**ЛОКАЛЬНАЯ РАБОТА НА ЛИЧНОМ КОМПЕ (ЛОКАЛЬНЫЙ РЕПОЗИТОРИЙ)С ПОМОЩЬЮ КОНСОЛИ**

(здесь описано как создать локальный репозиторий, добавлять изменения, коммитить их, создавать и удалять новые ветки, подгружать удалённый репозиторий, мёржить ветки)

Dla raboti s cp snachala nado sozdat papky gde bydet xranitsia proekt, который будет отслеживать гит

Zatem pereidia v ety direktoriy nado nabrat komandy:

----------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git init** - С ее помощью можно инициализировать новый пустой репозиторий.

Большинство остальных команд Git невозможно использовать без инициализации

репозитория, поэтому данная команда обычно выполняется первой в рамках

нового проекта.

repozitorii-xranilische, gde lezzit proekt.

----------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git --version** возвращает версию

—--------------------------------------------------------------------------------------------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

git add- dobovliaet faili ili izmeneniia v statys otslezzivaemix(staging

area(obretayt zelenii cvet)), posle etogo dannie izmeninia mozzno soxranit ispolzyia

komandy git commit. Est raznie varianti dannoi komandi:

**git add** nazvanie faila; (Название файла без кавычек и с расширением)

**git add.** -dobavit vse faili v tekyschei papke (pered tochkoi probel)

**git.add\*.java** -dobavit vse faili v tekyschei papke s rasshireniem .Java

----------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git commit -m “Nazvanie kommita ili kommentarii k kommity”** (создание коммита, кавычки двойные)

**git commit -m "Add first time"** ( ← это пример) -m <--oboznachaet 'message' dalee idet soobschenie (delaetsia na English) tak skazat kommentarii k commity. Commit eto onalog soxraneniia.

**git commit** - создаёт коммит с развёрнутыми коментариями к нему, т.е. открывает подредактор где можно прописывать что-то более углубленно. Выйти из этого редактора можно нажав Esc :wq

Takzze est variant ispolzovaniia dannoi komandi s parametram 'a',

naprimer: v tekyschii moment изменений v otslezzivaemix failax repozitoriia net. Vnosim izmeneniia, i mozzno vnesti ix v staging area s pomosch`y komandi 'git add', no mozzno vospolzovatsia

**git commit -a -m “nazvanie komita”** Pri etom mi kakbi propyskaiaem etaр s komandoi 'git add' i srazy sozdaem komit. Vazzno pomnit chto dannii variant rabotaet tolko s izmenennimi failami и уже имеющимися, но не с novimi/tolko chto sozdannimi

git commit -amend. Itak mi zdelali commit i vdryg ponimaem chto v dannom izmenenii

nado zdelst opiat nebolshoe izmenenie, opiat vnesli изменения в документе,но новый komit sozdavat ne variant,

i vot zdes delaem:

**git add “nazvanie\_izmenennogo\_faila”** (добовляем в staging area), a zatem

**git commit --amend** (тем самым переписываем/редактируем последний коммит не создавая новый)

(после --amend mozzno dobavit -m 'nazvanie komita')

Esli ykazat nazvanie kommita (-m '...') to predidyschee nazvanie bydet perepisano.

esli ne ykazivat to posle vipolneniia, git poprosit v svoem redaktore ykazat nazv-

anie, mozzno nabrat ' Esc :wq ' <--vixod s redaktora soxraniaia izmeneniia.

----------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git status** <-- otobrazzaet tekyschee sostoianie gita-a!

Napishem git status i vidim 'nothing to commit, working tree clean' t.e. сохранить

poka nechego tak-kak faili otslezivaemie git-om ne vidoizmeneni. No nado v tekyschem

dokymente vnesti izmeneniia i soxranit eti izmeneniia cherez (ctrl + s) ili zakrit

ykazav soxraneniia. Posle etogo Git 'vidit' izmeneniia (oni krassnie) i pri vipolnenii komandi

git status 'govorit' chto fail bil 'modified', mozzno vipolnit git add 'nazvanie

faila',a zatem 'zakomitit' chtob vnesti izmeneniia i zdelat 'snimok' proekta.

