<https://online.sbis.ru/shared/disk/ddfd5d71-a391-4fce-a630-5df8f4fa21e8>

# Вопросы по Git

## Как можно найти на локальном компьютере склонированный репозиторий?

Куда склонировал репозиторий там и будет

## Ваша команда push отклонена git-ом. Почему такое произошло и ваши дальнейшие действия.

В моем случае из-за не соблюдения стилистики кода.

Также push сработает в том случае, если вы клонировали с сервера, на котором у вас есть права на запись, и если никто другой с тех пор не выполнял команду push. Если вы и кто-то ещё одновременно клонируете, затем он выполняет команду push, а после него выполнить команду push попытаетесь вы, то ваш push точно будет отклонён. Вам придётся сначала получить изменения и объединить их с вашими и только после этого вам будет позволено выполнить push.

## Для чего служат pre-push хуки на git.sbis.ru?

Хуки представляют собой скрипты, запускаемые при событии "commit". Это событие наступает при любом изменении/добавлении/удалении кода, будь это непосредственный коммит изменений в существующие файлы, либо создание новой ветки, либо выполнение мержа и т.д. Скрипты анализируют полученную от GIT-a информацию о коммите и следуя прописанным в них правилам разрешают либо запрещают выполнять данный коммит.

## Объясните, что означает origin в команде git push origin master? Расскажите об отличии удаленных веток от локальных. Какими командами можно обновить локальные ветки? В чем отличие этих команд?

**origin** – это удаленный репозиторий (откуда делали clone).

**git fetch origin -** извлекает все наработки, отправленные на удаленный сервер, но не сливает их с какими-либо вашими наработками и не модифицирует то, над чем вы работаете в данный момент. Вам необходимо вручную слить эти данные с вашими, когда вы будете готовы.

**git pull** - автоматически получит изменения из удалённой ветки master и сольет их со своей текущей веткой master

**git clone <repo>** - получить репо как он есть на github

## Какие состояния файла могут быть в git? С помощью каких команд можно изменить состояние файла?



**git add <file>** - добавить в индекс не отслеживаемый файл (сделать Staged), либо добавить в индекс измененный файл (сделать Staged).

**git reset HEAD <file> -** убрать файл из индекса

**git checkout -- <file>** - отменить изменения в файле до состояния последнего комита (если изменений еще нет в индексе)

**git commit –** зафиксировать изменения в локальном репо.

**git commit –amend –** заменит предыдущий комит

## Расскажите про параметры skip, abort и continue у команд git.

<https://habr.com/ru/post/161009/>

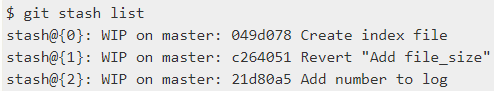
* git rebase –abort – отменить процесс rebase при конфлитке
* git rebase –continue – продолжить rebase, когда разрешили конфликты в файле и сделали git add
* git rebase –skip – пропустить конфликтный комит (не добавлять)

## Что делает команда $ git stash? Как посмотреть список всех сохраненных состояний в стеке? Как применить последнее изменение назад? Как удалить изменения, «спрятанные» командой $ git stash? Как создать ветку и сразу же применить «спрятанные» изменения?

* Делает вот что [https://git-scm.com/book/ru/v2/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-Git-%D0%9F%D1%80%D0%B8%D0%BF%D1%80%D1%8F%D1%82%D1%8B%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%B8-%D0%BE%D1%87%D0%B8%D1%81%D1%82%D0%BA%D0%B0#r\_git\_stashing](https://git-scm.com/book/ru/v2/Инструменты-Git-Припрятывание-и-очистка#r_git_stashing)

Операция stash берет изменённое состояние вашего рабочего каталога, то есть изменённые отслеживаемые файлы и проиндексированные изменения, и сохраняет их в хранилище незавершённых изменений, которые вы можете в любое время применить обратно.

* **Чтобы посмотреть список припрятанных изменений**, вы можете использовать git stash list:



* Вы **можете применить только что припрятанные изменения**, используя команду, указанную в выводе исходной команды: git stash apply. Если вы хотите применить одно из предыдущих припрятанных изменений, вы можете сделать это, используя его имя, вот так: git stash apply stash@{2}. Если вы не укажете имя, то Git попытается восстановить самое последнее припрятанное изменение.
* Команда apply только пытается восстановить припрятанные наработки — при этом они останутся в хранилище. **Для того, чтобы удалить их,** вы можете выполнить git stash drop, указав имя удаляемых изменений:



* Вы можете выполнить команду git stash branch, **которая создаст для вас новую ветку, перейдёт на коммит, на котором вы были, когда прятали свои наработки, применит на нём эти наработки** и затем, если они применились успешно, удалит эти припрятанные изменения:

## Какой командой можно изменить описание, указанное при создании коммита? Что следует учесть при использовании этой функции?

git commit —amend -m "New commit message."

