[**Web Programiranje**](https://www.webprogramiranje.org)

**Skripte**

**Git naredbe**

**.gitignore**

Pre aktiviranja Git-a potrebno je sprečiti čuvanje i praćenje izmena nepotrebih fajlova. Najčešće su to fajlovi koje koristi editor ili fajlovi koji nastaju kompajliranjem i sl. Fajlove ili cele direktorijume koje ne želimo da pratimo definišemo u okviru **.gitignore**. **.gitignore** fajl se smešta u okviru projektnog foldera.   
Ovaj fajl se popunjava odgovarajućim naredbama uz čiju pomoć Git saznaje koje fajlove ne treba da prati. Princip obeležavanja fajlova i foldera koje ne želimo da pratimo unutar **.gitignore** je sledeći:

* Upišemo ekstenzije fajlova sa razmakom koje ne želimo da pratimo kao npr.

.swo .swp

* Upišemo foldere koje ne želimo da pratimo:

ime\_foldera

U slučaju da unutar foldera postoji fajl koji ipak treba da se prati upišemo:

!nekiFajl.txt

* Upišemo folder i njegove podfoldere koje ne želimo da pratimo:

ime\_foldera/ime\_podfoldera

Pregled grana

* Upišemo folder i sve njegove podfoldere koje ne želimo da pratimo:

ime\_foldera/\*

* Upišemo sve objekate i arhive koje želimo da ignorišemo:

\*.[oa]

Postoji na internetu veliki broj primera *.gitignore* u zavisnosti kakav je projekat u pitanju (WordPress, Android….), a odličan sajt za nalaženje adekvatnog možete naći na sledećem linku: [toptal/.gitignore](https://www.toptal.com/developers/gitignore).

**Prekid praćenja**

Veoma je bitno napraviti *.gitignore* file pre iniciranja Git-a jer ukoliko Git već započene praćenje odredjenog fajla on neće prestati da ga prati čak i ako taj neželjeni fajl naknadno ubacimo u **.gitignore**. U nastavku su opisane potrebne radnje za prekid prećenja nekog fajla.

**a) Prekid praćenja bez fizičkog brisanja**

Ukoliko su greškom commit-ovani neki nebitni fajlovi za projekat, čije promene ne želimo da čuvamo, ali ne želimo ni da ih fizički obrišemo iz radnog direktorijuma, potrebno je da naredimo git-u da prestane da ih prati (tj. da ih izbaci iz repozitorijuma) sa naredbom:

git rm --cached naziv\_fajla.ekstenzija

|  |  |
| --- | --- |
|  |  |

Naredba za prestanak praćenja direktorijuma je:

|  |
| --- |
| git rm -r --cached imeDirektorijuma |

gde **-r** govori Git-u da stane da prati sve podirektorijume i fajlove unutar njega.

Nakon naredbe za prekid praćenja, fajl se nalazi u prostoru pripreme (index). Da bi sprečili da fajlovi na sledećem *commit-u* ponovo ne udju u repozitorijum, potrebno je te fajlove pre *commit-ovanj*a, ubaciti u **.gitignore** fajl.

NAPOMENA:   
Obratiti pažnju da ukoliko naziv fajla sadrži razmak izmedju reči, neophodno je staviti ga u znake navoda, kao na sledećem primeru:  
git rm "neki naziv fajla sa razmacima.ekstenzija"

**b) Prekid praćenja sa fizičkim brisanjem**

Ukoliko postoji potreba da se fajl pored izbacivanja iz repozitorijuma i obriše iz **radnog direktorijuma** koristimo naredbu:

|  |  |
| --- | --- |
|  | git rm imeFajla |

Ukoliko je u pitanju direktorijum koristimo naredbu:

|  |  |
| --- | --- |
|  | git rm -r imeDirektorijuma |

gde **-r** govori Git-u da stane da prati sve podirektorijume i fajlove unutar njega. Nakon naredbe za brisanja fajlova je takodje neophodno uraditi commit (bez prethodnog slanja u index).

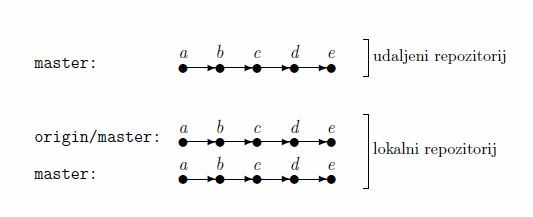
Više o ovoj naredbi pogledajte na [oficijelnoj stranici](https://git-scm.com/docs/git-rm)

**Lokalni repozitorijum**

Kada želimo da aktiviramo Git tj. da počnemo da pratimo izmene u projektu, potrebno je da unutar projektnog folder-a iniciramo pravljenje **“radnog git repozitorijuma”** koji se vrši na sledeći način:

|  |  |
| --- | --- |
| 1  2 | cd projektni\_folder  git init |

Ova naredba unutar našeg projekta kreira novi folder **.git** (tzv. “*repozitorijum”*) i u njemu će se nalaziti celokupna baza o našem projektu. Od tog trenutka Git prati projekat i radi snapshot-ove stanja kad uradimo commit. Ukoliko želimo da Git više ne prati naš projekat dovoljno je da obrišemo ovaj folder.



Lokalni repozitorijum je u specifičnoj vezi sa udaljenim jer je on “svestan” postojanja udaljenog repozitorijuma. U okviru lokalnog repozitorijuma uvek postoje minimum dve grane (vidi sliku):

1. Grana pod nazivom **“origin/master”** koja je lokalna verzija (kopija) udaljenog repozitorijuma. Na ovu granu se ne mogu slati naše izmene, već ona služi da sa nje uzmemo izmene (merge) jer je ona praktično posrednik tj. medjukorak za dobijanje izmena sa udaljenog repozitorijuma.
2. Grana pod nazivom **“master”** je naša lokalna grana direktno povezana za projekat (u kojoj pravimo commit-e).

**Udaljeni repozitorijum**

Udaljeni repozitorijum kao što mu i sam naziv govori **nije direktno vezan sa projektom** na kome radimo već se nalazi na nekom drugome mestu (najčešće na nekom serveru na internetu npr. Github, Bitbucket…). Ovaj repozitorijum nije zamišljen da se njemu direktno šalju izmene, već je neophodno da se izmene prvo sačuvaju u lokalu, a tek onda da se šalju na udaljeni repozitorijum.   
Na udaljeni repozitorijum može “(*“push-ovati”*)” izmene **samo onaj koji ima ovlašćenja**, dok saradnici bez privilegija imaju opciju da pošalju izmene uz zahtev za prihvatanje njihovih izmena (tzv. *“pull request”*).

Najvažniji razlozi za postojanje udaljenog repozitorijuma su: pravljenje sigurnosne kopije i olakšana saradnja više učesnika na istom projektu.

