**Git и GitHub**

[**https://github.com/aliaskov/bashscripts/blob/master/FeatureDevelopmentWithGit.pdf**](https://github.com/aliaskov/bashscripts/blob/master/FeatureDevelopmentWithGit.pdf)3 области – 1.Рабочий каталог 2. Очередь на коммит (добавленные в индекс) 3.Последний коммит  
**git команда - h** – вывод ключей и опций к команде **git команда -- help** –вывод подробного мануала по команде **;** - разделение команд в командной строке (успешное выполнение предыдущей)  
\*\* - разделение команд в командной строке (безусловное выполнение)

Эти 3 глобальные настройки делаем 1 раз на своем компе по умолчанию. Можно использовать **local** на отдельные проекты/папки с другими данными (user, name)  
**git config --global user.name 'Stanislav Tonkich' -  
git config --global user.email 'st.tonkikh@gmail.com'   
git config --global core.editor 'code --wait' (или 'vim') -**Другие настройки **git config --global help.autocorrect 1**  -… **git config --global help.autocorrect 2** -… **git config --global credential.helper** -… **git config --global credential.helper** -…

**git config --global -l** – проверить настройки git **git –version** – номер версии  
**code файл** – открыть файл в редакторе vscode

**Ключи  
ssh –lah ~/.ssh** - проверить наличие моего публичного ключа на GitHub  
**ls –lah ~/.ssh** - проверить наличие моего публичного ключа на локальном компе  
**cat ~/.ssh/id\_rsa.pub** – вывод ключа на экран  
  
  
**git init** – создает новый репозиторий  
**git remote –v** – проверяет,есть ли привязка локального репо с дистанционным  
**git status** – выводит состояние рабочего каталога –можно и нужно писать в любой непонятной ситуации  
**git log** – показывает коммиты. Выход из этого режима – q или Ctrl+C  
**git status -s** – сокращенная версия проверки статуса файлов в каталоге  
**git diff –** сравнивает и показывает разницу между раб.каталогом и индексом, т.е. что красным цветом  
**Рекомендуется делать перед коммитом.  
git diff –cached/staged** - сравнивает и показывает разницу между индексом и последним коммитом, т.е. что зеленным цветом)  
**git diff HEAD** (между раб. каталогом и посл. коммитом) **git add .** – добавляем в индекс все файлы из папки или **git add –A** все файлы или **git add файл**  
**git commit -m ‘комментарий’** – создаем коммит из командной строки без открытия редактора  
**git commit -am ‘комментарий’** – добавляет в commit только **отслеживаемые файлы** (добавленные в индекс/очередь на коммит), новые файлы не добавляются – для них нужно делать **git add.**   
**git commit –amend** **-m ‘внес правки’**- перезаписать посл.коммит, **если только его не отправили в удаленный репозиторий** (деструктивная команда, не рекомендуется так делать). Ситуации: неправильный комментарий или забыл что то добавить файлы в индекс

**git reset (файл)** – отчищаем индекс от всех или одного файла, т.е. удаляем на коммит  
**git reset –hard HEAD~1 -** отчищаем индекс от последнего коммита **- !!!**  
**git reset –hard ID\_коммита -** отчищаем индекс до коммита с указанным ID **- !!!**  
 **git rm файл** – удаляем файл **физически!!!** из каталога и репозитория и автоматически делаем индексацию. После этого нужно закоммитить  
**git rm --rf .git** – удаляем на локальном компе **весь репозиторий  
git rm --cached файл** – снимает файл с учета из индекса, но локально и в репозитории он остается

3 типа файлов  
(1) неотслеживаемые ?? или UU – неотслеживаемый/untrecked и не проиндексированный  
(2) отслеживаемые   
MM **–** M –в файле, находящ. в индексе произошли изменения (modif), M-файл необ. добавить в индекс  
А–файле находится в индексе и готов к коммиту  
(3) игнорируемые   
 **git log** – выводит журнал коммитов (**q** – выход из режима)  
git log –graph – выводит в графическом режиме  
**git log --oneline** – краткая информация по коммитам (id коммита укороченный, комментарий)  
**vim .gitignore** – создаем файл для игнорированных файлов/папок  
 \*.log  
 \*\*/logs  
 # комментарий   
**/ключевое\_слово** – поиск в редакторе ключевого слова  
**less файл** – просмотр содержимого файла

