**ПКоманды в GitBush**

**Не использовать:**

**Git rebase –** изменяет часть предыдущих коммитов (меняет их хэш номера на новые), выстраивая их в красивую блядь ровную линию без ответвлений**. И если от них (еще не измененных) кто-то из команды повел свою новую ветку, то он хрен нормально сольется назад в общую ветку!** Не пользоваться!!!! Минусы в десятки раз превышают плюсы!!!!

**Создать:**

**Git add –I** потом нажать **u**потом выбрать нужный/е номер/а файлов, например **1, 3** потом нажать Enter, потом ничего не вводя еще раз  нажать Enter, потом нажать 7 и Enter (*для выхода из этого режима назад в командную строку git-а*), потом ввести команду **git commit –m “- - ”** (*вводя коммит именно для этого\этих изменений*), потом повторить всю процедуру сначала для других изменений – если изменено много файлов и нужно их добавлять в коммиты по частям, а не одним общим огромным коммитом

**Git add –I**  потом нажать **p**потом нажимать **y** или **n**для тех кусков файла, которые будет показывать  потом **7** – если много изменений ***в разных частях*** одного файла и нужно их добавлять в коммиты по частям, а не одним общим большим коммитом

**Git branch [ветка]** - создаст у меня на компе новую ветку (***от той ветки в которой я сейчас нахожусь!***), но не перейдет в нее.

**Git checkout - b [ветка]** - создает у меня на компе новую ветку (***от той ветки в которой я сейчас нахожусь!***) и сразу в нее перейдет для работы

**Git config - -global alias.br “branch”** +дописать в файле **.config** (лежит в папке **.git**) **[alias] br = пbranch –** создаст псевдоним (сокращение) для команды **branch**, чтобы писать быстрее (**git br**). Если оболочка поддерживает, то можно писать псевдонимы команд, дописав в файл **.profile** например **alias gs=”git status ”**

**Git push origin -u [ветка]** – создает /отправляет на удаленный сервер новую созданную на компе ветку и сразу создает связь между веткой на компе и на удаленном сервере, чтобы потом автоматически делать push/pull. Если это не сделать, то потом для связи веток нужно будет написать **Git push - -set-upstream** **origin [ветка]**.

**Git clone git@github.com:githubmaxim/Java-Exersises.git e:/github/j –** создает/клонирует на комп на диск e:/github/j необходимый репозиторий с удаленного сервера git@github.com:githubmaxim/Java-Exersises.git (при таком копировании будет использоваться SSH протокол обмена данными)

**Touch [название файла с расширением** или **без]** – создаст на компе в текущей ветке новый файл (например для создания файла **config** в папке **.ssh** где для возможности создания второго пользователя на своем компе нужно будет прописать параметры этого второго пользователя)

**Удалить:**

**rm [название файла с расширением] –** удалит файл у меня на компе

**Git branch –d [ветка]** - удалит ветку у меня на компе если она уже слита с основной веткой. Если нет, то потребует либо слить либо поставить ключ “-D” означающий «все равно удалить»

**Git checkout - -** **[файл с расширением]**– удалит из файла все изменения до состояния последнего коммита

**Git push origin :[ветка]** – удаляет ветку на удаленном сервере

**Git rm [название файла с расширением]** потом **Git commit –am “”** потом **Git push** - удалит файл у меня на компе и на удаленном сервере. Github не может удалить файл из репозитория, не создавая коммит.

**Git rm log/\\*.log** - эта команда удаляет все файлы находящиеся в директории log/ и имеющие расширение .log .

**Git rm - -cached [название файла с расширением]** потом **Git commit –am “”** потом **Git push** - удалит файл только на удаленном сервере. Github не может удалить файл из репозитория, не создавая коммит.

**Git reset - -hard a1b2c3 потом Git push - -force –** удаляет все коммиты и изменения в коде программы до указанного имы откатываемся к коммиту (и коду программы) a1b2c3 + сообщает об этом удаленному серверу и он тоже удаляет

**Git commit - -amend** – удаляет последний коммит и заменяет его новым коммитом. Если были еще какие-то изменения в файле, то вначале нужна еще будет команда **git add .**, чтобы проиндексировать дополнительные изменения.

**Git reset HEAD [файл с расширением]**– удалили из индексирования последние изменения в файле, при этом файл остается измененным.