Sozdadim i soxranim v etoi zze (direktorii/papke) esche odin dokyment, pri git

status Git 'skazzet' chto est Untracked files: 'nazvanie novogo dokymenta'.

Opiat zze dobovliaem 'novii dokyment' k staging area s pomosch`y git add 'nazvanie

faila', i 'kommitim...

Esli izmenenn odin fail i dobavlen drygoi, to polychaem bolee chem odno izmenenie

v nashei rabochei direktorii i vipolniaia komandy add mozzno ykazat nazvanie oba

faila: git add "nazvanie\_1" "nazvanie\_2" takim obrazom ne vazzno skolko bilo

izmenenno failov ix vse nado 'perenesti' v staging area, a zatem git commit -m

"kommentarii" i polychim novii 'slepok' nashego proekta.

----------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git log** <-komanda pozvoliaet posmotret vse 'kommiti' kotorie bili zdelani za vse

vremia. Tam est xesh <--ynikalnii nomer kazzdogo kommita, imia avtora...

----------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git diff**- komanda pozvoliayschaia otslezzivat izmeneniia mezzdy tekyschim sostoia-

niem faila i sostoianiem na moment poslednego commita (soxraneniia).

Primer:

1) Vipolnim git status i polychaem nothing to commit, t.e. izmenenii net

2)vnosim izmeneniia v fail... i proveriaem na git status <-- vidim krasnii tekst govoriaschii

chto bili proizvedeni izmeneniia.I vot teper nabrav komandy git diff nam krassnim tekstom

bydet pokazano poslednne soxraneniie faila (pri poslednem kommite) a zelenim to chto bilo

izmeneno/vneseno.

Esli izmeneniia neystraivayt mozzno zdelat **git reset --hard**, dannaia komanda 'otkatit'

nazad gde net nikakix izmenenii (изменения не будут сохранены), tak-kak --hard otkativaet nazad bez soxranenia izm-

enenii. Т.е. если есть первый и второй коммит (щас находимс на втором) и с помощью –hard откотить на первый то второй коммит удалится, но это только про –hard

--------------------------------------------------------------------------------------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HEAD <-- eto YKAZATEL tekyschego polozzeniia proekta

**git reset**- pozvoliaet pereiti k predidyschemy commity. Est raznie parametri etoi komandi:

**git reset --soft xesh\_kommita\_kyda\_pereiti** <--perexodit k dannomy kommity, при этом сохраняя все изменения которые были зделаны позднее данного комита, что даёт возможность переписать историю коммитов, сохранив новый функционал. (pri etom soxroniaet vse sozdannie i izmenennie faili, no kakbi oni yze ne zakomicheni

(ne soxraneni), NO otslezzivaemie (zelenie), i mozzno zdelat kommit srazy bez komandi add. Pri etom kommiti s komentariami k nim steraytsia,no kak ia yzze skazal, izmeneniia vse ostaytsia.

**git reset --mixed 'xesh kommita kyda pereiti'** <--perexodit k dannomy kommity

pri etom vse dannie i izmeneniia v failax zdelannie posle dannogo kommita kyda mi pereshli

popodayt v neotslezzivaemyy zony(unstaged - krasnie) pri zzelanii ix mozzno dobavit (add .) v

otslezivaemie i dalshe zakommitit kak xochesh.

git reset --hard 'xesh kommita kyda pereiti' <--perexodit k dannomy kommity

pri etom vse dannie i izmeneniia v failax zdelannie posle dannogo kommita kyda mi pereshli,

bydyt ydaleni.Primer: sozdali fail,-zdelali commit.# 1

izmenili fail,-zdelali commit.# 2

Ponimaem chto vse izmeninia ne korrektni, delaem git reset --hard commit #1 i perexodim v

'sostoianie' kogda y nas sozdanni fail bez vnesennix posle sozdaniia faila izmenennii, t.e.

kakbydto mi otmotali nazad i stoim seichas na tom meste kogda sozdali fail.Dannie sterti.