NAPOMENA:   
“origin” je alias za punu URL adresu online repozitorijuma (npr. https://url\_link\_udaljenog\_repozitorijuma). Ovakvo skraćeno predstavljanje URL-a nam omogućava jednostavniji rad sa naredbama jer izbegavate da koriste ogroman URL svaki put kada vam je u naredbi potreban. Koji tačno link predstavlja ovaj alias može se proveriti sa naredbom:

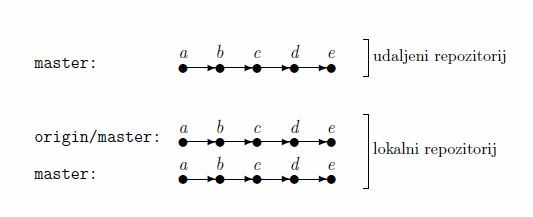
|  |  |
| --- | --- |
| 1 | git remote -v |

Kada se u naredbama pominje neka grana na udaljenom repozitorijumu (npr. master) potrebno je pored imena grane dodati i link do udaljenog repozitorijuma npr:

|  |  |
| --- | --- |
| 1 | git push https://url\_link\_udaljenog\_repozitorijuma master |

Znajući da je **“origin”** alias za ovaj link, sada se “master” grana na udaljenom repu može označiti jednostavnije sa **“origin master”** (grana na slici pod nazivom “master”). Pa bi komanda iz prethodnog primera izgledala ovako:

|  |  |
| --- | --- |
| 1 | git push origin master |



Više informacija o udaljenom repozitorijumu može dobiti sa naredbom:

|  |  |
| --- | --- |
| 1 | git remote show origin |

Šta je “Bare repo”?   
**Bare repo** je specijalna vrsta udaljenog repozitorijuma koji sadrži samo **.git** direktorijum. Takav repozitorijum nema radnu verziju projekta tj. nije povezana sa nekim lokalnim repozitorijumom koji ima radni direktorijum.

**Povezivanje lokalnog i udaljenog repozitorijuma**

Ovde postoje dva slučaja koji se razlikuju zavisnosti od u kojoj je fazi projekat kada povezujemo lokalni i udaljeni repozitorijum.

**a) Imamo postojeći lokalni repo a pravimo udaljeni repozitorijum**

Pravljenje novog udaljenog repozitorijuma (koji će da postane **bare repozitorijum**) se izvodi sa naredbom:

|  |  |
| --- | --- |
| 1 | git config --bool core.bare true |

NAPOMENA:   
Ovo nije potrebno ukoliko se koristi neki od online servisa kao što je *GitHub* ili *BitBucket*, jer se to uradi servis pri kreiranju praznog repozitorijuma.

Kada imamo “bare repository” (lokalni ili na cloudu) potrebno da ga povežemo sa lokalnim, a to radimo tako što ćemo lokalnom repozitorijumu dati do znanja koji je njegov udaljeni repo sa sledećom naredbom:

|  |  |
| --- | --- |
| 1 | git remote add origin link\_do\_udaljenog\_repozitorijuma |

Sa prethodnom naredbom samo praktično definisali url adresu koju predstavlja **“origin”**. Kontrolu dobro obavljenog posla možemo izvršiti sa naredbom:

|  |  |
| --- | --- |
| 1 | git remote -v |

Nakon čega bi trebalo da izadje poruka :

|  |  |
| --- | --- |
| 1  2 | origin https://url\_link\_udaljenog\_repozitorijuma (fetch)  origin https://url\_link\_udaljenog\_repozitorijuma (push) |

**Primer:**

Dodavanje repozitorijuma sa **GitHub-a** bi izgledalo ovako:

|  |  |
| --- | --- |
| 1 | git remote add origin https://github.com/username\_git\_korisnika/naziv-udaljenog-repozitorijuma.git |

Dok bi povezivanje na **BitBucket-u** izgledalo ovako:

|  |  |
| --- | --- |
| 1 | git remote add origin https://username@bitbucket.org/username/naziv-udaljenog-repozitorijuma.git |

Napomena: na BitBucket-u ovu naredbu dobijate čim se napravi novi repo ukoliko se izabere opcija **“imam postojeći projekat”**

Ovaj postupak se koristi samo kada na udaljenom repu imamo [“bare repozitorijum”](https://www.webprogramiranje.org/git-komande/#bare_repo) (prazan repo koji sadrži samo **.git** direktorijum), dok u ostalim slučajevima koristimo pristup objašnjen u narednom delu **“kloniranje udaljenog repozitorijuma”**.

**b) Kloniranje udaljenog repozitorijuma**

Ovaj slučaj podrazumeva da imamo postojeći projekat sa aktiviranim Git-om, a želimo da napravimo lokalnu kopiju. **Kloniranje** predstavlja kopiranje tog udaljenog repozitoriuma, stim što postoji izvesna razlika od samog prostog kopiranja, jer pored toga novi (lokalni) repozitorijum postaje *“svestan”* da je **on kopija nekog udaljenog repozitorijuma** i on čuva informaciju o repozitorijumu iz kojeg je nastao.

**b.1) Kloniranje repo samo sa jednom granom**

Ovaj slučaj je jednostavniji i rešava se u dva koraka. Prvi korak je da u terminalu dodjemo do foldera gde držimo sve projekte (npr. folder pod nazivom “projekti”):

|  |  |
| --- | --- |
| 1 | cd projekti |

Tek kada se u terminalu nadjemo u željenom folderu možemo da izvršimo naredbu za kloniranje:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | //ukoliko je u drugom folderu:  git clone /path/to/repo    //ukoliko nije na istom sistemu a možemo mu pristupiti preko SSH:  git clone username@remote\_system\_ip:/path/to/repo/on/remote      //ukoliko je repo na internetu:  git clone url\_link\_do\_udaljenog\_repozitorijuma |

Sa prethodnom komandom će biti u direktorijumu napravljen folder pod istim imenom kao projekat koji klonirama. Ukoliko želim da se taj folder drugačije zove potrebno je na prehodnu komandu dodati i novi naziv folder-a:

|  |  |
| --- | --- |
| 1 | git clone url\_link\_do\_udaljenog\_repozitorijuma novi\_naziv\_foldera\_projekta |

**Primer**

|  |  |
| --- | --- |
| 1 | C:\folder\_sa\_git\_projaktima> git clone https://github.com/username\_git\_korisnika/naziv-udaljenog-repozitorijuma.git |

**b.2) Kloniranje repo-a koji ima više grana**

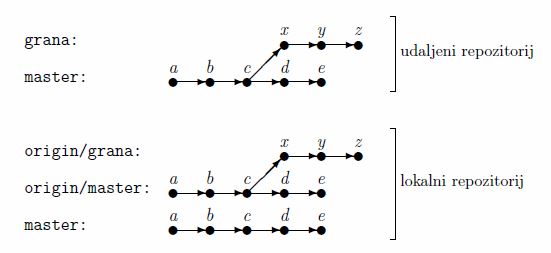
Ovaj slučaj je komplikovaniji jer neke stvari moramo dodatno da uradimo. Početak je isti, startujemo naredbu za kloniranje:

|  |  |
| --- | --- |
| 1 | git clone url\_link\_do\_udaljenog\_repozitorijuma |

Medjutim kada se ispita stanje grana sa naredbom:

|  |  |
| --- | --- |
| 1  2 | cd klonirani\_projekat  git branch |

Dobijemo obaveštenje da se u kloniranom repozitorijumu nalazi **samo “master” grana** koja se **ne račva** (vidi “master” na dnu slike).



Ovaj problem ćemo rešiti zahvaljujući tome što se u lokalu naša pomoćna grana “origin/master” ipak račva (vidi na slici “origin/master” i “origin/grana”), i to možemo da iskoristimo. Stanje ove pomoćne lokalne kopije udaljenog repzitorijuma možemo proveriti sa naredbom “branch” uz flag **“-a”**:

|  |  |
| --- | --- |
| 1 | git branch -a |

Kao rezultat prethodne naredbe se vidi lista svih origin/grana. Da bi se u i našem radnom direktorijumu videle i druge grane, potrebno je (iako se one ne vide) prebaci na svaku od njih i sa naredbom (napomena: bez korišćenja imena origin/….):

|  |  |
| --- | --- |
| 1 | git checkout naziv\_grane |

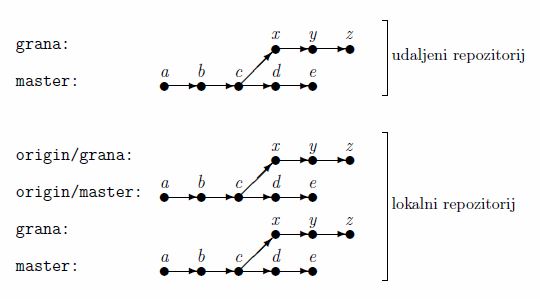
Nakon čega će git poslati sledeću poruku:

|  |  |
| --- | --- |
| 1  2 | Switched to a new branch 'naziv\_grane'  Branch 'naziv\_grane' set up to track remote branch 'naziv\_grane' from 'origin'. |

Kao što se iz same poruke vidi git je automatski napravio granu i prebacio nas u nju. Ovo možemo proveriti sa naredbom:

|  |  |
| --- | --- |
| 1 | git branch |

Nakon čega se vide i druge grane (isto kao na slici ispod).