## Схема

* git branch -f F2 C11 – ветку F2 переключить на комит C11
* git checkout F1 – переключиться на F1
* git merge F2 – влить F2 в F1 (комит F14)
* git branch -D F3 — удалить ветку F3
* git branch -f master C14 – переключить ветку master на комит C14
* git branch -D F1 – удалить ветку F1

## Схема

* git checkout F1
* git cherry-pick C4 C5
* git checkout master
* git merge F1
* git cherry-pick C3 C6 C7

## Схема

* git checkout C7
* git branch F3
* git checkout F3
* git commit
* git branch -f master F2
* git checkout master
* git cherry-pick C4 C5

или

* тоже самое
* git checkout F1
* git rebase master
* git revert C2'

## Схема

* git branch -f master F3
* git commit
* git cherry-pick C3 C7

или

* тоже самое
* git checkout F2
* git rebase master
* git branch -f master C7’

## Схема

* git branch -f master C10  
  git cherry-pick c3 c4 c5

или

* git branch -f master C10
* git checkout F1
* git rebase master

## Схема

* git branch -f master F1
* git checkout master
* git commit
* git checkout C10
* git branch F5
* git checkout F5
* git commit
* git checkout master
* git merge F3
* git cherry-pick C6
* git checkout F5
* git commit

## Схема

* git reset C2
* git commit
* git cherry-pick C4 C5

или

* git branch -f master C2
* git commit
* git checkout F1
* git rebase master
* git revert C3

## Схема

* git checkout master
* git merge C4
* git merge F2

## Схема

* git checkout master
* git merge C5

## Схема

* git checkout F1
* git cherry-pick C5
* git checkout F3
* git commit x2
* git checkout master
* git merge F2

## Схема

## Схема

Как на сайте:

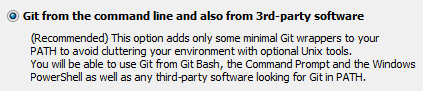
* git checkout master
* git cherry-pick c6 c7
* git commit
* git push

## Схема

# Вопросы по Tensor-flow

## Настройка Git в Windows

* С начала скачал git клиент
* Разрешил при установке использование Tortoise Git из командной строки Windows



* Установил визуальную оболочку для работы с Git - TortoiseGit.

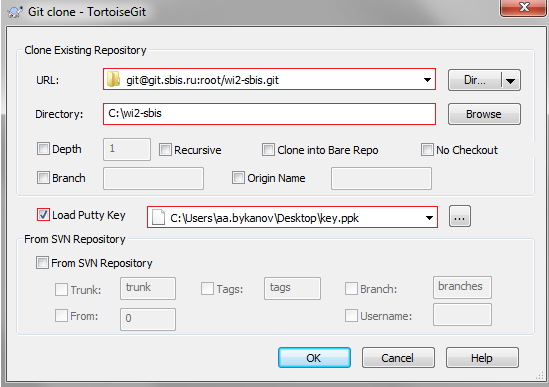
Далее необходимо создать SSH-ключ

* Запустить PuTTY Key и нажать Generate
* Нажать Save private key и сохранить на раб. стол
* В PuTTY Key экспортируем OpenSSH key в папку C:\Пользователи\ИмяПользователя\.ssh

Далее нужно добавить SSH-ключ на GitLab для того, чтобы можно было отправлять правки (push) в удалённые репозитории.

* Зайти на GitLab, ввести логин и пароль от компа
* Перейти в Settings, на закладку SSH Keys и в поле Key вставить наш OpenSSH ключ

Далее, при Clone с помощью TortoiseGit, должен быть указан приватный ключ:



## (3)Рассказать про особенности именования веток в tensor-flow.

* **master** — ветка, создаваемая в репозитории по умолчанию; не выкладывается на боевые сервера; может использоваться только в сервисных хранилищах или тестах.
* **rc-YY.ZZNN** — ветка, в которой копятся изменения и исправляются ошибки перед очередным выпуском версии продукта (обновлением ветки master)

YY - год.

ZZ - номер платформы (обычно 10, 11, 12, 20, 21, 22).

NN - номер релиза.

**development** — необязательная ветка

**Правила именования собственных веток:**

*main\_branch/class/class2/.../token*

**main\_branch** — это обозначение базовой ветки, от которой создаётся новая.

Пример: *20.2100/feature/new-class*

## 20.1100. -текущая версия RC, 20.2100. -следующая версия RC, которая готовится к релизу, development -основная ветка разработки. Описать пошагово действия для исправления ошибки в текущей версии RC см. (рис. 1)

* Переключиться на ветку 20.1100
* Сделать pull
* Создать новую ветку от 20.1100, переключиться на нее
* Исправить ошибку
* Сделать коммит
* Сделать пуш
* Создать Merge request

## Зачем нам необходимо мержить фичеветку, если какие-то изменения не требуются в следующих релизах?

чтобы не потерять изменения, нужно сливать в RC следующей версии и текущей. Чтобы все изменения уже изначально были в следующем релизе.

## Какие ваши действия, чтобы фикс из текущей RC не попала в development, если вы выполняете MergeRequest’ы согласно правилам?

## Мы имеем некоторые коммиты в development, которые нам необходимо прокинуть в Release Candidate, как нам это сделать?