Esli commit, kak-bi pereskakivaet cherez neskolko to 'pereskochennie' komiti ydaliaytsia.

Takzze mozzno prosto napisat git reset <--eto vizov dannoi opcii kak-bi poymochaniy,

v etom variante proisxodit 'otkat' na samii poslednii kommit s parametrami 'mixed', t.e.

vse izmeneniia perexodiat v statys ne otslezzivaemix (krasnix)

Takzze mozzno vmesto xesh-a kommita ispolzovat simvoli: '^' ili ~2 (otkat na 2 kommita)

Primer: git reset HEAD^ <--'otkat k poslednemy kommity

git reset HEAD^^<--'otkat k pred-poslednemy kommity

OPANKI-tyt okazivaetsia nelzia polzovatsia git reset --hard ydaliaia k primery neskolko pos-

lednix kommitov v pyblichnom repozitorii. Tak-kak v pyblichnom rep. nad proektom mozzet

rabotat neskolko razrabotchikov i ia k primery ydaliay poslednii komit, a moi tovarisch

yzze rabotaet nad moim poslednim komitom proekta. A esli tovarischei neskolko.Polychaem

chto neskolko versii, y kazzdogo razrabo seichas svoia versia, chto est ooochen ploxo.

Kakyy seichas versiy schitat za tekyschiy?

-----------------------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Kommitit mozzno ne vse faili srazy a k primery po odnomy izmenennomy.K primery est 2-a faila

izmenennix i oba oni v neotslezzivaemoi/unstaged zone, zdelaem git add "fail 1",git commit -m

"soobschenie"= zakomitili tolko odin fail.vtoroi tak i ostalsia v unstaged zone. Tozze-samoe

delaem i s vtorim,tem samim kak bi deferenciryia kommiti, chto mozzet bit polezno v bydyschem.

-----------------------------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git checkout-** imeet 3 scenariia primenenia:

1)slyzzit dlia peremescheniia mezzdy kommitami

**git checkout HEAD^^** <--otkativaet na dva komita v 'proshloe', s pomosch`y dannoi komandi nelzia

delat kakie-libo komiti, naxyia eta komanda nyzzna? Prosto naverno chtobi

posmotret kak vigliadel proekt na tom ili inom etape... Takzze dannyy komandy mozzno ispolzovat

s xeshem kommita, tipa **git checkout e4aa4ea1917bb14c479ba76017dedc96f6366806**. Rezyltat takoizze.

**git checkout master** -- vozvraschaet v samyy aktyalnyy versiy proekta.

2)Peremeschenie mezzdy versiiami failov.

Naprimer mi naxodimsia na poslednei versii programmi i zdelali kakieto izmeneniia v faile. Dannie

faili popodayt v razdel ne otslezzivaemix (untrackted/krassnix). No dannie izmeneniia nas ne ystraivayt

i mi xotim ix ydalit. Pishem

**git checkout nazvanie faila** <-- otkatit do predidyschei versii faila, gde net vnesennix izmenenii.No eto

tolko pri yslovii chto izmeneniia ne otslezzivaemie(krasnie).

**git checkout -- .** <-- otkatit vse izmeneniia (v slychae esli izmeneniia bili zdelani vo mnogix failax)

v to mesto gde bilo zdelano poslednee soxraneniia (kommit).No eto tolko pri yslovii

chto izmeneniia ne otslezzivaemie(krasnie). Ne zabyd probel pered tochkoi.)