**Delimično kloniranje repozitorijuma**

Git repozitorijum sadrži celu istoriju projekta, i ukoliko projekat dugo traje može biti prilično veliki. Zato ako želimo skinuti projekat samo zato da bi pogledali njegov kod, a da nas ne zanima nas cela istorija, moguće je klonirati samo nekoliko zadnjih commitova sa naredbom:

|  |  |
| --- | --- |
| 1 | git clone --depth 5 --no-hardlinks git://github.com/adresa\_repozitorijuma.git |

**Prekid veze sa udaljenim repo**

Ukoliko želimo da raskinemo vezu lokalnog repozitorijuma sa nekim udaljenim (*“origin”*), koristimo naredbu:

|  |  |
| --- | --- |
| 1 | git remote remove origin |

Nakon ove naredbe naredba git remote -v ne daje nikakve podatke.

**Stanje i pregled promena**

**Git status**

Kontrola da li imamo promene na fajlovima u odnosu na zadnji *commit*, se vrši sa naredbom:

|  |  |
| --- | --- |
| 1 | git status |

Ako je stanje na radnoj verziji našeg projekta potpuno isto kao i u zadnjoj verziji git repozitorijuma, onda će nas git obavestiti da nemamo ništa za commit-ovanje. U suprotnom, će istaći koji fajlovi su izmenjeni.

**Pregled promena koda**

**diff**

Ispisivanje svih razlika izmedju trenutnog stanja projekta na lokalu i zadnje verzije projekta snimljene u repozitorijumu, se dobija sa naredbom:

|  |  |
| --- | --- |
| 1 | git diff |

**Razlika izmedju komita na istoj grani**

Na sledećem primeru dobijamo razliku izmedju prethodnog komita i komita tri koraka unazad.

|  |  |
| --- | --- |
| 1 | git diff HEAD~3 HEAD~1 |

**Razlika izmedju komita na dve grane**

Kada želimo ispis svih razlika u projektu izmedju dva odredjena snimka stanja na različitim granama, koristimo naredbu sa argumentima nazivi grana:

|  |  |
| --- | --- |
| 1 | git diff master sekundarna\_grana |

Na prethodnom primeru se naredbom dobija razlika izmedju master grane i sekundarne\_grane. Obratite pažnju da je **redosled** pri pisanju grana **bitan**. U slučaju nepravilnog redosleda desilo bi se da neki fajl koji je dodan u sekundarnoj grani bude prikazan kao da je obrisan.

**reflog**

Sledeća naredba je veoma važna u radu sa promenom istorije, jer nam daje informacije o radnjama koje su vršene sa git-om. Naredba izlistava sve promene praćene gitom kroz sve grane:

|  |  |
| --- | --- |
| 1 | git reflog |

ili još detaljnija verzija listinga sa datumom i vremenom:

|  |  |
| --- | --- |
| 1 | git reflog --relative-date |

OBJAŠNJENJE:   
Ova naredba nam omogućava da vidimo SVE promene pa čak i one koje se više ne vide zbog prepravljanja istorije. Sa ovom naredbom dobijamo jasno opisan listing svih komita sa odgovarajućim brojem pozicije HEAD (“HEAD@{broj}”), pa čak i onih koji se više ne vide u log-u.

Pored ove naredbe za listanje stanja možemo da koristimo:

|  |  |
| --- | --- |
| 1 | git log --graph --decorate --oneline |

**Pregled komita u istoj grani (checkout)**

Sa naredbom **checkout** možemo se prebaciti (“teleportovati”) na drugu granu ili na stariji commit u istoj grani. Takvo stanje se zove **“detached HEAD”** tj. stanje kada HEAD pointer pokazuje na neki commit koji nije poslednji u grani.

**Primer**

Ukoliko je potrebno da se vratimo za 10 komita (koraka) unazad koristimo naredbu:

|  |  |
| --- | --- |
| 1 | git checkout HEAD~10 |

Nakon prebacivanjana na drugi commit, u radnom direktorijumu vidimo fajlove u verziji za taj commit. Ukoliko pravimo izmene na tim fajlovima, da bi ih sačuvali potrebno je da napravimo novu granu i da *commit-ujemo* promene. Nakon ovoga će HEAD pointer pokazivati na commit na kraj nove grane, pa repozitorijum više neće biti u *“detached HEAD”* stanju.

**Pretraga**

**Pretraga komentara**

Ukoliko vršimo pretragu komentara pri commit-ovanju koristimo:

|  |  |
| --- | --- |
| 1 | git log --grep="reč po kojoj se pretražuje" |

**Pretraga reči unutar fajlova**

Ukoliko vršimo pretragu neke reči (string) unutar fajlova koristimo naredbu:

|  |  |
| --- | --- |
| 1 | git log -Sreč\_po\_kojoj\_se\_oretrazuje |

Ukoliko rečenica koja se pretražuje ima razmake potrebno ju je staviti izmedju navodnika.

|  |  |
| --- | --- |
| 1 | git log -S"recenica po kojoj se pretražuje" |

**Primer**

Na sledećem primeru se traži tekst **“strava program”**:

|  |  |
| --- | --- |
| 1 | git log -S"strava program" |

Pretraga nam **nalazi commit** u kome se nalazi traženi fajl u obliku **76cf802d23834bc74473370ca81993c5b07c2e35**. Naziv komita čine prvi pet cifara **76cf8**. Dati commit možemo pregledati sa naredbom:

|  |  |
| --- | --- |
| 1 | gitk 76cf8 |

**Čuvanje izmena**

**Slanje promena u pripremni prostor**

Da bi poslali snapshot promenjene verzije nekog fajla (npr. *proba.txt*), u repozitorijum, moramo prvo da pošaljemo snapshot u pripremni prostor (tzv. index). Tek fajlovi koji se nalaze u ovom prostoru se mogu commit-ovati tj. poslati u repozitorijum.

**Slanje odredjenog fajla**

Slanje samo odredjenog fajla u pripremni prostor sa naredbom:

|  |  |
| --- | --- |
| 1 | git add proba.txt |

**Slanje odredjenog direktorijuma**

Sledeća naredba šalje odredjeni direktorijum u pripremi prostor:

|  |  |
| --- | --- |
| 1 | git add naziv\_direktorijuma/\* |

**Slanje snapshot-a svih promenjenih fajlova odredjene ekstenzije iz root-a radnog foldera**

Sledeća naredba će prebaciti u pripremni direktorijum sve fajlove koje nadje u glavnom radnom direktorijumu sa .txt ekstenzijom, ali bez fajlova iz podfoldera.

|  |  |
| --- | --- |
| 1 | git add \*.txt |

**Slanje snapshot-a svih promenjenih fajlova odredjene ekstenzije uključujući i iz podfoldera**

Ukoliko treba da se nadju svi fajlovi sa istom ekstenzijom uključujući fajlove u podfoderima onda se koristi naredba sa **navodnicima**:

|  |  |
| --- | --- |
| 1 | git add '\*.txt' |

**Slanje svih fajlova**

Sledeća naredba će napraviti snapshot svih promenjenih fajlove iz radnog direktorijuma u pripremni direktorijum.

|  |  |
| --- | --- |
| 1 | git add . |

Obratiti pažnju kod ove naredbe **postoji razmak** (space) izmedju add i tačke!