**git checkout** **id** – переключение на коммит по его id и делаем его текущим/активным  
Его мы делаем текущим и все файлы возвращаются в состояние этого коммита  
**git checkout** **-f** – откат на последний коммит (деструктивная команда) – удаляем все правки и возвращаем состояние рабочего каталога на последний коммит.  
**git show** – просмотров изменений в файлах – текущих и которые закоммичены **git show** id – просмотров коммитов (внутри) по его id  
**git show id:файл** – просмотров изменений в конкретном файле в конкретном коммите  
  
**Проверить есть ли привязка к Гитхабуgit remote show** или **git remote -v** - посмотреть список всех привязок   
**git remote show origin** - проверить конкретную привязку (текущего репо с удаленным)   
**git remote rm origin** - удалить привязку

**Ветки**

Концепция ветвлений

|  |  |
| --- | --- |
| **GIT FLOW** -. Master/ main / stable – не прикосновенна. Сюда попадает только отлаженный и проверенный код (перед самым продакшеном) - Develop / current –сюда сливаются все фиксы кода и новые части кода и т.д. Т.е. рабочий и текущий он - Ветки с коротким циклом жизни (short-term-branches) hotfix1/ login / payments/ issue001 / ticket002 | **GITHUB FLOW** - Master / main  - Short-term-branches |

**Начинаем работу с удаленным репозитрием, если его у нас не было локально  
git clone git@github.com:ИмяАккаунта/ИмяРепозитория.git /папка-** скачать себе локально (в текущую или указанную папку) чей то репозитарий с гитхаба. Либо можно его Code-Download.Zip  
 **Скачиваем обновления с удаленного репозитрия  
git pull –** обновляем наш локальный репозитарий обновлениями с удаленного. Это тоже самое, что слияние (merge) удаленного репо с нашим локальнымю  
 **Начинаем работу с локальным репозитрием**Создаем на Github папку проекта  
**git remote add origin ssh\_(ссылка скопированнная с GitHub)** – привязка локального с удаленным  
**git remote rm origin** – удаление привязки локального репозитария к удаленному  
**git remote show origin** – показывает привязку локального репозитария с удаленным

**git branch** – посмотреть список веток;  
**git branch name\_branch** – создание новой ветки;  
**git branch –v** вывести подробности о ветке   
**git checkout –b name\_branch/**– создание новой ветки и сразу переключение в нее  
 **Переименование удаленной (на Гитхабе) ветки  
1. git branch –m name\_branch –** переименование текущей локальной ветки   
**2. git push origin –delete** **старое\_название\_ветки** – удаление удаленной старой ветки  
**3. git push -u origin** **новое\_название** – пушим ново-переименованную ветку

**git log –** вывод информации о коммитах в текущей ветки  
**git log --oneline –decorate** осмотреться, в какой ветке мы находимся  
**git checkout** (или **switch**) **name\_branch** - переключение между ветками (с текущей ветки HEAD на ветку с именем name\_branch)  
  
**git branch -d name\_branch** – **безопасное** **удаление ветки** локально, т.е. **с вопросом на подтверждение**  
Если выходит сообщение “is not fully merge” – это означает, что эту ветку никуда не сливали. **git branch -D name\_branch** – удаление ветки **принудительно** (вез вопросов/подтверждений) локально   
**git push origin –delete** (или **-d**) **name\_branch** – удаление ветки на **GitHub**  
**Слияние веток  
git branch –v** – смотрим информацию о ветках (имена, id, коммитв\_ состояние наших веток **git diff name1 name2** – смотрим, что будет добавлено в ветку 1 из ветки 2 при слиянии   
или эту же команду   
**git diff** -  **git diff ..name –**смотрим, что привнесется в текущую папку, в которой находимся при слиянии из папки name (тоже самое выполняет команда **git show**)  
**git merge namebranch** – слияние ветки namebranch с текущей, в которой находимся  
**git branch –abort** – прерывание слияния  
**git show bugfix:файл** – посмотреть файл в ветке bugfix не переключаясь в ту ветку  
**git show id\_коммита:файл** – посмотреть файл в коммите с указанным id  
**git show id\_коммита –** посмотреть изменения в коммите с указанным id  
  