Chtobi ydalit yzze otslizzivaemie izmenenia(zelenie) mozzno ispolzovat git reset --hard. V etom

slychae proekt bydet otkachen nazad do poslednego kommita i vse izmeneniia bydyt ydaleni. NO esche odin

sposob zdelat eto tak: git reset <--(po ymolchaniy primeniaetsia parametr 'mixed') perevedet otslezzivae-

mie faili (zelenie) v neotslezzivaemie (krasnie), i dalshe git checkout nazvanieFaila <-- ydaliaet vse

izmeneniia.

3)Esli mi xotim chtobi tekyschii fail sootvetstvoval etomy-zze faily no v proshlom.

Tak skazat vziat staryy versiy faila za tekyschyy. Pishem:

**git checkout xesh\_kommita\_gde\_lezzit\_nyzznaia\_versiia\_faila -- nazvanie\_faila\_kotorii\_xotim\_v\_tekyschei\_versii**

pri etom v repozitorii vse ostanetsia tem-zze, no versia faila ykazannaia pomeniaetsia na ty kakoi bila v yk-

azannom kommite.

-------------------------------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git clean**

Primer: Mi vnesli v repozitorii novii fail. No ponimaem chto on nenyzzen, s pomosch`y git reset --hard

y nas ne polychitsia etot fail ydalit tak-kak dannaia komanda rabotaet s izmeneniiami v failax no ne s

samimi untracted failami (naverno). V obschem xochy ydalit danni fail, kak? git clean rabotaet s untrakted

failami. Nabiraem git clean -n <-- polychaem otvet "Would remove TTT.txt" t.e. ydaly takoi-to fail.

naberaem git clean -f <--tem samim podtverzdaia chto mi xotim ydalit vse untracked faili iz repozitoriia.

Proveriaem faila net))).

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**RABOTA S YDALENNIM (REMOTE) REPOZITORIEM iz konsoli**++++++++++++++++++

С помощью GIT можно свою работу сохранить в интернете, это называется удаленным репозиторием, данная практика позволяет дополнительно сохранить версию проекта, а также позволяет другим делать коммиты в наш проект, тем самым даёт возможность работать многим над одним проектом. Для этого существуют спец. ресурсы GIThub BitBucket, GitLab

**Чтобы начать** **работать с удалённым репозиторием** надо сначало его создать на GitHub-е, для этого нажми плюсик с верху с права (в своём акаунте на github), выбрать New repository, задать имя и нажать создать. Далее скачай ссылку (HTTPS) - это и есть ссылка, чтобы её указать в локальном репозитории, это и называется добавлением удалённого репозитория

**git remote add "nazvanie\_удалённого\_Repozitoriia" ssilka na gitHub** <--Добавление локального репозитория на удалённый (в github нажми CODE > HTTPS и скопируй ссылку которую и надо вставить), после этого локальный репозиторий знает об удалённом и можно добавлять локальные ветки на удаленный. "nazvanie удалённого Repozitoriia" <--Это может быть любым именем, оно даётся, чтобы в будущем обращаться к данному репозиторию не по ссылке а по заданному имени

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**git remote -v** <-- pokazivaet vse chto mozzno delat s repozitoriem

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**git push название\_удалённого\_репо nazvanie\_vetki\_kotoryy\_Pyshim(т.е. отпровляем)** <--Команда для отправки локального репозитория на удалённый, выполняется на стороне локального репозитория. После того как мы запушили нашу ветку и например работали дальше и зделали ещо пару коммитов в нашем локальном репо., то нам необходимо сново пушить ветку на удалённый, если хотим чтоб на удалённом репо лежало всё актуальное. **nazvanie\_vetki\_kotoryy\_Pyshim-**имеется в виду, чтоможно пушить и другие ветки созданные локально у себя, на удалённый репозиторий.