**Commit-ovanje**

Sa *commit-ovanjem* mi **pravimo snimak promena stanja** fajlova, koje smo prethodno izabrali i prebacili u pripremni prostor.

|  |  |
| --- | --- |
| 1 | git commit -m "obavezan komentar faze" |

**Poništavanje promena**

**Poništavanje nekomitovanih promena**

**a) Pražnjenje *“staging area”* (index-a)**

Ukoliko smo greškom poslali fajlove na *“staging area”* tj. *“index”*, uklanjanje svih fajlova iz *“staging area”* je sa naredbom:

|  |  |
| --- | --- |
| 1 | git reset HEAD . |

**b) Povratak stanja fajla sa poslednjeg komita**

Ukoliko još nismo commit-ovali izmene napravljen u odredjenom fajlu, a želimo da vratimo verziju koja je sačuvana poslednim komitom, dovoljno je se prebaciti na taj fajl sa *“checkout”* naredbom i promene će biti “zaboravljene” jer ih nismo komitovali.

Ako smo greškom izmenili fajl ali **nismo ga još commit-ovali**, možemo da vratimo verziju tog fajla sa poslednjeg komita naredbom:

|  |  |
| --- | --- |
| 1 | git checkout -- fajl.extenzija |

NAPOMENA:   
U slučaju da se fajl nalazi unutar nekog foldera, moramo podesiti da u komandnoj liniji putanja pokazuje na taj folder, inače git neće naći traženi fajl.

**Primer**

Za vraćanje fajla *proba.txt* koji se nalazi u folderu “probni\_folder”, potrebno je da postavimo putnju u taj folder:

|  |  |
| --- | --- |
| 1  2 | cd probni\_folder  git checkout -- proba.txt |

**c) Povratak stanje svih fajlova od poslednjeg komita**

Povratak stanja svih fajlova, koji još nisu komitovani, na stanje snimljeno sa poslefnim komitom, vraćamo sa naredbom:

|  |  |
| --- | --- |
| 1 | git checkout -- . |

Ova naredba poništava sve ne komitovane promene.

OBJAŠNJENJE:   
Oznaka -- znači da se sve iza nje tretira kao filename (čak i da postoji neka naredba, neće se izvršiti nego će se tretirati kao filename).

**Izmena zadnjeg komita (–amend)**

**Izmena samo poruke poslednjeg komita**

Ukoliko želimo da delimično promenimo poruku iz poslednjeg komita dovoljno je da koristimo naredbu “commit” uz “flag” --amend:

|  |  |
| --- | --- |
| 1 | git commit --amend |

Nakon ove naredbe se otvara tekstualni fajl na čijem vrhu se nalazi poruka iz poslednjeg komita, za promenu poruke je dovoljno u tom fajlu editovati poruku a zatim sačuvati i zatvoriti fajl, nakon čega će biti “pregažen” poslednji commit sa istim snimkom stanja ali sa drugačijom porukom.

Ako želimo da promenimo celu poruku bez editovanja stare poruke dovoljno je koristi sledeću kombinaciju naredbi i flag-ova:

|  |  |
| --- | --- |
| 1 | git commit --amend -m "Totalno nova poruka" |

**Izmena sadržaja poslednjeg komita**

Ako imamo sitnije izmene koje bi smo želeli da su bile deo prethodnog commita, potrebno je željene dodatne imene ubaciti na “stage” (index):

|  |  |
| --- | --- |
| 1 | git add fajl-koji-je-naknadno-promenjen |

a zatim koristiti “commit” naredbu sa flag-om: --amend:

|  |  |
| --- | --- |
| 1 | git commit --amend |

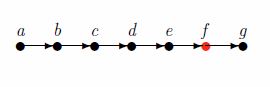
Nakon ove naredbe se takodje otvara tekstualni fajl na čijem vrhu se nalazi poruka iz poslednjeg komita, za promenu poruke je dovoljno u tom fajlu editovati poruku a zatim sačuvati i zatvoriti fajl.

NAPOMENA:   
Ukoliko pri izmeni poslednjeg komita ne želimo da menjamo poruku dovoljno je da dodamo flag --no-edit. Ako koristimo ovaj flag neće se otvarati fajl za editovanje poruke, već će se odmah sačuvati promene.

|  |  |
| --- | --- |
| 1 | git commit --amend --no-edit |

**Revert**

Naredba **“revert”** se koristi za poništavnje komitovanih promena. Ovaj pristup je sigurniji od pristupa gde se koristi naredba *reset*, jer ne briše commit, nego dodaje novi commit tj. čvor u kome su izmene napravljene u nekom od prethodnih commit-a. Sa ovom naredbom se “vraćamo na stanje iz nekog starog komita” (pri čemu i dalje postoji način da povratimo izmene koje smo upravo pregazili).



Dakle ukoliko je poslednji *commit* u čvoru “f” i želimo da izbacimo promene nastale u njemu, koristimo naredbu *revert* koja ga neće obrisati iz istorije *komita*, nego će napraviti novi *commit* u čvoru “g” koji će biti isti kao u čvoru “e”.

|  |  |
| --- | --- |
| 1 | git revert HEAD~0 |

ili jednostavnije:

|  |  |
| --- | --- |
| 1 | git revert HEAD |

Ukoliko želimo da se vratimo par komita unazad, prvo je potrebno sa naredbom **reflog** da pronadjemo broj HEAD-a komita na koji želimo da se vratimo:

|  |  |
| --- | --- |
| 1  2 | git reflog  git revert HEAD~3 |

OBJAŠNJENJE:   
Ova naredba je sigurna jer ponovnim revert-om (tzv. Re-Revert-om) možemo da se vratimo na prvobitno stanje.

**Resetovanje**

I naredba **“reset”** se koristi za poništavnje komitovanih promena, stim što ima tri moguća podtipa:

**a) Soft reset**

|  |  |
| --- | --- |
| 1 | git reset --soft HEAD~brojKomitaUnazad |

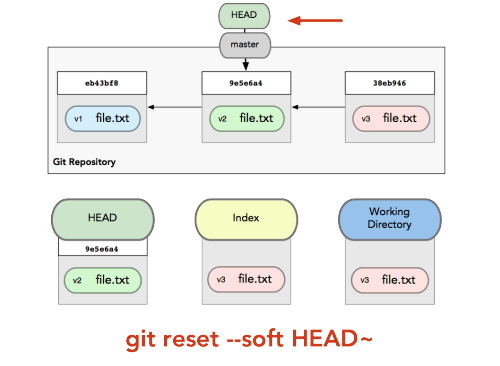
Sa naredbom **“reset”** i flag-om --soft brišemo odgovarajuće komite sve do komita odredjenog naredbom HEAD~broj. U tom trenutku su fajlovi sa svim izmenama spremi za naredno komitovanje (nalaze se na “stage”). Praktično bi smo se sa komitovanjem u to trenutku vratili na stanje “brojKomitaUnazad+1”.

**Primer**

Sa naredbom u ovom primeru, brišemo poslednji komit a HEAD ostaje ns predposlednjem komitu (HEAD~1), a izmene iz poslednjeg komita se nalaze na stage (takodje su iste i u working direktorijumu)

|  |  |
| --- | --- |
| 1 | git reset --soft HEAD~1 |

Ukoliko bi smo odmah uradili komit svih fajlova na “stage” vratili bi se na početak.