**git revert commit\_id** - удалить правки в коммите через создание нового коммита

Если находились в ветке bugfix и нужно, что делали в ней слить с master  
**git checkout master** –переходим в ветку master **git merge bugfix** –слияние ветки bugfix с текущей master **Алгоритм - создание ветки, внесение правок, закомитить, слить с основной веткой, удалить временную ветку, переключится в другую ветку, не закомитив текущую**(1) **git checkout -b новая\_ветка** (создать и переключиться на новую ветвь)(2) внести необх. правки в новой ветке (через любимый редактор)(3) через **git diff** (посмотреть правки)\*(4) выполнить фиксацию правок (**git commit –am ‘комментарий ’**) Смотри ниже комментарий  
(5) переключиться на ветку **master (git checkout master**)(6) привнести изменения из другой ветки в master (**git merge другая\_ветка**)(7) удалить ненужную ветвь (**git branch -d другая\_ветка**)

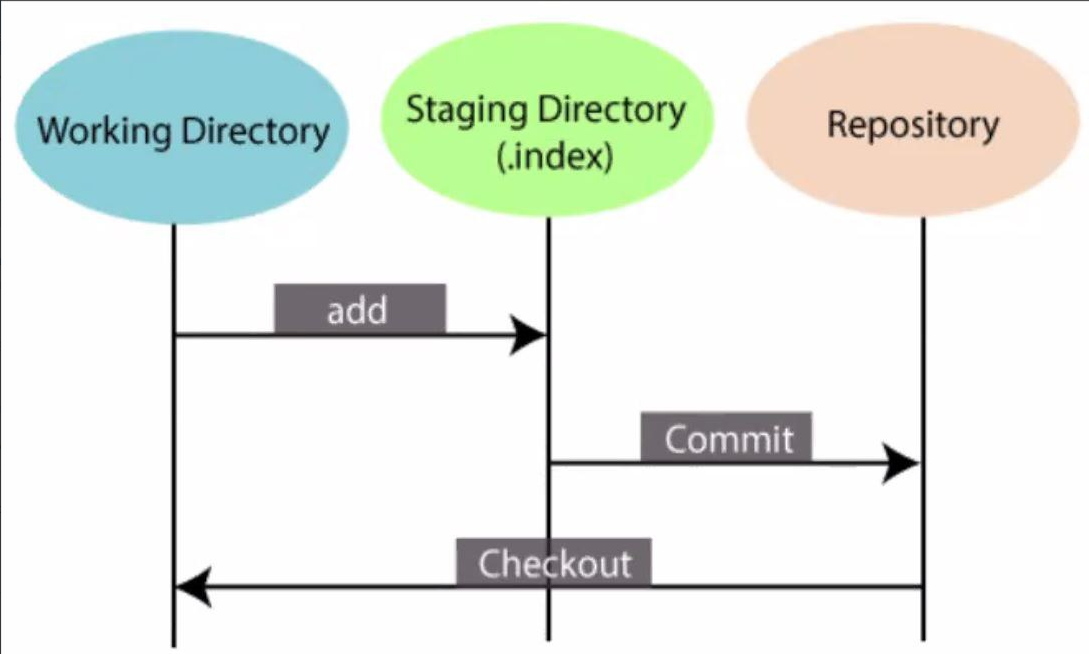
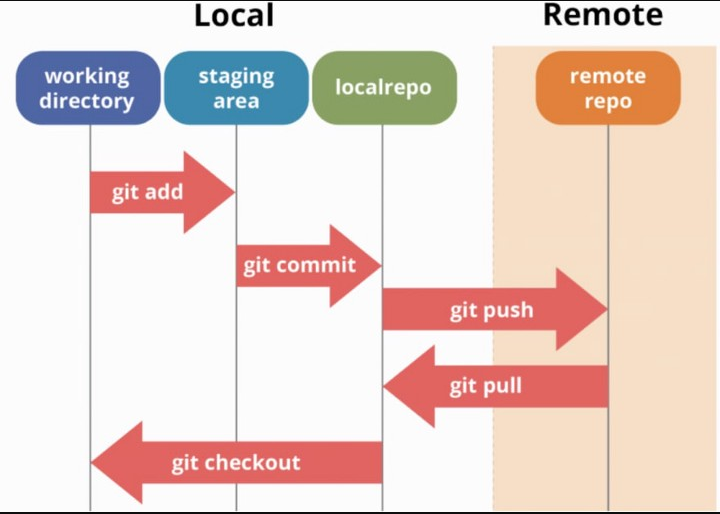
На шаге (4) может быть ситуация, когда в рабочей ветки сделали изменения, но нет смысла делать коммит (по требованиям к коммитам) Без этого мы не сможем переключится в другую ветку, не сделав либо commit либо stash  
  
