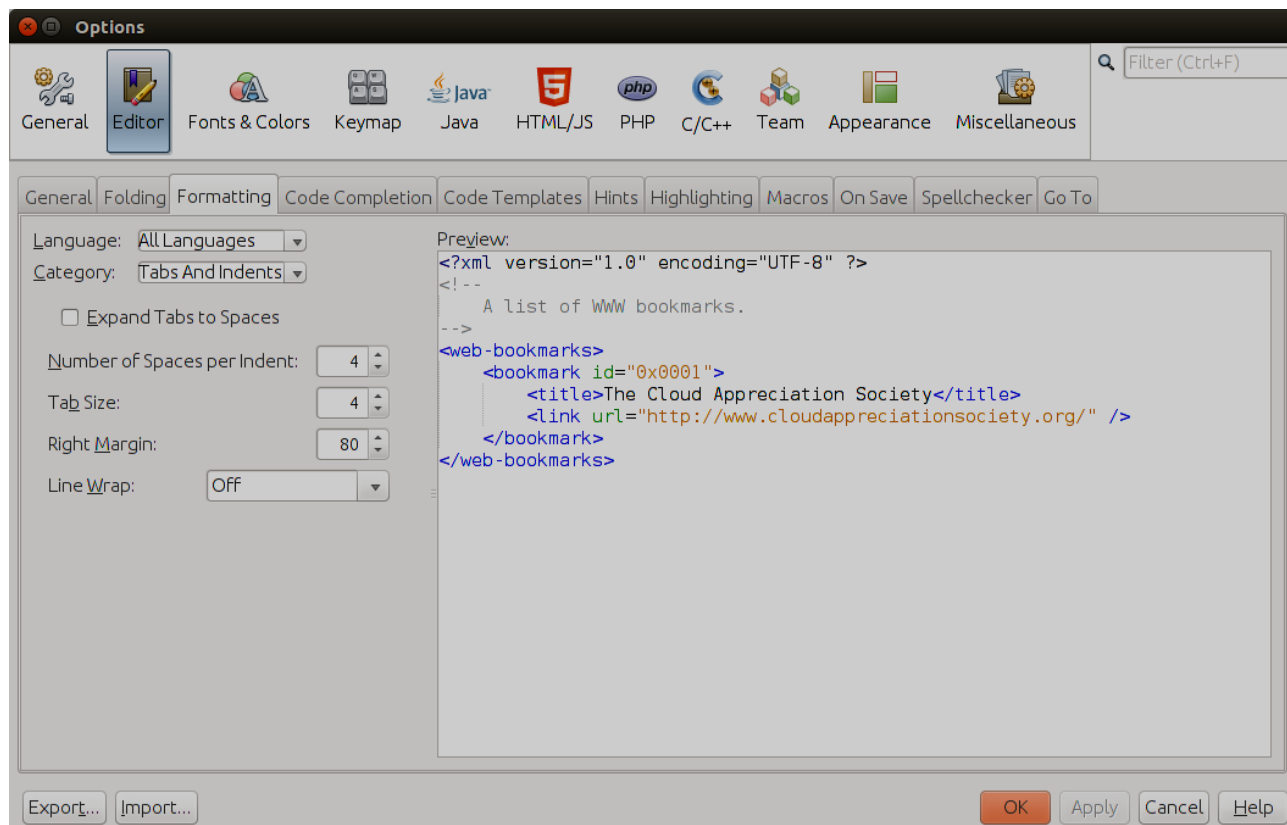
Tools → Plugins

Должно быть как на снимке ниже.