**For example:** **git push LerningMaterials master**

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**переписываем (удаляем коммиты) историю коммитов на удалённом репозитории**

**git push -f название\_репо\_куда\_пушим название\_ветки\_которую\_пушим.** Например если мы удалим все коммиты у себя на локальном репозитории с помощью git reset –soft и хотим данный репозиторий запушить в удалённый, то выскочит ошибка (error: failed to push some refs to…), так-как история коммитов в удалённом репозитории будет намного больше и git думает, что допускается ошибка. В этом случае данная команда принудительно переписывает историю коммитов на удалённом репозитории (на локальном это делает git reset –soft, сохраняя все изменения)

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**git pull 'nazvanie\_sviazki' 'nazvanie vetki'** <-- skachaet repozitorii. Dlia etogo nado 'bit' v toi direktorii kyda xochesh 'polozit' proekt s gitHuba i eta direktoriia dolzna bit repozitoriem (t.e. zdes dolzzen rabotat git(git init)).

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**Спуливание удалённого репозитория на локальный (который совершенно никак не связан с удалённым)**

**git pull Nazvanie\_ydalennogo\_repozitoriia master\_eto\_vetka\_kyda\_pyllim --allow-unrelated-histories** - Позволяет спулить удалённый репозиторий к себе на локальный, когда есть ошибка **“fatal: refusing to merge unrelated histories”. ←** Эта ошибка возникает когда пытаешся в свой локальный репозиторий который включает в себя свою локальную историю коммитов и который никак не связан с удалённым репозиторием который хотим запулить в наш локальный, git как-бы это видит и не хочет сливать два проекта мол они вообще разные и вообще чё ты творишь так делать нельзя. спасает данная инструкция. Такая ситуация возникает например не только когда хочешь запулить какой то удалённый репо себе, но и когда хочешь свой запушить в удалённый, для этого придётся сначала спулить (произойдет автоматическое merge) себе удалённый, а затем пушить уже всё вместе на удалённый

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

git restore --staged .

Ety komandy ia primenil v sledyyschem variante. V ideia ia dobavil fail v kornevyy papky proekta, v

terminale nabral:git status i moi fail naxoditsia v unstaged zone (krasnii). Ia zdelall git add . i iz-za

etogo dobavilis v staging area vse faili neotslezzivaemie,vse stali zelenie, OPA, a takogo povedeniia mne

ne nado. Komanda git restore --staged . <--vernyla vse v unstaged zony

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**git remote remove <name>** <--ydaliaet ссылку (название) на удалённый репозиторий из локального

При этом сам удалённый репозиторий остаётся, его можно удалить зайдя в него в интернете.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**как удалить ненужные файлы с удалённого репозитория**

**git rm -r --cached .idea/ ←**Удаляет из локального репозитория папку .idea которая закэширована ну или любой другой файл указанный вместо него. Применимо в том случае когда файл gitignore не был создан изначально в проекте и на удалённый репозиторий попали ненужные файлы. Если просто создать, а затем ввести названия ненужных файлов в gitignore, а затем запушить, то это не поможет по отношению к тем ненужным файлам которые уже попали с первым пушем. Необходимо сначала:

откоректировать gitignore,

удалить поочереди с помощью **git rm -r --cached название\_файла** ненужные закэшированные файлы,

если на этом этапе сделать git status -то увидим удалённые файлы обозначенные зелёным цветом и красным gitignore так-как он пока не отслеживаемый. Делаем git add . тем самым добовляя gitignore и он становится зелёным, после этого можно git commit -m “name of the commit” и зате пушим всё на удалённый git push nazvanie\_ydalennogo\_repozitoriia nazvanie\_vetki\_na\_ydalennom\_kyda\_zotim\_zalit\_izmeneniia

ООО я бля хотел бы добавить что если гид работает в корневой папке, а в корневой папке есть ещо папка которую мы нехотим отправлять/комитить на удалённый репо и мы добовляем **название\_этой\_папки/** в игнор файл и … видим что наша папка благополучно комитится бля ёпта. Дак надо ее удалить из кэша !!! Удаляем из кэша, а git нам говорит что такого файла я вообще не знаю и с ним не работаю ??? Чё за …. Когда удаляем из кэша надо прописывать не название файла если этот файл/директория вложенные в корневой а путь от корневого файла да того который хотим удалить

Пример:

git rm –r –cached Programmin/Rezume/ <- - в этом случае мы удалим из кеша то что надо.