**Заначка (stash) – переключение между ветками бех сохраненного коммита**  
**git stash** - сохранить в “заначку” незакоммиченные правки.   
**git stash save ‘комментарий’** - сохранить незакоммиченные правки с комментарием.  
**git stash list** - посмотреть списком все "заначки"  
**git stash pop (stash\_id)** - вернуть/применить ранее сохраненные правки с его удалением из “заначек”  
**git stash apply** - вернуть/применить ранее сохраненные правки БЕЗ его удаления из “заначек”  
**git stash drop stash\_id** - удалить "заначку" по ID  
**git stash show <stash\_id> -p** - посмотреть правки в "заначке"  
**git stash clear** - удалить все "заначки"  
  
**Конфликты в ветках**  
Сценарии  
1) **git merge abort –**прерываем слияние  
2) Если при слиянии произошел конфликт, то заходим в любимый редактор и вручную удаляем лишнее, оставляем нужное или оставить оба варианта (Accept Current Change/ Accept Incoming Change/Accept Both Change/Compare…) и т.д.   
После этого делаем **git commit –am ‘комментарий’** , т.е. завершаем слияние после ручных

**Выгрузка на GitHub**

1. Создать проект локально и проинициализировать его **git init**  
2. Добавить файлы/папки в индекс **git add .**   
3. Закоммитить **git commit –m ‘comment’**  
4. Создать репозиторий на GitHub вручную  
5. Связываем локальный репозиторий с созданным на Гитхабе  
**git remote add origin** [**git@github.com:ИмяАккаунта/ИмяРепозитория.git**](mailto:git@github.com:ИмяАккаунта/ИмяРепозитория.git) **git remote add origin** git@github.com:St-ton/Git.git  
6. Пушим наш локальный репозитарий на гитхаб   
или **git push –u origin name\_branch**  
 **Копирование ключей на GitHub**  
Можем подключить по ключу  
**1) ls ~/.ssh -** переход в локальную папку с ключами **2) cat ~/.ssh/id\_rsa.pub –**открываем файл с ключами3) **GitHub – SSH and GPS keys – SSH keys**   
Копируем публичный ключ c экрана и копируем ключ в настройки **4) ssh –T** [**git@github.com**](mailto:git@github.com) – проверяем, авторизированы ли мы

**Требование к коммитам**- завершенность  
- атомарность  
 **Правила написания commit  
операция (модуль): текст коммита**  
fix(profile): исправил функцию определя балансаrefactor(profile): почистил старый кодtest(profile): написал тест функции ABCbuildfeat(profile): добавил поддержку оплаты через банк

**git diff …** не написали что значит

**Задача на уроке   
1.git clone git@github.com:Xspeed/test-repo.git** – клонируем себе репозитарий в текущую локальную папку  
**2. cd test-repo  
3. git log** находим коммит с ‘initial commit’, берем его id **4. git show 2422:index.html**