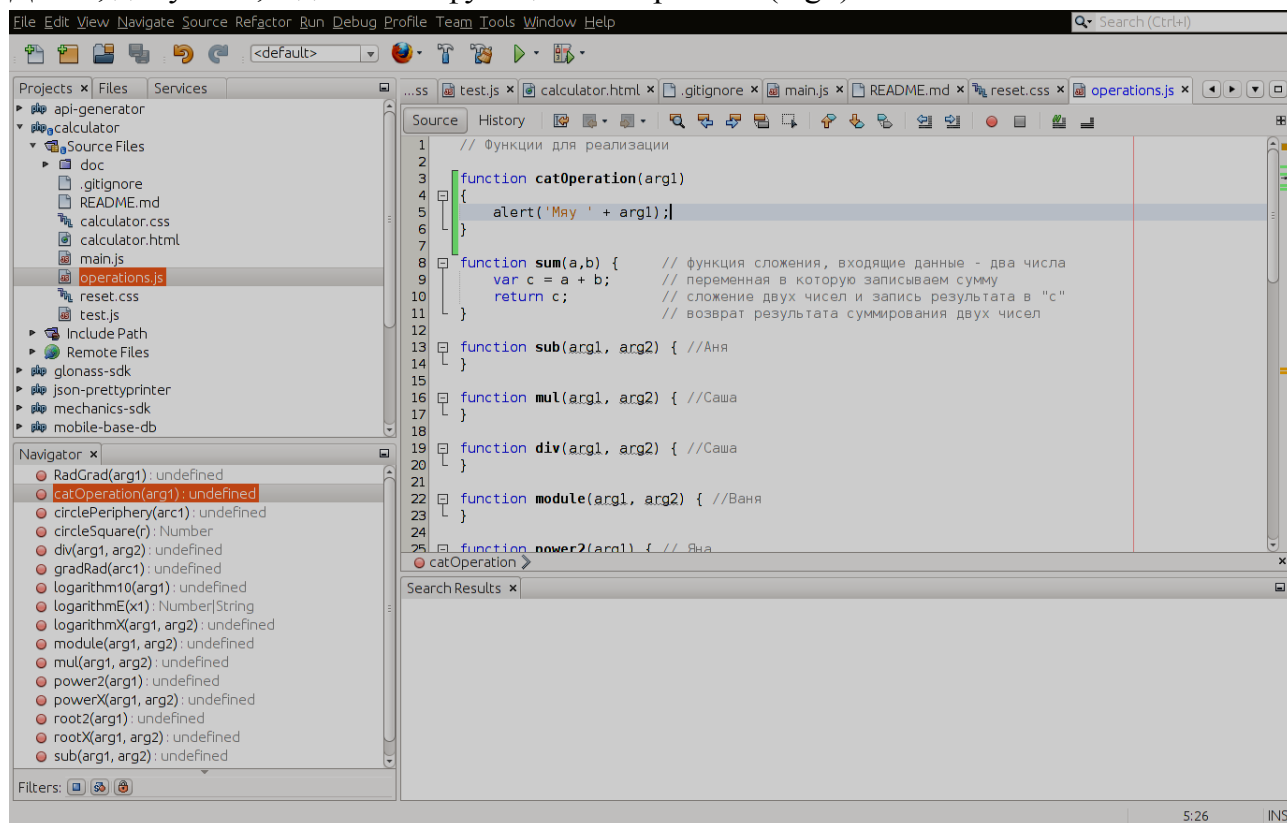


Далее мы пишем код, тестируем и доходим до момента, что нам надо запустить.

Обязательно в настройках для ВСЕХ языков прописываем использование канонического таб, равного 4 пробелам:



Далее, допустим, я добавила функцию catOperation(arg1):



Конечно, моя функция не по ТЗ, которое нам написала Ненси, поэтому 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:

```
olga@olga-MacBookAir: ~/NetBeansProjects/calculator
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ ping google.com
ping: unknown host google.com
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   operations.js

no changes added to commit (use "git add" and/or "git commit -a")
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git diff
diff --git a/operations.js b/operations.js
index 3c0032f..c41bf77 100644
--- a/operations.js
+++ b/operations.js
@@ -1,5 +1,10 @@
 // Функции для реализации

+function catOperation(arg1)^M
+{^M
+    alert('Мяу ' + arg1);^M
+}^M
function sum(a,b) {           // функция сложения, входящие данные - два числа
    var c = a + b;           // переменная в которую записываем сумму
    return c;                // сложение двух чисел и запись результата в "c"
olga@olga-MacBookAir:~/NetBeansProjects/calculator$
```

Обратите внимание на `^M` — это свидетельствует о виндовском стиле окончания строк.

