# **Git**

**https://learngitbranching.js.org/**

Vytvoreniegitovskeho repozitaru:

* pravym klik na priecinok, GIT BASH HERE, nasledne otvori prikaz. riadok a napisem ‘**git** **init**’
* GIT BASH HERE dam vzdy vtedy, ked chcem pouzit nejaky prikaz z gitu

Stav git. repozitaru:

* Po GIT BASH HERE dam v prikaz. riadku ‘**git status**’, vypise mi v akej faze sa nachadza dany subor (untracked files – nesleduje git)

Zacat sledovat a upravit nejaky subor:

* Pomocou prikazu ‘**git add** text.txt’ zacneme sledovat subor text.txt
* Pokial zmenim subor text.txt, tak mi vytvori kopiu povodneho suboru(zostava v **staged area**- sledovany priestor) a novy upraveny subor, ktory pomocou ‘git status‘ sa mi zobrazi ako modifikovany(modified), zobrazene budu teda dva subory
* Ak chcem mat aj novy upraveny subor v staged area, musim dat znova git add toho suboru

Zdielanie na lokalny a vzdialeny repozitar:

* Prikaz ‘**git commit’** (lokalny)
* Prikaz ‘**git push**’ (vzdialeny, predtym ale musi byt v lokalnom repozitari aby mohol ist na vzdialeny

Nastavenia gitu:

* ‘**git config**‘
* ‘**git config --global user.name “dani“**‘ – nastavi pre vsetky projekty(global) meno dani
* ‘**git config --global user.email “danieva14@gmail.com“**‘ – nastavi email globalne
* ‘**git config --l**‘ -vypise vsetko z konfiguracneho suboru
* Ak nedam global, bude to len pre konkretny projekt
* Po vypise konfiguracneho suboru sa zobrazi **core.editor** (graficky editor), ktory mozno zmenit

Zobrazenie zmeny suboru:

* ‘**git diff**‘ nam zobrazi zmeny medzi povodnym suborom v staged area a medzi upravenym suborom v modified
* ‘**git diff --staged**‘ nam zobrazi co je v staged, co mozem nasledne komitovat do lokalneho repozitaru

Komitovanie na lokalny repozitar:

* ‘**git commit --m ‘sprava komitu‘ ’**

Vetvenie a spajanie vetiev v gite:

* Master sa nazyva ten, kto daval git init, to je akoby hlavny uzivatel hlavnej vetvy
* Vetvy sa zobrazuju az po komitnuti, kym nespravim ziadny komit, dovtedy ziadna vetva neexistuje
* ‘**git branch’** - zobrazi vsetky vetvy, hviezdickou oznacuje na ktorej sme vetve
* ‘**git branch feature’** -vytvori novu vetvu s nazvom feature, ale neprepne sa do nej
* ‘**git checkout feature’** - prepnutie do vetvy s nazom feature
* ‘**git checkout -b feature2’** -vytvoru novu vetvu feature2 a rovno sa do nej aj prepne
* **‘git branch -d hotfix‘** -mazanie vetvy hotfix(nemozno zmazat vetvu v ktorej sa nachadzas)
* Tiez nemozno zmazat, ak neni mergnuta s upstream vetvou alebo headnuta s vetvou v ktorej sa nachadzam
* **Upstream branch** -je vetva, ktoru si stiahnem napr z netu, je to nasa vetva ale na remote pocitaci
* **HEAD** -pointer na vetvu, v ktorej sa nachadzam
* **‘git branch -D hotfix‘** - nasilne mazanie vetvy
* **‘git merge hotfix‘ -**spojenie vetvy v ktorej sa nachadzam s vetvou hotfix
* **‘git merge --abort‘** -da sa to do stavu pred mergovanim

Postup mergovania:

* Mam dve vetvy s nazvami merge1 a merge2. Pokial v merge1 upravim nejak subor, ktory sledujem a nasledne ho komitnem, tak mi ostane v tejto vetve, pokial sa prepnem do druhej vetvy a text prepisem,ulozim, pridam do gitu a komitnem, v druhej vetve sa bude nachadzat iny text a podla toho v ktorej vetve budem,ten dokument budem mat dostupny. Pokial ich chcem spojit pomocou ‘**git merge**‘, tak moze dojst ku konfliktu, ktory sa mi v danom textovom dokumente zobrazi. Ten musim nasledne upravit a pridat do gitu a komitnut a toto bude ten novy dokument.
* Mam aj druhy sposob ako vyriesit konflikty pri mergovani a to taky, ze pouzijem **‘git merge --tool -help‘** pre zobrazenie toolov(viac info ako to pouzit vo videu c. 10).