Такой же принцип надо указывать и в .gitignore . Пример:

ReadMe.txt

Programming/Rezume/ < -- Вот так!!!

Programming/Регистрации/ < -- Вот так!!!

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**Клонирование удалённого репозитория**

**git clone ссылка\_на\_удалённый\_репозиторий** - Пулит полностью удалённый репозиторий в данную директорию, тем самым полностью инициализирует работу с git-ом, т.е. не надо делать **git init**, не надо подгружать удалённый к этому локальному выполняя **git remote add "nazvanie\_удалённого\_Repozitoriia" ssilka na gitHub**, также как и не надо затем пулить **git pull nazvanie\_ydalennogo nazvanie\_vetki\_kyda\_pyllim**. Всё это заменяет **git clone**. Но гит будет работать начиная с корневой папки проекта, например есть папка1 в ней вызвали cmd и прописали git clone ссылка\_на\_удалённый, скачался проект под названием Zags, дак вот гит будет работать в Zags, но не в папка1.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**BRANCHING**

**git branch** <-- показывает наличие всех веток в данном репозитории и на какой ветке мы находимся она будет зелёного цвета и с переди будет стоять звёздочка

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git branch nazvanie\_novoi\_vetki** <--создаёт и dobovliaet novyy vetky. NO ne perexodit na nee.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git checkout nazvanie\_vetki** <--Переходит на указанную ветку.Tak mozzno perexodit s novoi na master, pri etom

vse chto bilo sozdano/izmenenno v novoi vetke kak-bi ischeznet iz pole videniia, но ostanetsia v novoi tak-kak mi bydem naxoditsia v master vetke, gde razrabotka idet svoim (otdelnim) etapom. Sootvetstvenno: vse chto bydet sozdanno posle branchinga v kazzdoi iz vetok bydet idti по индивидуальному сценарию разработки каждой из веток . Но надо помнить, что финальной (чистовой вариант) веткой остаётся всегда **master** и merge всегда выполняются в мастер . Y nas kak-bi 2 proekta (ny ili stolko skolko vetok). S pomosch`y dannoi komandi mozzno perexodit na lybyy vetky, ne tolko s novoi na master no i s master na drygie vetki.

Ogo esli zdelat git log to kolichestvo kommitov tozze bydet otlichatsia, kakbydto raznimi pytiami razoshlis, v kazzdoi vetke v zavisimosti ot vetki, t.e. izmeneniia i kommiti v odnoi vetke nevidni i ne ychitivaytsia drygoi - chto v principe logichno, kazzdaia vetka seichas eto otdelni proekt.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git merge 'nazvanie vetki kotoryy slivaem'** <--Слияние ветки в master (важно находится в master в момент совершения слияния(merge). После выполнения данной команды , если сейчас вызвать команду git log (будучи в master) то увидим, что все коммиты которые были в ветке которую хотим слить, объединились и находятся в ветке master. Proizoshlo sliianie. V principe posle sliianiia vozmozhno chto bydet ymestno ydalit vetky, faili i izmeneniia iz kotoroi mi ob`edinili s master...

**Dannoe sliianie nazivaetsia 'fast-forward'-**Tak-kak nikakix izmenenii v vetke master nebilo v to vremia poka mi rabotali v drygoi vetke to sliianie proizoshlo legko, gity bilo poniatno kak eto zdelat. При таком слиянии гит не делает никаких дополнительных коммитов.