Подробнее об этом мы сможем узнать из сверхважной команды `file`:

```
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ file operations.js
operations.js: UTF-8 Unicode text, with CRLF line terminators
olga@olga-MacBookAir:~/NetBeansProjects/calculator$
```

Эта команда выводит всю инфу о формате файла.

`CRLF` - это две команды 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.

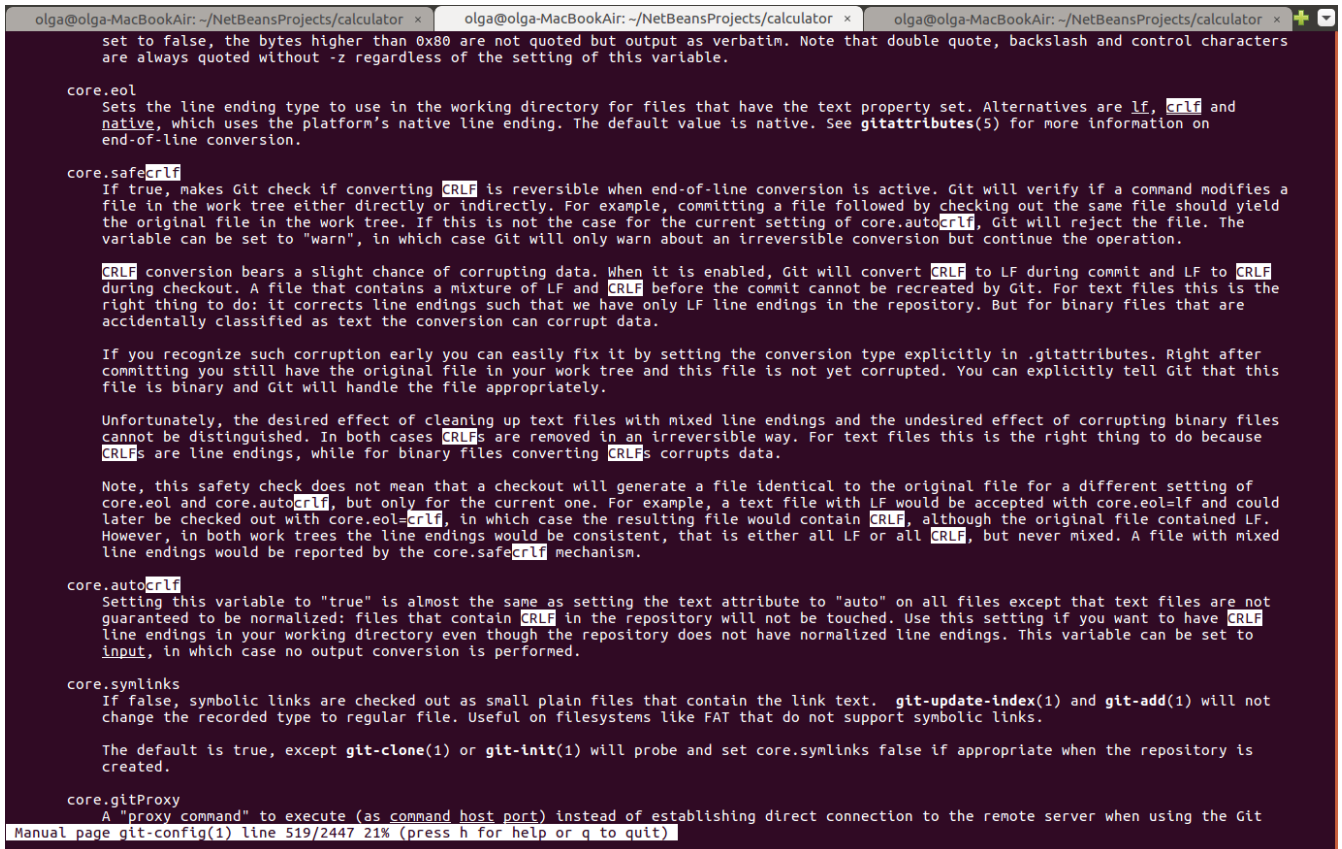
Винда использует обе команды, а Линукс только LF.