Tagy:

* Annotated tag – uchovava viac informacii
* Lightweight tag
* **‘git tag‘** -vypise verziu
* ‘**git tag -a v0.1 -m ‘verzia 0.1’** ’ -takto sa udava verzia tagu, to ‘a’ udava annotated tag
* **‘git tag -l “v1.\*“ ‘** – najdi vsetky verzie 1.
* **‘git log --pretty=oneline ‘ –** vypise tagy predchadzajucich suborov, nemusime tagy udavat my, vytvori si ho aj sam git
* **‘git tag -a v0.3 e80e9bc9 -m 'verzia v0.3' ‘ –** takto mozem vytvorit z nejakeho existujuceho komitu, staci mi udat len cast toho tagu, ktory som zistil cez predchadzajuci prikaz
* **‘get show v0.3‘ -**zobrazi informacie o tagu
* **‘git checkout -b v0.1\_hotfix1 v0.1 ‘** – vytvor mi novu vetvu a rovno sa do nej prehod, na konci je najprv nazov a potom verzia(v0.1), z ktorej cerpa
* **‘git log --graph‘ –** nejaky suhrn toho co som robil

Vytvorenie suboru v prikazovom riadku:

* cat > info.log

nejaky subor s nejakym textom

* nasledne ak chcem zadavat znova prikazy,musim stlacit CTRL+C

Ignorovanie suborov:

* cat > **.gitignore**

\*.log

!error.log

/dani

-vytvoril som si novy subor s koncovkou gitignore, v ktorom sa specifikuju, ze ma ignorovat vsetky subory s koncovkou .log, okrem error.log (vykricnik je negacia,preto okrem toho)... /dani bude zas ignorovat priecinok dani...podrobnejsie kody su v pdf dokumente, ako pisat ignory

Oprava komitu a inych veci:

* ‘git commit -m ‘nejaky commit‘ ‘ – vytvorenie komitu
* ‘git add nejakySubor‘ -po vytvoreni komitu chces pridat este don subor s nazvom nejakySubor
* ‘**git commit --amend -m ‘komit1‘** ‘ – tento prikaz posledny commit zmaze a prida k nemu subor nejakySubor a vytvori z toho novy kommit, nebudu tam teda dva komity, cize oprava komitu
* ‘**git checkout -- text.txt**‘ -z modifikovaneho suboru nam zmaze zmeny
* ‘**git reset -- nejakySubor**‘ -opak prikazu git add, nejakySubor mi zo staged file da spat do untracked files

Zdielanie a stahovanie s REMOTE serverom:

* ‘**git pull <remote>**’ - miesto remote musim zadat url stranku remote servera, pull mi stiahne udaje z tohto servera do vetvy na ktorej aktualne som a potom spravi merge s mojimi datami… ak dam len git pull, tak udaje stiahne z originu
* ‘**git fetch <remote>**‘ -to iste ako git pull, akurat nespravi merge,mozme si zmeny najprv pozriet, ak nezadam remote, zas to bere z originu
* ‘**git push <remote> <branch>**‘ -pomocou git push ja odoslem zmeny na vzdialeny server, pokial dojde ku konfliktu, tak mi to nedovoli odoslat, preto je lepsie najprv stiahnut udaje a mergovat a az potom pushnut, do branch nastavim ktoru vetvu chcem preniest
* ‘**git branch -R**‘ -vypise zoznam remote vetiev

Klonovanie z remote servera:

* Najprv zo stranky github dam clone or download a skopirujem link
* Vo svojom pracovnom adresari cez git bash zadam prikaz ‘**git clone <skopirovany link>**’
* ‘ls‘ -vylistovanie
* ‘cd gitflow/‘ otvori priecinok gitflow, musim byt nastaveny v prikazovom riadku o jedno vyssie

Praca s remote repozitarmi:

* ‘**git remote**‘ –vylistovanie
* ‘**git remote -v**‘ -vypise aj URL adresu repozitarov
* ‘**git remote add jw <https link>**‘ -takto pridam repozitar, ktory som si pomenoval ako jw za ktorym nasleduje link
* ‘**git pull**‘ -ak dam prikaz len takto bez remote repozitaru, bude stahovat z originu
* ‘**git pull jw master**‘ -stahovanie uz konkretneho repozitaru z vetvy master
* ‘**git remote show origin**‘ -viac informacii o remote-e
* ‘**git push origin master**‘ -tlacim udaje na remote server, ak ale nemam k nemu pristup(povolenie), tak sa mi to nepodari, ak medzicasom niekto pushol na remote, tak ho musim najprv stiahnut(nech sa merguje s mojim projektom) a az potom ho mozem pushnut
* ‘**git remote rename jw novyNazov**‘ -premenovanie remote repozitaru z jw na novyNazov
* ‘**git remote remove jw**‘ -mazanie remote repozitaru
* ‘**git remote set-url jw <SSH kluc>**‘ -su dva sposoby linkov na github, cez https sme si uz ukazovali, toto je cez SSH kluc
* ‘**git branch -r**‘ -vypisanie remote branchov, bez -r by sa zobrazili len moje lokalne branche
* ‘**git checkout -b next jw/next**‘ -vytvorenie aj prepnutie branche z remote repozitaru, pricom next je nazov z remote za lomitkom a jw/next je nazov mojej vytvorenej branche
* ‘**git checkout -b next jw/next** ‘ -takyto prikaz sa robieva aj ked **fetchujem**,pretoze samotny fetch len zobrazi zmeny,ale nemerguje ich, zmeny si do branchu ukladam teda tiez tymto sposobom

Postup vytvorenia remote repozitaru:

* Vytvorim repozitar na gite
* Na moj priecinok otvorim git bash a dam **‘git init‘**
* Spravim **‘git add repo.txt‘** a **‘git commit -m ‘prvy commit‘ ‘**
* **‘git remote add origin1** [**https://github.com/Dani126/skuska.git**](https://github.com/Dani126/skuska.git)**‘** -pridanie remote repozitaru pod nazvom origin1
* **‘git push -u origin1 master ‘** -u znamena upstream(smerom od nas), origin a master znamenaju ze kam to chcem poslat, ak to rejectne, tak musim dat pull najprv
* aby som nemusel pri git pull zadavat este aj remote a branch za tym, mozem si to nastavit pomocou prikazu **‘git branch --set-upstream-to=origin1/master master‘** kde origin1 je remote a branch je master, nasledne mozem pouzivat uz len ‘git pull‘

Praca s vetvami a tagmi na remote repozitari:

* **‘git fetch‘ -** tymto prikazom sa mi zobrazia nove branche a nove taky z remote-u
* **‘git tag‘ -** tento prikaz zobrazi potom uz vsetky tagy z remote-u
* **‘git branch -a‘ -**ak chcem zobrazit vsetky branche, musim pridat ‘-a‘, inak by mi len lokalne zobrazilo
* **‘git checkout -b develop origin1/develop‘ -** takto vytvorim(skopirujem z remote) a sa prepnem do branche develop
* **‘git push origin1 --delete v0.1‘ -**posielam zmeny na repozitar, v ktorom urcujem ze chcem mazat v remote origin1 a za delete zadavam ze co, v tomto pripade chcem mazat tag, tieto veci viem robit aj rucne cez ikony na stranke github
* **‘git push origin1 --delete develop‘ -**zmazanie branche na remote, nemusim sa prepinat na iny, lebo mazem len na remote,lokalne mi zostane

Spolupraca na projekte:

* Ak mame nejaky spolocny projekt,svoje zmeny poslem tak, ze dam **new pull request** a **create pull request**

SSH kluc:

* **‘git remote set-url origin1** [**git@github.com:Dani126/skuska.git**](mailto:git@github.com:Dani126/skuska.git)**‘** -pripojenie sa k remote cez SSH kluc
* **‘ssh-keygen -t rsa -b 4096 -C “danieva14@gmail.com“ ‘** -generovanie ssh kluca
* **‘eval $(ssh-agent -s)‘** -musim najprv vytvorit agenta az potom pridat kluc
* **‘ssh-add ~/.ssh/id\_rsa‘-**pridanie SSH kluca
* Na ceste C:\Users\danie\.ssh mam svoje SSH kluce, ten co je pub, otvorim a skopirujem, nasledne na stranke github pojdem do settings-> ssh and gpg keys ->new ssh key-> do key prilepim kluc a do title mu dam nejaky nazov