**Копировать:**

**Git checkout - -track origin/[ветка]**– копирует мне на комп новую ветку с удаленного сервера и начинает отслеживать

**Git pull/ (git fetch+git merge) –** копирует /получает мне на комп с удаленного сервера обновленные файлы + проводит слияние с моими файлами

**Git push/ Git push [удаленный сервер][ветка]**  (возможный вариант **git push origin master**) **–** копирует/отправляет мои файлы на слияние на удаленный сервер

**Просмотр информации:**

**Gitk –** запуск информационной программы со всей информацией по репозиторию в котором мы работаем

**Git branch** – покажет какие ветки есть у меня на компьютере для этого репозитория

**Git branch - -merged** – покажет какие ветки у меня уже слиты с веткой в которой я сейчас нахожусь

**Git branch - -no-merged** – покажет какие ветки у меня не слиты с веткой в которой я сейчас нахожусь

**Git diff -** покажет не проиндексированные изменения в файлах (до git add)

**Git diff - -cached –** покажет проиндексированные изменения в файлах (после **git add** до **git commit**)

**Git status -** покажет какие файлы в каком состоянии находятся

**Git log -p -1** или **-2** или **-3** и т.д. – расширенная информация (какие изменения/кто/когда), 1-го/2-х/3-х и т.д. последних коммитов

**Git log - -since=2.weeks -** покажет список коммитов, сделанных за последние две недели

**Git log - -no-merges master..sss** – покажет коммиты ветки sss, которые еще не слиты с веткой master. И наоборот если написать **sss..master**

**Git remote –v** – покажет к каким удаленным репозиториям у меня есть доступ

**Остальное:**

**Git add .** или **[название файла с расширением]** – для только, что созданного файлов /файла добавит их/его в отслеживаемые

**Git checkout [репозиторий]/[ветка]** - переключиться на другую ветку. Если работать с локальным репозиторием, то указывать его имя не обязательно

**Git commit –m “ ”** – закоммитить после **commit add .** (если файл новый)

**Git commit –am “ ”** – закоммитить без **add** (если файл не новый), т.е. закоммитить сразу без промежуточной индексации

**Git diff –check** – показывает ненужные пробелы в коммите перед отправкой

**Git init –** поместит папку, в которой сейчас мы находимся в GitBush, под контроль Git-а.

**Git merge [ветка] –** сольет у меня на компе эту ветку с той в которой я сейчас нахожусь (нужно переключиться на ту ветку которая будет главной и с нее выполнять команду)

**Git mv file\_from file\_to** – переименование файла

**cd e:/github/java-exersises -** заходит по написанному адресу (например как тут **e:/github/java-exersises)** в нужный репозиторий у меня на компе для работы

**Ctrl + x** **и подождать** *(должно выйти само)*– нажать, чтобы выйти из открывшегося окна при слиянии веток/просмотра коммитов/д.р.

**Выкладывание своего проекта на GitHub.com** написано в отдельном файле “Как выложить свой проект на GitHub.com”.

**Игнорирование файлов**

**Touch .gitignore** - создает файл **.ignore** в котором будут перечисляться файлы или папки, файлы которые будут игнорироваться (не подтягиваться) **GitHub-**ом. Потом заходим в этот файл и записываем в него названия файлов с расширениями/***\*.***расширения/ папки (типо ***ignore/***) – то, что должно будет игнорироваться.

Файл **.gitignore** это общий файл для всех и будет записываться после **git add . + git commit + git push** на удаленный сервер для всех.

**Для создания личного** (без сохранения на удаленном сервере) файла игнорирования нужно те же изменения внести в файл **exclude**, который лежит впапке **.git/info**

Для игнорирования чего-то во всех проектах (а не только в этом), нужно создать файл **.gitexcludes**, заполнить его, после чего в файл **.gitconfig** вручную добавить:

**[core]**

**excludesfile = ~/.gitexcludes**

**Спрятать недоделанные файлы без коммитов**

(действует и когда уже сделан add, и когда нет)

**Git stash –** удалит и спрячет все внесенные после последнего коммита изменения в файлы + даст номер этой записи 0 или 1 или 2 и т.д. При создании новой спрятанной записи (“заначки”) даст ей номер 0, а остальные сдвинет на +1

**Git stash list –** покажет все “заначки”

**Git stash apply stash@{**номер “заначки” в списке **list:** или **0** или **1** или **2** и **т.д.} –** применение к файлам изменений из заначки. Это необходимо делать, находясь в ветке, на которой сделана была “заначка”.

**Git stash drop stash@{**номер заначки в списке **list} –** удалит эту “заначку”

**Восстановить удаленные “reset -- hard” коммиты**

Все что попало в коммит почти всегда можно восстановить.

**Git log –g** – ищем нужный коммит

**Git branch [название ветки] *ab1afef*** – создаем новую ветку, указывающую на найденный коммит

Если коммит не был найден, но очень нужен:

**Rm –Rf .git/logs/ -** удаляем весьreflog

**Git fsck - - full –** покажет вообще все объекты базы. Ищем наш коммит там где в начале строчки будет написано ***dangling commit*** (“висячий" коммит)

**Git branch recover-branch *ab1afef*** – создаем новую ветку, указывающую на найденный коммит