В коде это зачастую выглядит как: «\r\n».

Как с этим бороться?

1. Изменить формат файла.
2. Сказать гиту — пуш с Линусовским окончанием строк.

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



```
olga@olga-MacBookAir: ~/NetBeansProjects/calculator * olga@olga-MacBookAir: ~/NetBeansProjects/calculator * olga@olga-MacBookAir: ~/NetBeansProjects/calculator *
set to false, the bytes higher than 0x80 are not quoted but output as verbatim. Note that double quote, backslash and control characters
are always quoted without -z regardless of the setting of this variable.

core.eol
Sets the line ending type to use in the working directory for files that have the text property set. Alternatives are lf, crlf and
native, which uses the platform's native line ending. The default value is native. See gitattributes(5) for more information on
end-of-line conversion.

core.safecrlf
If true, makes Git check if converting CRLF is reversible when end-of-line conversion is active. Git will verify if a command modifies a
file in the work tree either directly or indirectly. For example, committing a file followed by checking out the same file should yield
the original file in the work tree. If this is not the case for the current setting of core.autocrlf, Git will reject the file. The
variable can be set to "warn", in which case Git will only warn about an irreversible conversion but continue the operation.

CRLF conversion bears a slight chance of corrupting data. When it is enabled, Git will convert CRLF to LF during commit and LF to CRLF
during checkout. A file that contains a mixture of LF and CRLF before the commit cannot be recreated by Git. For text files this is the
right thing to do: it corrects line endings such that we have only LF line endings in the repository. But for binary files that are
accidentally classified as text the conversion can corrupt data.

If you recognize such corruption early you can easily fix it by setting the conversion type explicitly in .gitattributes. Right after
committing you still have the original file in your work tree and this file is not yet corrupted. You can explicitly tell Git that this
file is binary and Git will handle the file appropriately.

Unfortunately, the desired effect of cleaning up text files with mixed line endings and the undesired effect of corrupting binary files
cannot be distinguished. In both cases CRLFs are removed in an irreversible way. For text files this is the right thing to do because
CRLFs are line endings, while for binary files converting CRLFs corrupts data.

Note, this safety check does not mean that a checkout will generate a file identical to the original file for a different setting of
core.eol and core.autocrlf, but only for the current one. For example, a text file with LF would be accepted with core.eol=lf and could
later be checked out with core.eol=crlf, in which case the resulting file would contain CRLF, although the original file contained LF.
However, in both work trees the line endings would be consistent, that is either all LF or all CRLF, but never mixed. A file with mixed
line endings would be reported by the core.safecrlf mechanism.

core.autocrlf
Setting this variable to "true" is almost the same as setting the text attribute to "auto" on all files except that text files are not
guaranteed to be normalized: files that contain CRLF in the repository will not be touched. Use this setting if you want to have CRLF
line endings in your working directory even though the repository does not have normalized line endings. This variable can be set to
input, in which case no output conversion is performed.

core.symlinks
If false, symbolic links are checked out as small plain files that contain the link text. git-update-index(1) and git-add(1) will not
change the recorded type to regular file. Useful on filesystems like FAT that do not support symbolic links.

The default is true, except git-clone(1) or git-init(1) will probe and set core.symlinks false if appropriate when the repository is
created.

core.gitProxy
A "proxy command" to execute (as command host port) instead of establishing direct connection to the remote server when using the Git
Manual page git-config(1) line 519/2447 21% (press h for help or q to quit)
```

