Git

Autori:
Goran Savić
Milan Segedinac

1. Sistemi za kontrolu verzija

U realnim softverskim projektima važan aspekt upravljanja projektom je upravljanje verzijama projekta. U toku razvoja projekta, programski kod se menja. Korisno je evidentirati sve bitne izmene nad programskim kodom, tako da u svakom trenutku programer ima dostupne sve verzije koda ukoliko mu zatrebaju. Razlozi za ovakvu praksu su višestruki. Prvi bitan razlog je taj što je nekad potrebno iskoristiti određeni programski kod koji je ranije bio implementiran, a trenutno se ne koristi u aplikaciji. Slično, često se ispostavi da novoimplementirani kod ne zadovoljava zahteve i da je potrebno vratiti se na kod koji je ranije bio implementiran i za koji se ispostavilo da bolje funkcioniše. Drugi važan razlog je obezbeđivanje podrške za različite verzije programskog koda iste aplikacije u različitim situacijama. Na primer, određeni delovi aplikacije su različiti kada se aplikacija koristi interno u programerskom timu u odnosu na korišćenje iste aplikacije u produkcionom okruženju od strane klijenta.

Postoje različiti načini da se obezbedi upravljanje verzijama programskog koda u toku razvoja aplikacije. Često se stare ili nekorišćene verzije programskog koda ostavljaju zakomentarisane u kodu, kako bi se mogle iskoristiti po potrebi. Druga varijanta je da se vrši arhiviranje koda softverskog projekta pri svakoj bitnijoj izmeni, tako da uvek postoji i stara kopija fajlova koju je naknadno moguće iskoristiti. Iako ovakvi i slični pristupi mogu da delimično reše problem upravljanja verzijama, oni se standardno ne koriste jer ne obezbeđuju upravljanje verzijama na efikasan način. Ovo se odnosi kako na efikasnost skladištenja različitih verzija koda, tako i na efikasnost pronalaženja odgovarajuće verzije koda.

Iz tog razloga, za upravljanje verzijama se danas standardno koriste specijalizovane softverske aplikacije, koje se najčešće nazivaju sistemi za kontrolu verzija (eng. *version control systems*). Ove aplikacije omogućuju sistematično evidentiranje različitih verzija programskog koda uz automatizovanu podršku za preuzimanje željene verzije programskog koda i spajanje koda iz različitih verzija.

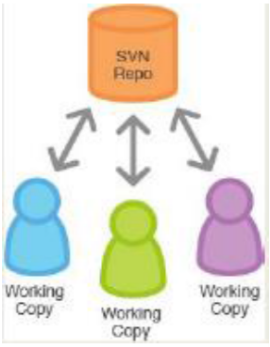
Evidencija i spajanje verzija programskog koda omogućuje automatizovanu podršku za još jedan aspekt razvoja softverskih projekata, a to je timski rad. Naime, standardan scenario u razvoju softverskih aplikacija je da više programera učestvuje u kreiranju aplikacije. Svi programeri koriste isti programski kod u kojem vrše svoje modifikacije. Izvršene modifikacije se moraju inkorporirati u programski kod tako da budu dostupne i ostalim članovima tima. Svaka modifikacija predstavlja novu verziju programskog koda. Verzija kreirana od strane člana tima se automatizovano spaja sa verzijama koju prave drugi članovi tima. Standardan tok razvoja aplikacije je takav da se svaka nova funkcionalnost evidentira kao nova verzija i odmah spaja sa ostatkom koda, kako bi svi članovi tima praktično svo vreme radili sa aktuelnom verzijom koda.

Sistemi za kontrolu verzija standardno organizuju kod u repozitorijume koda. Repozitorijum sadrži hijerarhiju foldera i fajlova čije se verzije evidentiraju. Određeni broj izmena u kodu (obično one izmene koje čine jednu zaokruženu logičku celinu) se evidentiraju kao nova verzija koda koja se postavlja na repozitorijum. Svaka verzija koja se sačuva na repozitorijumu, pored samih izmena u kodu, standardno evidentira i informacije o tome ko je napravio izmenu, kada je izmena izvršena, kao i opis izmene. Ovi podaci trebaju da pomognu u kasnijem pronalaženje određene verzije programskog koda. Sistem za kontrolu verzija pruža pristup istorijatu izmena koda na repozitorijumu, te je moguće pristupiti ranijim verzijama projekta i svim relevantnim podacima vezanim za izmenu.