Bitbucket je take ako github, akurat ma privatny repozitar zadarmo

Git flow:

* Do vetvy **master** sa nedava ziadny kod, pouziva sa len pre tag verzie, pripadne tam uz pridavam finalnu verziu ktora ide k zakaznikovi
* Do vetvy **develop** sa pise kod
* **Hotfix** je urceny je rychle opravy
* **Feature branche** sa bere ako nejaka pomocna vetva na skusanie
* **‘git checkout -b myfeature develop’**- vytvorim novu vetvu myfeature z developu
* V myfeature ked si spravim kod, ktory chcem vlozit spat do developu, tak sa prepnem do developu a az potom merguje. Mergovat treba z vetvy, do ktorej sa chcem dostat. Prikaz je nasledujuci: **‘git merge --no-ff myfeature‘** -to no-f znamena, ze ponecha informacie o vetve, ktoru mergujem(necha vetvu feature)
* Feature branche sa pouziva z developu, ked uz nieco finalizujem,ale este to nedavam do mastra, napr. posledne upravy atd. Pokial by som daval uz do mastra, tak musim zrkadlovo dat aj do developu,nech aj tam je ta verzia: **‘git checkout -b release-1.2 develop‘** a este otagujem **‘git tag -a v0.1 -m ‘verzia 0.1’** **‘**

Git aliaces:

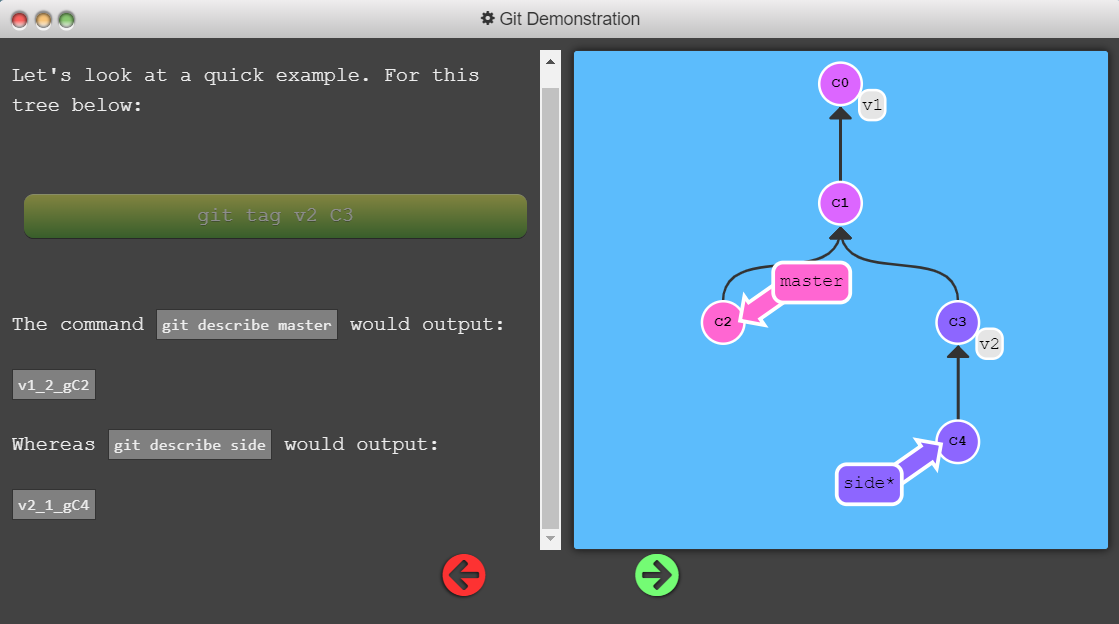
* ‘**git config --global alias.br branch**‘ - spravil som si skratku a miesto ‘git branch‘ napisem ‘git br‘
* ‘**git config --global alias.last ‘log -1 HEAD‘** ‘ -ak je dlhsi prikaz, tak ho pisem do uvodzoviek, pouzijem to teda prikazom ‘git last‘