Прочитав всё это, мы понимаем, что нужна следующая конфигурация:

```
git config --global core.eol «lf»
git config --global core.safecrlf false
```

Выходим из команды `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`, чтобы занести в индекс:

```
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git add .
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   operations.js

olga@olga-MacBookAir:~/NetBeansProjects/calculator$
```

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:

```
olga@olga-MacBookAir: ~/NetBeansProjects/calculator x olga@olga-MacBookAir: ~/NetBeansPr
commit dc73000728951213cac80cf644d04ffa3375a430
Author: Оленька Кошечка <nurbardagan2@gmail.com>
Date:   Wed Jul 5 19:09:23 2017 +0300

    Кошачья Функция

commit b17eadfbf50289feaf6916a3543f214147e7915a
Merge: 6a00b34 eff5463
Author: Andrey Pshenichnikov <nurbardagan2@gmail.com>
Date:   Tue Jul 4 23:11:22 2017 +0400

    Merge pull request #1 from Lady3mlnm/master

    test

commit eff5463aeff3aa1caac6359a25a5ba2ec95b7ff0
Merge: 4703d9c 661e4c2
Author: Lady3mlnm <lady3mlnm@gmail.com>
Date:   Tue Jul 4 17:58:01 2017 +0300

    Merge pull request #5 from Gothness/master

    Add documentation

commit 661e4c208d9bfff628387152ec33f94a373d67beb
Author: Gothness <gothness@ymail.com>
Date:   Tue Jul 4 17:41:05 2017 +0300

    Add documentation

commit 4703d9ce3b567a8cd90d43e930fd12d282c3e7fc
Author: Lady3mlnm <lady3mlnm@gmail.com>
Date:   Tue Jul 4 16:31:41 2017 +0300

    Доработка слияний, мелкие исправления и добавления

    Небольшие изменения по всем файлам

commit f93f138052ba68a623f83271b5fb472ac72a9f7d
Merge: 56c27c6 30f95c1
Author: Lady3mlnm <lady3mlnm@gmail.com>
Date:   Tue Jul 4 16:14:52 2017 +0300

    Merge pull request #4 from adeyn/patch-1

    Update operations.js

commit 30f95c1bc78d832cc42d4ea862a464e39bda08ec
:█
```

Мерзавец GitHub сохранил моё старое имя для моего предыдущего коммита, который смерджила Мальвина после Pull Request-a.

Стрелочки вверх/вниз на клавише дают нам возможность навигации, а выйти, как и из `map-a`, можно с помощью `q`.

Далее пушим.

Для успешного пуша можно проверить, что есть ping на remote (зуглим, 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:

```
olga@olga-MacBookAir:~/NetBeansProjects/calculator$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Net Unreachable
From 10.0.0.1 icmp_seq=2 Destination Net Unreachable
From 10.0.0.1 icmp_seq=3 Destination Net Unreachable
From 10.0.0.1 icmp_seq=4 Destination Net Unreachable
From 10.0.0.1 icmp_seq=5 Destination Net Unreachable
From 10.0.0.1 icmp_seq=6 Destination Net Unreachable
From 10.0.0.1 icmp_seq=7 Destination Net Unreachable
^C
--- 8.8.8.8 ping statistics ---
7 packets transmitted, 0 received, +7 errors, 100% packet loss, time 6151ms

olga@olga-MacBookAir:~/NetBeansProjects/calculator$
```

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

Я не буду ждать сети, а перейду дальше.

Далее для успешного пуша вам понадобится кое-что суперважное! Это пара ключей.