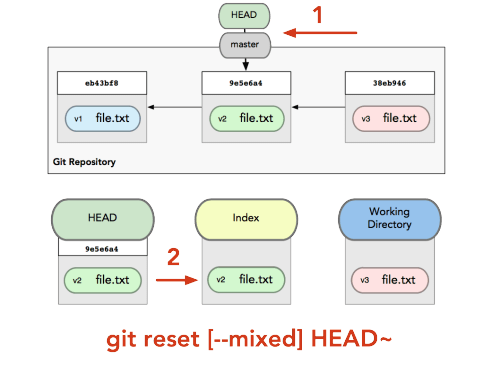
**b) Mixed reset**

Sa naredbom **“reset”** i flag-om --mix takodje brišemo odgovarajuće komite sve do komita odredjenog naredbom HEAD~broj. Stim što se u tom trenutku fajlovi sa izmenama (za sledeći komit) nalaze u radnom direktorijumu.

**Primer**

U ovome primeru brišemo poslednji komit (glava se vraća na komit dalje tj. na predposledni), ali sve izmene su nam sada u radnom repozitorijumu (i dalje nije ništa izgubljeno).

|  |  |
| --- | --- |
| 1 | git reset --mixed HEAD~1 |



Ukoliko bi smo odmah bez nekih dodatnih izmena poslali fajlove na “stage”, a zatim uradili komit vratili bi se na početak.

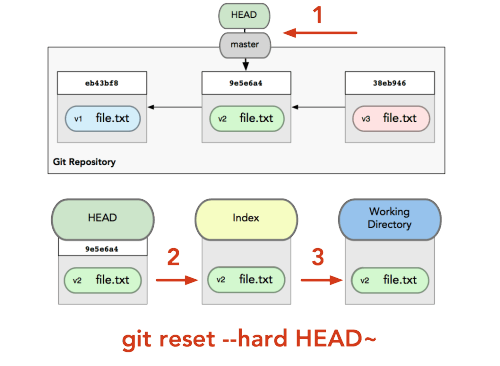
**c) Hard reset**

Sa naredbom **“reset”** i flag-om --hard brišemo odgovarajuće komite sve do komita odredjenog naredbom HEAD~broj, dok je stanje fajlova je isto kao da smo upravo izvršili komit.

**Primer**

U ovome primeru brišemo poslednji komit (glava se vraća na komit dalje tj. na predposledni), trenutno nema promena za komitovanje i sve promene iz prošlog komita su nestale.

|  |  |
| --- | --- |
| 1 | git reset --hard HEAD~1 |



NAPOMENA:   
Umesto korišćenja HEAD~1 uvek možemo da koristimo tačan naziv komita koji dobijamo sa **“reflog”** kao npr.   
git reset --hard f3cb6e2

**Primer**

Ovaj primer prikazuje nepohodne akcije ako smo slučajno poslali izmene na master a trebali smo ih poslati na novu granu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | # kreiramo novu granu sa trenutnim stanjem master-a  git branch nova\_grana    # Ukidanje tog poslednjeg pogrešnog komita na master grani  git reset HEAD~ --hard    # Prebacivanje na novu granu koja i dalje sadrži onaj problematični komit  git checkout nova\_grana |

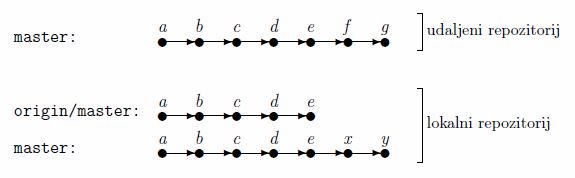
**Primer**

U ovome primeru se vidi redosled akcija koje su potrebne u slučaju da smo komitovali na pogrešnu granu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # undo poslednjeg komita ali ostavljamo izmene u fajlovima  git reset HEAD~ --soft  git stash    # prebacivanje na korektnu granu  git checkout name-of-the-correct-branch  git stash pop    # Dodavanje izmena i njihovo komitovanje na pravu granu  git add . # or add individual files  git commit -m "your message here"; |

**Ažuriranje lokalne “origin/master” grane (fetch)**

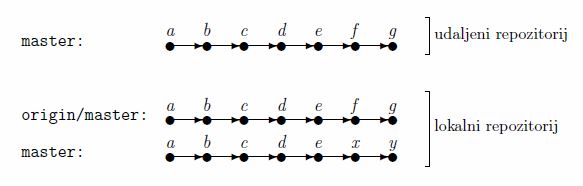
Dok mi radimo u lokalu, paralelno rade i naše kolege u svojim lokalnim repozitorijumima. Ukoliko push-uju svoje promene na udaljeni repoitorijum, pojaviće se razlika izmedju master grane na udaljenom repozitorijumu (na slici *“master”*) i njegove lokalne kopije (na slici *origin/master*).



Na slici gore možemo da primetimo da naša lokalna kopija udaljene grane **“origin/master”** ima dva komita manje. Kada je situacija na projektu slična situaciji na prethodnoj slici, ažuriranje *origin/master* grane se vrši naredbom:

|  |  |
| --- | --- |
| 1 | git fetch |

Nakon ove naredbe novo stanje lokalne grane (“origin/master”) je ažurirano, i grana je sada identična kopija “master” sa udaljenog repozitorijuma:



**Spajanje promena iz “origin/master” u lokalnu “master” granu (merge)**

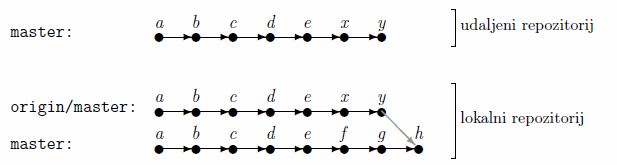
Sada kada imamo ažuriranu lokalnu verziju udaljenog repozitorijuma možemo preuzeti te promene i u naš radni repozitorijum u okviru grane “master” (na slici donja grana master). Za ovu potrebu koristimo naredbu:

|  |  |
| --- | --- |
| 1 | git merge origin/master |

Git će se čak snaći i ako ne napišemo šta hoćemo da merge-ujemo:

|  |  |
| --- | --- |
| 1 | git merge |

nakon čega grafik izgleda ovako:



**Pull**

Zbog čestog zajedničkog korišćenja naredbi **fetch** a zatim i **merge** se pojavila potreba da se smanji kod i sve obuhvati samo jednom naredbom:

|  |  |
| --- | --- |
| 1 | git pull |

Koja je zamena za obe:

|  |  |
| --- | --- |
| 1  2 | git fetch  git merge origin/master |

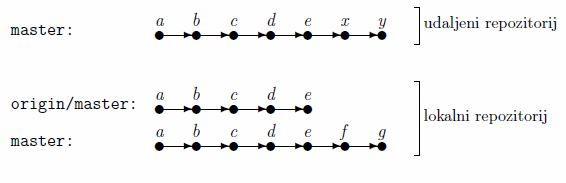
**Slanje izmena na udaljeni repozitorijum**

Običnim commit-ovanjem se promene ne šalju u udaljeni repozitorijum kao kad je slučaj sa lokalnim repozitorijumom, nego se koristi posebna naredba za to **“push”**. Praksa je da u lokalu napravimo više komita, pa tek onda kada završimo neku celinu pošaljemo promene na udaljeni repozitorijum. Za ovu radnju postoji dve opcije:

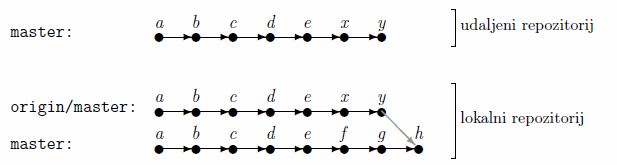
1. **push** – kada imamo ovlašćenja
2. **pull request** – kada nemamo ovlašćenja

**Push**

Kada na jednom projektu radi više saradnika na različitim problemima situacija tokom vremena postaje komplikovanija. U slučaju da vlasnik repozitorijuma radi na svom repozitorijumu i commit-ovao je svoja dva komita **x i y**, dok istovremeno saradnik radi u lokalu na drugom problemu i commitovao je **e** i **f** čvorove, dijagram grana izgleda ovako:



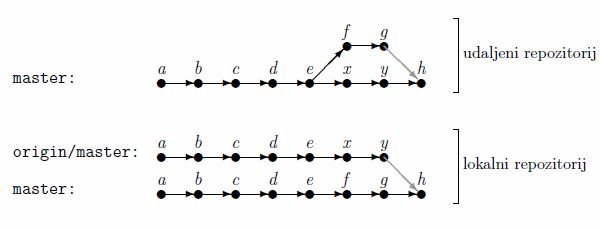
Ukoliko koristimo kao u prethodnom primeru naredbu **push**, git je **neće prihvatiti**, jer moramo prethodno da *“sredimo”* naš lokalni repozitorijum i ažuriramo našu lokalnu *origin/master* granu sa naredbom **git pull**.



Tek nakon “sredjivanja” lokalnog repozitorijuma i ažuriranja lokalne grane “master” možemo poslati naše commit-e na udaljeni repozitorijum i ažurirati udaljenu granu “master” sa naredbom:

|  |  |
| --- | --- |
| 1 | git push origin master |

Nokon ovoga dobijamo definitivni grafik kao na slici ispod:



**Pull request**

Pull request nije ništa drugo nego kratka poruka vlasniku nekog udaljenog repozitorijuma koja sadrži adresu našeg repozitorijuma, opis izmena koje smo napravili i predlog da on te izmene preuzme u svoj repozitorijum.

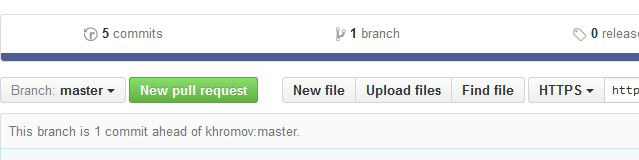
Uz email vlasniku repozitorijuma s porukom, koristimo i sledeću naredba:

|  |  |
| --- | --- |
| 1 | git request-pull |

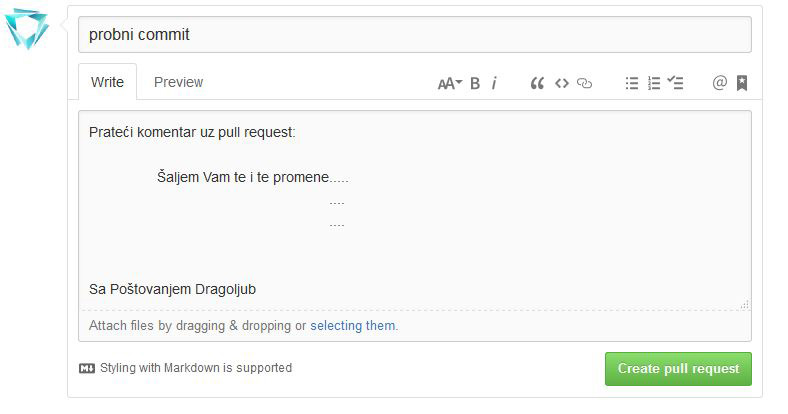
koja priprema sve detalje izmena koje ste napravili za tu poruku.

**GitHub Pull request**

Ukoliko koristite Github za svoje projekte tamo postoji više potpuno automatizovanih načina da se pošalju promene iz našeg repozitorijuma na GitHub-u na repozitorijum sa koga smo *“fork-ovali”*. Najjednostavniji način je direktno preko dugmeta **New pull request**.



Nakon izbora grane sa koje želimo da pošaljemo promene i aktiviranja dugmeta se otvara nova stranica **Comparing changes** gde se pregledno vidi šta su promene i na kojim fajlovima. Kada nakon pregleda utvrdite da su to te promene koje želite da pošaljete, izberete dugme **Create pull request**. Otvara se nova stranica gde treba da upišemo naslov i opcioni prateći komentar, nakon čega definitivno šaljemo naše promene sa aktiviranjem dugmeta **Create pull request**.



**Osnove grananja Git-a**

**Pravljenje nove grane**

Pravljenje grane se vrši sa naredbom:

|  |  |
| --- | --- |
| 1 | git branch naziv\_nove\_grane |

**Pregled grana**

Da bi smo se uverili da smo napravili novu granu ćemo proveriti sa naredbom *“branch”* koja pravi spisak svih grana u repozitorijumu:

|  |  |
| --- | --- |
| 1 | git branch |

Pri izlistavanju u code-u sa **\* (zvezdicom)** je obeležena grana u kojoj se trenutno nalazimo.

**Objavljivanje “svetu” da postoji nova grana**

Nova grana je napravljena ali se trenutno nalazi samo kod nas u lokalu, da bi obelodanili postojanje grane i na udaljenom repoztorijum potrebno je da pušujemo granu:

|  |  |
| --- | --- |
| 1 | git push origin ime-grane |

**NAPOMENA:**   
Iako smo poslali sve izmene da udaljeni repozitorijum, kolege ne mogu da se prebace na ovu granu sve dok ne povuku sve izmene na svoj lokal.

|  |  |
| --- | --- |
| 1 | git fetch |

**Prebacivanje na novu granu**

Iako je sada nova grana napravljena, mi smo trenutno i dalje na istoj grani na kojoj smo i bili (obično master), tako da je potrebno da se prebacimo na novu granu:

|  |  |
| --- | --- |
| 1 | git checkout naziv\_grane\_na\_koju\_se\_prebacujemo |

Nakon samog kreiranja grane a zatim i prebacivanja na nju, naš radni direktorijum će biti kopija grane na kojoj smo bili kada smo napravili novu granu.

Medjutim postoji i jednostavniji nači koji obuhvata prethodne dve naredbe:

|  |  |
| --- | --- |
| 1  2 | git branch naziv\_nove\_grane  git checkout naziv\_nove\_grane |

A to je:

|  |  |
| --- | --- |
| 1 | git checkout -b naziv\_nove\_grane |

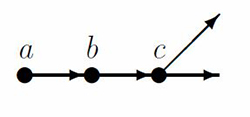
Sa ovim fleg-om **-b** smo u kreirali i prebacili se na novu granu.