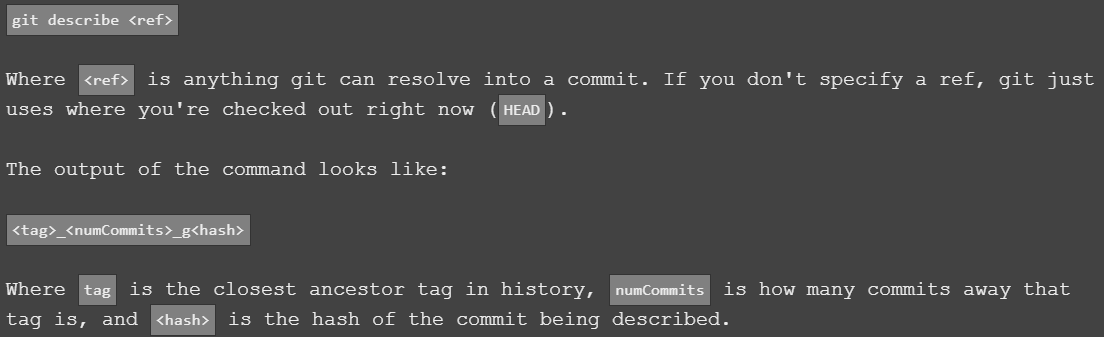
Git stash a clean a rebase:

* **‘git stash‘** -tento prikaz pouzijem vtedy, ak mam nejaky subor, ktory nechcem komitnut,ale nechcem aby sa mi tieto zmeny stratili pri prepinani do inej vetvy, tak ich dam zatial do stashu, kde su nedorobene subory
* Pred stashom treba ale dat git add, subor musi byt sledovany, lebo na untracked files sa neda pouzit
* Existuje ale prikaz, ked chcem untracked files hodit do stashu a to: **‘git stash -u‘ -** tento prikaz hodi aj sledovane aj nesledovane subory do stashu
* **‘git stash list‘-**vylistovanie stashov
* **‘git stash apply stash@{0}‘** - stash@{0} je nazov stashu, tymto prikazom ich viem znova pouzit, tie nedokoncene subory
* **‘git stash drop stash@{0}‘ -** vymazanie stashu stash@{0}
* **‘git stash pop stash@{0}‘ -** tento prikaz najprv aplikuje stash a nasledne ho zmaze, cize je to apply a drop dokopy
* pri mazani stashov sa meni pozicia, ak mam dva stashe a zmazem nulu, tak jednotka sa zmeni na nulu
* **‘git stash branch testchanges‘ -** tento prikaz mi stash hodi do novej vetvy testchanges a nasledne stash drobne, je to dobre na to, ked chcem skusit mergovat dva subory mimo svojej hlavnej vetvy, ak nenastanu konflikty, pripadne ich upravim, mozem nasledne mergovat s hlavnou vetvou
* **‘git clean‘ -** zmaze subory okrem ignorovanych suborov(asi)
* **‘git clean -n‘ -**tento prikaz sluzi pre simulovane mazanie,ukaze mi,co by sa stalo ak by som sa pokusil zmazat
* **‘git clean -f‘ -**mazanie nasilu
* **‘git rebase master‘ –** sme vo vetve bugFix, bugFix vetva nahradi vetvu master a potom sa pôvodná zruší(prevzatie vetvy) s tym ze povodna zostane tiez
* **‘git checkout HEAD^‘ –** príkaz prepne na dokument o jedno vyššie( nie na vetvu)
* **‘git branch –f master HEAD~3‘ –** prepne vetvu master o tri urovne vyssie v stromovej strukture (f znamena force), s tym ze pointer nam aktualne ukazuje na inu vetvu, pokial by som to chcel spravit s aktualnou vetvou, musim pouzit checkout, to cislo udava posunutie od mojho **aktualneho** pointera smerom nahor.