))) Это очень важная тема, которую я сейчас опишу.

Итак, PKI.

PKI расшифровывается как Public/Private Key Infrastructure - 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 подразумевает наличие пары ключей - личный и публичный.

Личный ключ только у субъекта, использующего PKI.

Публичный ключ выкладывается в публичный доступ.

С помощью личного 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 односторонним 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 односторонние*).

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, чтобы 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 бекаю 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 командой `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/Q0r1UWttrQQCA3CTnRzVnQl3HDj/3niGSVLuQnzT4 olga@olga-MacBookAir
The key's randomart image is:
+---[RSA 2048]---+
|      +==+0=00+ |
|      0000+ .0+ |
|      .. . 0=. |
|      . . .++00 |
|      S 000== . |
|      . =**= 0 |
|      .0=* .0. |
|      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 командой
`cat id_rsa`

Публичная в id_rsa.pub

Видим, что созданы они одновременно.

В known_hosts отпечатки пальцев серверов, куда мы разрешаем системе подключаться по ssh без запроса внести отпечаток пальцев данного сервера в этот самый файл.

Просматриваем все файлы.

Теперь вне зависимости, 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 AAAAB3NzaC1yc2EAAAADAQABAAQCFFeU+HqVpW3u/m4oHAKgZq9T2Z3BotC79wxd40zpSqWhrM5LQj0rLt6HE+h/cutZ94cbr2Fhtz90LJRnI3vAbrulx5o8AyZooyQWHSiciwslZ8FP
RQ2cAAGBzqbDAzrgZ8FGPPH4ZdoJmdjzE8s9T09f7YNGUSZi+Y8+/psyVShcsjXXXQE9orAbdGma+OuAFLC4T28o6AqcFk4Pg068etV+bPB4K0uvKLEFALsvjocfNlyBMLGz7yh+fsp7BAgVVe1c3p
Tzfp1MEwUKS7SrAHNBqTlJ0z9c7qIjRsNv6t3KgIFPvTQcFgXeBnJDe/foEKgApVF+9GD/nopAoV0X
olga@olga-MacBookAir:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCFFeU+HqVpW3u/m4oHAKgZq9T2Z3BotC79wxd40zpSqWhrM5LQj0rLt6HE+h/cutZ94cbr2Fhtz90LJRnI3vAbrulx5o8AyZooyQWHSiciwslZ8FP
RQ2cAAGBzqbDAzrgZ8FGPPH4ZdoJmdjzE8s9T09f7YNGUSZi+Y8+/psyVShcsjXXXQE9orAbdGma+OuAFLC4T28o6AqcFk4Pg068etV+bPB4K0uvKLEFALsvjocfNlyBMLGz7yh+fsp7BAgVVe1c3p
Tzfp1MEwUKS7SrAHNBqTlJ0z9c7qIjRsNv6t3KgIFPvTQcFgXeBnJDe/foEKgApVF+9GD/nopAoV0X 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 вставляю.

И так, на GitHub-е заходим в настройки профиля, ищем ключи (ещё раз, смотрим галерею CodeReview) и добавляем наш публичный ключ.

Далее просто делаем `git push`.

Git скорее всего попросит у нас указать дефолтный ремоут (*upstream*), поэтому запускаем первый раз команду таким образом:

```
git push --set-upstream origin master
```

В следующие разы делаем просто `git push`.

Таким образом, если всё пройдет без эксцессов, на нашем форке Мальвининового 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 наиболинейший репозиторий проекта.

Далее, когда много людей ещё туда запускают с помощью своих pull request-ов, мы делаем pull request от Мальвины к самим себе, одобряя это сами же затем.

Причём по умолчанию GitHub думает, что мы хотим смерджить что-то к Мальвине от нас, поэтому 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 дальнейшем будем сразу коммитить в главный Мальвинин ремоут.

Но это потом)

Мурррр.