**Preimenovanje grane**

Ukoliko se pojavi potreba da preimenujemo granu **u kojoj se nalazimo** to možemo uraditi sa naredbom:

|  |  |
| --- | --- |
| 1 | git branch -m novi\_naziv\_grane |

**Čuvanje izmena na novoj grani**



Kada se napravi grana u njoj nema ni jednog čvora, novi čvorovi nastaju tek kad napravimo neke izmene i komitujemo ih, što radimo na standardni način:

|  |  |
| --- | --- |
| 1  2 | git add .  git commit -m "komentar uz komit" |

**Slanje izmena na novu granu**

Izmene na ovoj grani sa lokala šaljemo na udaljeni repozitorijum na isti nači kao i na master grani stim što umesto naziva grane **koristimo naziv sporedne\_grane**:

|  |  |
| --- | --- |
| 1 | git push origin naziv\_grane |

**Spajanje grana “merge”**

Nakon napravljenih izmena u sporednoj grani postupak slanja tih izmena u glavnu granu se sastoji iz sledećih koraka:

* **a) ažuriranje sporedne grane sa tudjim izmenama na sporednoj grani**
* **b) ažuriranje sporedne grane sa izmenama iz master grane**
* **c) Slanje merge-ovanih izmena sporedne grane na udaljeni repo**
* **d) Prebacivanje na master granu**
* **e) Kontrola ažuriranosti master grane**
* **f) Preuzimanje podataka iz sporedne u master granu**
* **g) Rešavanje konflikta pri spajanju grana**
* **h) Slanje merge-ovanih izmena sa mastera na udaljeni repo**

**a) ažuriranje sporedne grane sa tudjim izmenama na sporednoj grani**

Pre svega je neophodno da proverimo da neko u medjuvremnu nije pušovao izmene na sporednoj grani:

|  |  |
| --- | --- |
| 1 | git pull origin sporedna\_grana |

**b) ažuriranje sporedne grane sa izmenama iz master grane**

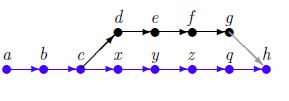
Ovaj deo je bitan jer postoji mogućnost da su u medjuvremenu napravljene neke izmene na master grani .

|  |  |
| --- | --- |
| 1 | git merge master |

NAPOMENA:   
Obratite pažnju da ovde nije u pitanju “origin master” (udaljena verzija mastera) već naša lokalna verzija master-a (bez origin dela)!

Iako se “merge branch” prevodi kao **“spajanje grana”** to nije ispravan smisao ove naredbe, jer standardno značenje rečenice “spajanje grane” bi napravilo samo jednu granu, što ovde nije slučaj jer grane nastavljaju svoj život odvojeno. Jedino što se od tog trenutka **sve nove izmene nastale u jednoj grani preuzimaju u drugu granu**.

**c) Slanje merge-ovanih izmena sporedne grane na udaljeni repo**



Nakon izvršenog pripajanja bez konfilikta nije potrebno raditi **commit** jer je sama naredba napravila novi čvor **“h”**, tako da je samo dovoljno da pušujemo te izmene na udaljeni repozitorijum sa naredbom:

|  |  |
| --- | --- |
| 1 | git push origin sporedna\_grana |

Medjutim ukoliko ima konflikta problematični fajl se neće *commit-ovati*, već je ostavljeno da korisnik sam edituje fajl i izabere verziju koja mu odgovara. Nakon toga prvo je potrebno napraviti commit a tek onda push:

|  |  |
| --- | --- |
| 1  2 | git commit -m "komentar komita"  git push origin sporedna\_grana |

**d) Prebacivanje na master granu**

Za preuzimanje izmena iz druge grane potrebno je **biti u grani kojoj želimo dodati izmene iz druge grane**. Stoga se moramo prebaciti na master granu:

|  |  |
| --- | --- |
| 1 | git checkout master |

**e) Kontrola ažuriranosti master grane**

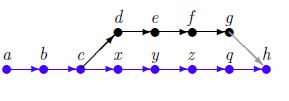
Čim smo se prebacili na master granu potrebno je ponovo proveriti da neko u medjuvremenu nije napravio dodatne izmene:

|  |  |
| --- | --- |
| 1 | git pull origin master |

**f) Preuzimanje podataka iz sporedne u master granu**

I tek sada možemo da koristimo naredbu koja će da prebaci sve promene iz sporedne grane u master granu:

|  |  |
| --- | --- |
| 1 | git merge naziv\_grane\_iz\_koje\_preuzimamo. |



Nakon izvršenog pripajanja nije potrebno raditi **commit** jer je sama naredba napravila novi čvor **“h”**. Za razliku od ostalih čvorova (komita) samo ovaj ima dva “roditelja” tj. dve strelice vode do njega.

Uz sledeće slučajeve ćemo bolje shvatiti kako funkcioniše naredba:

| **Slučaj** | **Rezultat naredbe** |
| --- | --- |
| Fajl je izmenjen u pripojenoj grani a u glavnoj nije | Dodaće se izmene u fajlu |
| Fajl je dodat u pripojenoj grani | Biće dodat fajl |
| Fajl je obrisan u pripojenoj grani a u glavnoj nije | Fajl će biti obrisan |
| U pripojenoj grani smo izmenili i preimenovali fajl, a u glavnoj smo samo izmenili fajl | Ako izmene na kodu nisu bile konfliktne, onda će se u glavnoj fajl *preimenovati* i *sadržaće izmene iz obe grane*. |
| U pripojenoj grani smo obrisali fajl, a u glavnoj smo ga samo izmenili. | KONFLIKT |

**g) Rešavanje konflikta pri spajanju grana**

U slučaju konflikta problematični fajl se neće *commit-ovati*, već je ostavljeno da korisnik sam edituje fajl i izabere verziju koja mu odgovara. Kada se javi konflikt ako stratujemo naredbu **git status** dobićemo obeležen deo problematičnog koda na sledeći način:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <<<<<<<< HEAD  ...  ...  ========  ...  ...  >>>>>>>> new\_branch |

Deo kod između **<<<<<<<< HEAD** i **========** sadrži onaj kod koji je prisutan u osnovnoj grani tj. grani na koju šaljemo izmene, dok su redovi koda između **========** i **>>>>>>>> new\_branch** prisutni u grani sa koje šaljeno izmene “new\_branch”. Na programeru je odluka koji deo će zadržati a koji obrisati (ili eventualno napraviti kombinaciju oba).

Postoje više programa (tzv. *merge tool)* koji se koriste za vizuelni prikaz konflikta i lakše rešavanje konflikata, pročitajte više o tome u članku: [“My favorite tools to resolve git merge conflicts”](https://blog.xoxzo.com/en/2019/03/29/my-favorite-tools-to-resolve-git-merge-conflicts/).

**h) Slanje merge-ovanih izmena sa mastera na udaljeni repo**

I na kraju se ceo krug završava slanjem ovih merge-ovanih izmena na masteru na njegov repo. Naglašavam da nije potrebno raditi commit jer je čvor automatski dodat, pa je dovoljno samo pušovati taj automatsko generisani commit:

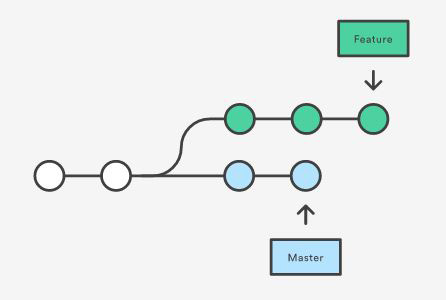
|  |  |
| --- | --- |
| 1 | git push origin master |

**Spajanje grana “Rebase”**

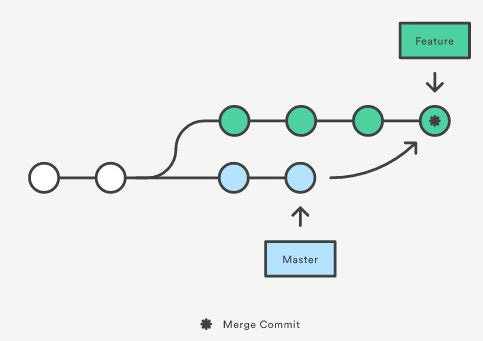
Naredba je slična naredbi *merge*, ali ih ne spaja u jedan novi krajnji čvor (kao merge commit), nego dodaje celu istoriju svih komita. Najlakše se shvata preko primera.