Toto moze mat vyuzitie tak, ze dam pointer(HEAD) na konkretny kommit a nasledne pouzijem **git branch -f master HEAD~0 ,** tym dosiahnem to, ze presuniem vetvu master smerom nadol na ktory kommit chcem

* **git reset HEAD~1** - vráti commit aktualneho pointera o jedno vyssie a stare zrusi
* **git revert HEAD -** spravi kopiu commitu na ktory aktualne ukazuje, napr z C1 spravi C1’
* **git cherry-pick C2 C4 -** pod aktualny pointer spravi kopiu C2‘ a C4‘ ako dalsie commity, takto mozem zozbierat a spravit si poradie commitov z roznych branchov, poradie sa urcuje v prikaze, teda skopiruje pod seba lubovolny commit
* **git rebase -i HEAD~4 -** od aktualnej vetvy(na ktoru pointer ukazuje) zoberie 4 skorsie commity, z tych si pomocou USER INTERFACE mozem vybrat(necham oznacene ktore chcem kopirovat, mozem menit aj poradie) z ktorych si chcem spravit kopie(napr. C2, C4 )a nasledne z tychto vytvori paralelnu branchu aj sa do nej prepne(C2‘, C4‘ pricom vetva ukazuje na C4‘) a stare C1 az C4 **zahodi**
* **git commit --amend -** spravi paralelny kommit nie seriovy(nie pod neho ale vedla)
* **git tag v1 c2 -**  prida kommitu c2 verziu v1
* **opis kommitov :**

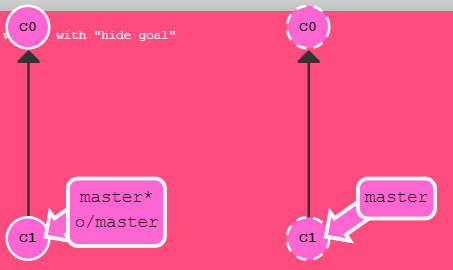
****

****

* **git checkout master^** - prepne o uroven vyssie na prveho rodica
* **git checkout master^2** - prepne o uroven vyssie na druheho rodica
* **git checkout HEAD^2** - tymto sposobom vyberam rodica, v tomto pripade druhu vetvu pokial idem smerom hore(aby sa dalo pri vetveni rozhodnut ktorou vetvou pokracovat, na to je dobry tento prikaz)
* **git checkout HEAD~2** - v tomto pripade sa posunie o dva hore v danej vetve
* **git checkout HEAD~^2~2** - tento prikaz znamena ze najprv ide o jedno hore (prikaz ~) vo vetve, nasledne vyberie druhu vetvu ktorou chce pokracovat (prikaz ^2) a nakoniec sa posunie v danej vetve o dva hore (prikaz ~2), pozor na cislovanie, vzdy nizsi kommit je prvy)

**REMOTE**

* git clone (klonovanie na remote z lokalu??)



* pomenovanie branchov ak pracujem uz s remotetom:



Toto pomenovanie je len na strane lokalu, pretoze tu potrebuje rozlisit ci dana brancha je remoteova alebo lokalova

* **git fetch**, stiahne mi z remote(dorobi kopiu, nevyraba novy) a doplni do lokalu co mi na lokale chyba, viacere vetvy
* **git pull** = git fetch a git merge dokopy spravi
* **git fakeTeamwork foo 3 -** spravi 3 kommity za sebou na remote repozitari
* **git push -** skopiruje rozdiel z lokalu do remote-u
* **git pull --rebase** - spravi dokopy pull a rebase
* **git push origin master** - da pointer na master branchu a da natlaci na remote depozitar original(origin asi nie je nazov ale konstanta ze ide o original, original pre master je o/master), pokial by nebol pointer na ziadnej branchi, tak nie je mozne spravit push, doslo by k chybe, napr pri pouziti HEAD sa neda pushovat
* **git push origin master:bar** - pokial chcem na remote dat branchu master do ineho repozitara napr s nazvom bar a nie do originalu mastera
* **git push origin foo^:master -** vezme branchu foo a da pointer o jedno vyssie, potom spravi pull do mastera na remote
* **git push origin master:newBranch -**natlaci branchu master do novej branche na remote s nazvom newBranch, nemusi na remote existovat,vytvori ju
* **git fetch origin foo -** pohlada foo na repozitari a na lokali spravi update podla remote
* **git fetch origin foo~1:bar** - pohlada na remote foo o jedno vyssie(~1) a skopiruje ho do baru na lokale
* **git push origin :foo -** zmaze repozitar foo z remote
* **git fetch origin :bar** - toto len spravi novu branchu bar, pretoze nema co fetchovat
* **git pull origin bar:foo** - pokial mam pointer na branchu master a pouzijem tento prikaz, tak mi skopiruje bar z remotu do foo branche na lokale a spravi merge s branchou na ktoru sa ukazuje(master)