Kada je reč o implementaciji upravljanja verzijama, postoje dva osnovna tipa sistema za kontrolu verzija:

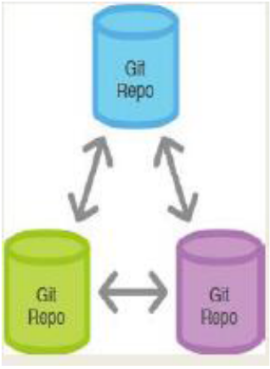
- centralizovani i
- distribuirani.

Arhitektura centralizovanih sistema za kontrolu verzija je prikazana na slici.



Kod ovakvih sistema postoji jedan centralizovani repozitorijum koda, koji skladišti aktuelnu verziju koda. Članovi tima preuzimaju određenu verziju koda sa repozitorijuma na svoj lokalni računar. Lokalni računari imaju samo preuzetu verziju koda, a sve ostale verzije su uskladištene na repozitorijumu. Kako se vrše lokalne izmene u kodu, tako se one postavljaju na repozitorijum. Ostali članovi tima mogu preuzeti ove izmene sa repozitorijuma. Najpopularniji predstavnik ovog tipa sistema za kontrolu verzija je Subversion (SVN).

Iako obezbeđuju relativno jednostavan rad, mana centralizovanih sistema za kontrolu verzija je to što postoji samo jedan centralizovan repozitorijum. Kod distribuiranih sistema za kontrolu verzija ideja je da svaki čvor koji učestvuje u razvoju projekta bude kompletan repozitorijum koji skladišti kompletnu istoriju izmena u kodu. Na ovaj način, verzije koda se evidentiraju bez obzira na to da li će i kada kod biti deljen. Kada postoji potreba, verzije koje sadrži jedan repozitorijum se mogu sinhronizovati sa izmenama na drugom repozitorijumu. Dakle, svaki čvor koji učestvuje u razvoju projekta se ponaša kao odvojen repozitorijum koji se može po potrebi sinhronizovati sa drugima. Ovakav pristup je ilustrovan na narednoj slici.



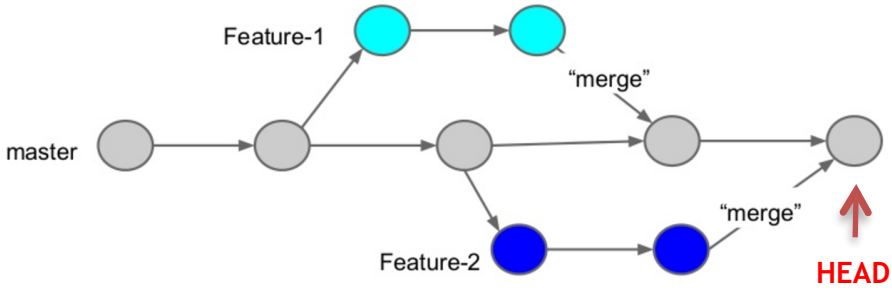
I pored ovakve arhitekture, ne postoji prepreka da se jednom repozitorijumu da uloga centralizovane tačke sinhronizacije kroz koju svi učesnici u projektu dobijaju izmene sa drugim repozitorijumima. Dakle, svi lokalni repozitorijumi mogu svoje izmene da šalju

jednom repozitorijumu i da sa njega preuzimaju sve izmene koje su izvršili drugi učesnici u projektu. Najpopularniji predstavnik ovog tipa repozitorijuma je Git. Iako je distribuiran sistem, standardno se Git koristi tako da se učesnici međusobno sinhronizuju kroz jedan svima dostupan repozitorijum.

2. Git

Git je distribuirani sistem za kontrolu verzija, koji je 2005. godine kreirao Linus Torvalds. Linus je inače autor Linux operativnog sistema i, obzirom da je Linux sistem otvorenog koda na kojem rade programeri širom sveta, Git je kreiran da bude sistem za upravljanje verzijama programskog koda Linuxa.