**Primer**

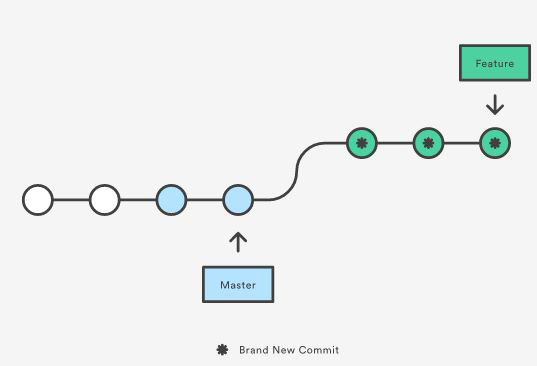
Na sledaćoj slici je prikazno stanje projekta, gde pored *master grane* postoji i sporedna grana, a u okviru nje par commit čvorova. Zadatak je da spojimo podatke iz master grane (plavi krugovi) u **sporednu**.



Ukoliko koristimo naredbu *merge*, napravili bi smo novi commit (zeleni sa zvezdom) kojim bi se dodale sve promene iz master grane, pa bi dijagram izgledao kao na sledećoj slici:



A ako to isto uradimo sa naredbom *rebase*, dijagram bi izgledao ovako:



Najveća prednost je jasnija istorija projekta čiji dijagram je linearan kao i eliminisanje dodatnih čvorova koji se pravi pri korišćenju *merge* naredbe.

Ova naredba se najčešće koristi za ažuriranje *origin/master* grane koja je lokalna kopija master grane udaljenog repozitorijuma.

**Spajanje grana “Cherry-pick”**

Cherry-pick je **specijalna vrsta spajanje dve grane**, kada želimo da prebacimo **samo jednan *commit*** iz jedne grane u drugu. Ne želimo da koristimo **merge** jer bi prebacio sve izmene koje smo napravili u celoj grani.

**Postupak**

Za prebacivanje poslednjeg *komita* sa sporedne grane u *master* granu je postupak sledeći:

1. **git log sporedna\_grana** – proučimo istoriju sporedne grane
2. izaberemo jedinstveni broj komita
3. **git cherry-pick jedinstveni\_broj\_komita** – naredba koja aktivira *cherry-pick*

U slučaju konflikta postupak je isti kao kada se javi konflikt pri korišćenju naredbe *marge*.

**Brisanje grane**

**Brisanje grane u lokalu**

Brisanje grane može napraviti neželjene posledice i gubitak koda, stoga je potrebno pre brisanja proveriti da li su uradjeni svi neophodni koraci. Brisanje lokalne grane (ukoliko je već sinhronizovana sa njenim “origin-om” tj. granom na udaljenom serveru) nije toliko “opasno” jer uvek možemo da je “povučemo” ponovo sa servera. Da bi obrisali neku granu ne smemo biti na njoj te je potrebno da se “checkout”-ujemo na neku drugu granu (npr. na master):

|  |  |
| --- | --- |
| 1 | git checkout master |

Tek nakon toga možemo da koristimo naredbu:

|  |  |
| --- | --- |
| 1 | git branch -d localBranchName |

Napomena:   
Opcija **-d** će izbrisati granu samo ako je već push-ovana i spojena sa udaljenom granom. Ako želite da prinudno izbrišete granu, čak i ako još uvek nije push-ovana ili spojena koristite **-D**.

**Brisanje remotely grane**

Udaljeni repozitorijum se obično zove **“origin”**, te za brisanje grane na udaljnenom repozitorijumu koristimo sledeću naredbu:

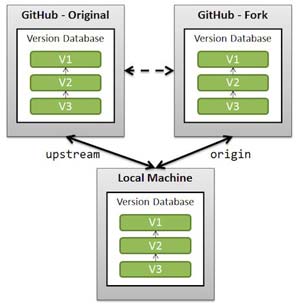
|  |  |
| --- | --- |
| 1 | git push origin --delete remoteBranchName |

**Forkovanje**

Naredba **fork** ne postoji u okviru git-a, već je vezana za servis **GitHub**. Ova naredba nam omogućava da neki postojeći projekat drugog korisnika GitHub-a kopiramo u naš Github nalog, stim da naša kopija bude *“svesna”* da postoji roditeljski repozitorijum.

Nakon forkovanja projekta na naš gitHub nalog potrebno je da se napravi lokalna verzija repozitorijuma kloniranjem. Ovaj lokalni repozitorijum je u vezi sa oba repozitorijuma na GitHub-u. U komandnoj liniji namestimo putanju na naš lokalni repozitorijum a naredba kojom lokalnom repozitorijumu dajemo do znanja da postoji početni repozitorijum na osnovu koga je forkovan repozitorijum na naš giohub nalog je sledeća:

|  |  |
| --- | --- |
| 1 | git remote add upstream https://github.com/ime\_korisnika/naziv\_repozitorijuma.git |



Nakon naredbe **> git remote -v** povezanost našeg lokalnog repa sa udaljenim *bare repozitorijumima* bi izgledao ovako:

|  |  |
| --- | --- |
| 1  2  3  4 | ime\_fork\_roditelja https://github.com/ime\_fork\_roditelja/naziv\_repa (fetch)  ime\_fork\_roditelja https://github.com/ime\_fork\_roditelja/naziv\_repa (push)  origin https://github.com/ime\_korisnika/naziv\_repa (fetch)  origin https://github.com/ime\_korisnika/naziv\_repa (push) |

Gledano sa lokala, roditeljski repozitorijum kod standardnog kloniranja se zove *“origin”*, a originalni GitHub repozitorijum koji je *forkovan* se naziva **“upstream”**

Veza izmedju lokalnog repozitorijuma i *upstream* repozitorijuma je bitna jer preko nje ažuriramo lokalnu granu ***upstream/master*** koja je kopija *upstream* grane sa naredbom:

|  |  |
| --- | --- |
| 1 | git fetch upstream |

nakon čega radimo spajanje promena sa naredbom:

|  |  |
| --- | --- |
| 1 | git merge upstream/master |

ili ako *upstrem* ima jednostavan set komita

|  |  |
| --- | --- |
| 1 | git rebase upstream/master |

Po završetku rada na fajlovima u lokalu, možemo standardno poslati sve promene na naš udaljeni repozitorijum *origin* sa naredbom *push*, nakon čega možemo na više načina da pošaljemo na *upstream* **pull request**.

Po završetku rada na fajlovima u lokalu, možemo standardno poslati sve promene na naš udaljeni repozitorijum *origin* sa naredbom *push*, nakon čega možemo na više načina da pošaljemo na *upstream* **pull request**, ali više o ovome na delu [GitHub Pull request](https://www.webprogramiranje.org/git-komande/#github_pullrequest).

**(BONUS) Kreiranje Git aliasa**

Prilikom korišćenja git-a u terminalu uočava se potreba da se skrate neke često korišćene naredbe, tj. da se kreiraju aliasi. Kreiranje alijasa je omogućeno na sledeći način:

**Primer**

|  |  |
| --- | --- |
| 1 | git config --global alias.st status |

U prethodnom primeru smo napravili skraćeni oblik za git naredbu **status** i od sada je možemo koristi sa istim učinkom kao:

|  |  |
| --- | --- |
| 1 | git st |

Ukoliko se naredba sastoji iz više reči onda je na Windows-u potrebno sve staviti izmeću dvostrukih navodnika **” “**, dok je na Unix-u dovoljno koristiti jednostruke navodnike.

**Primer**

|  |  |
| --- | --- |
| 1 | git config --globa alias.rso "remote show origin" |

Nakon čega možemo da pozivamo naredbu “remote show origin” skraćenicom:

|  |  |
| --- | --- |
| 1 | git rso |
|  |  |