No chto esli za vremia nashei raboti s drygoi vetkoi kto-to vnes izmeneniia v master? В etom slychae mozzet vozniknyt konflikt sliianiia когда будем сливать нашу ветку в мастер и при таком виде слияния git создаёт merge commit, такое слияние называется **‘recursive' strategy**. А merge коммит скорее всего будет называться ‘Merge branch nazvanie\_vetki’ и будет создан автомотически.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **удаление ветки** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**git branch -d nazvanie\_vetki\_kotoryy\_xotim\_ydalit** <-- ydaliaet vibrannyy vetky. Posle etogo mozzno proverit

s pomosch`y git branch skolko vetok ostalos. Но удаление пройдет только если удаляемая ветка была смёржена с какой либо.

**git branch -D nazvanie\_vetki\_kotoryy\_xotim\_ydalit** ← Так ветка будет удалена безвозвратно независимо от того, была ли она смёржина или нет. Будь внимателен!!!

**git push Nazvanie\_ydalennogo\_repozitoriia --delete nazvanie\_vetki\_kotoryy\_xotim\_ydalit** - удаляет ветку на удалённом репозитории. Например git push <remote\_repository\_name> --delete <branch\_name>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Всё про .gitignore файл**

Например: у нас есть проект под гитом, т.е. папка в которй была проведена команда git init, в ней есть файлы, мы их успешно закомитили. Мы решаем добавить в проект папку с исключениями .gitignore, если делаем git status то гит видит новый файл не отслеживаемый(т.е. gitignore ничем не отличается от других файлов, изменения в нём отслеживаются), добавляем его и комитим.

Сейчас каждый раз когда мы изменяем данный файл (gitignore) то гит его отслеживает как и любой другой, но файлы которые **прописаны/указаны** в .gitignore будут игнорироватся git-ом.(не всегда, читай до конца) Если теперь добавить папку (она пуста), гит не воспринимает это как изменения в проекте и если в этой папке будут файлы которые занесены в gitignore они также не будут отслеживаемыми. Но стоит добавить в новосозданную папку файлы не входящие в gitignore, то git это увидит, но увидит на этом этапе саму папку,(её название будет красным) её стоит добавить с помощью **git add .** После этого гит будет видеть сразу все файлы, находящиеся в данной папке и которые отслеживаемые (не входят в gitignore), на этом этапе эти файлы будут зелёными, дальше работать с ними можно как обычно, в дальнейшем гит уже будет отслеживать изменения и в этой папке.

Очень важно создавать и формировать gitignore на стадии создания проекта, так-как если данный файл будет создан в процессе, после нескольких коммитов то в последствии даже если в нём будут указаны/добавлены какие то очередные нежелательные файлы, то возможно они будучи зашифрованы в локальном репозитории будут всё равно комитедца, для этого **надо удалить закэшированные файлы на локальном репозитории**, а затем изменить настройки в gitignore. Смотри как удалить ненужные файлы, выше.

В данный файл заносятся файлы которые будут исключением при коммитах. Вопрос где должен распологатся данный файл при работе в IDE. -> в корневой папке проекта

1)-Для того чтобы добавить исключение(файл который не нужно загружать на сервер), необходимо полностью прописать его название.

2)-Существуют маски (сокращения в написании файлов)

а)- **\*.расширение** (например \*.txt) обозначает ,что все файлы с данным расширением будут игнорироватся гитом и не попадут в коммит

б)- **Название\_папки/** - в этом случае вся папка будет игнорироваться (но помни, что при внесении изменений в gitignore, эти изменения также нужно закомитить)

!!!Если документ вложен в корневой необходимо указывать путь + имя документа

ReadMe.txt

Programming/Rezume/

Programming/Регистрации/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Работа с GitHub из Intellij Idea**

Для загрузки проекта на GitHub из Idea нужен сам акаунт на GitHub.

Далее идут настройки где предлагается выбрать название репозитория на GitHub (т.е. как будет называтся репозиторий где будет лежать проект) и задать имя/название ссылки по которой локальный репозиторий будет обращаться к удалённому.

После этого будет создан репозиторий на GitHub-е.

**Как вариант можно работать с гит в IDE через Terminal, используя при этом все консольные команды описанные выше и не заморачиватся про интерфейсную интеграцию.**