Git se zasniva na pojmu repozitorijuma koji predstavlja evidenciju hijerarhijski organizovanih foldera i fajlova. Repozitorijum sadrži istoriju verzija ovih podataka. Svaka verzija se naziva *commit,* prema istoimenoj operaciji kojom se vrši postavljanje verzije na repozitorijum. Verzije formiraju hronološki tok, u smislu da svakoj verziji prethode određene verzije. Jedna verzija može da bude osnov za nastanak više različitih verzija. Takođe, nova verzija može nastati kao rezultat spajanja dve verzije. Tako da verzije u repozitorijumu formiraju graf. Primer grafa Git verzija prikazan je na slici.



*preuzeto sa https://www.slideshare.net/MatthewKLiu/git-branch-management

Svaki krug na slici predstavlja jednu verziju (jedan *commit*). Vidimo da je moguće kreirati više grana u repozitorijumu tako da izmene projekta na jednoj grani ne utiču na ostale grane, što pruža podršku eksperimentisanju i nezavisnom razvoju delova sistema. Svaka grana ima sekvencu izmena. Moguće je odvojiti novu granu iz određene grane, kao i spojiti jednu granu sa drugom granom (operacija *merge*). Vidimo da graf sa slike sadrži tri grane nazvane *master, Feature-1* i *Feature-2*. Generalno se preporučuje često kreiranje grana tako da se svaka nova funkcionalnost razvija nezavisno od postojećih. Kada se razvoj funkcionalnosti završi, ta grana se onda integriše u granu koja sadrži glavnu verziju koda.

Git repozitorijum je predstavljen kao folder na računaru u kojem se nalaze fajlovi i folderi koji se nalaze u repozitorijumu. Sadržaj tih fajlova je u skladu sa jednom od verzija evidentiranih u repozitorijumu. Fajlovi mogu da odražavaju sadržaj iz bilo koje od verzija. Koju verziju će fajl sistem sadržati definisano je referencom koja se naziva HEAD. U svakom trenutku, HEAD referencira jednu od verzija iz grafa. Moguće je postaviti HEAD da pokazuje na bilo koju od verzija u grafu. Na prethodnoj slici vidimo da HEAD referencira poslednju verziju na master grani, što znači da će u fajl sistemu fajlovi imati sadržaj kakav imaju u toj verziji.

Pored fajlova sadržanih u određenoj verziji, repozitorijum sadrži i specijalni `.git` folder u kojem se nalaze konfiguracioni fajlovi repozitorijuma, kao i programski kod iz svih verzija

koje su dodate u repozitorijum. Skladištenjem svih verzija omogućeno je „pomeranje“ HEAD reference na bilo koju od verzija. Zbog efikasnijeg skladištenja, kod se skladišti kompresovan.

Nakon instalacije git sistema, moguće je iz komandne linije pozivati git naredbe.

Kreiranje repozitorijuma

Za kreiranje git repozitorijuma na lokalnom računaru, potrebno je izvršiti inicijalizaciju repozitorijuma u željenom folderu putem naredbe:

```
git init
```

Ovim je folder u kojem je izvršena naredba postao git repozitorijum.

Rad sa git repozitorijumom je moguće konfigurisati definisanjem vrednosti za različite konfiguracione parametre. Minimalno, neophodno je konfigurisati podatke o korisniku koji postavlja izmene u repozitorijum obzirom da je to jedna od informacija koje se evidentiraju za svaku izmenu. Konfigurisanje imena, odnosno e-maila korisnika se vrši naredbama:

```
git config user.name "Petar Petrovic"
git config user.email "petar@mail.com"
```

Na ovaj način, korisnik će biti konfigurisan samo za repozitorijum iz kojeg se naredba izvršava. Moguće je i globalno konfigurisati podatke o korisniku za sve repozitorijume na lokalnom računaru. Tada se naredbama dodaje parametar --global, što je prikazano u nastavku:

```
git config --global user.name "Petar Petrovic"
git config --global user.email "petar@mail.com"
```

Iako je git distribuiran sistem u kojem su repozitorijumi ravnopravni, najčešće jedan repozitorijum ima specijalnu ulogu glavnog repozitorijuma, sa kojeg se primaju, odnosno na koji se šalju fajlovi sa repozitorijuma na lokalnom računaru. Repozitorijume na lokalnom računaru nazivamo lokalni repozitorijum, dok se glavni repozitorijum naziva udaljeni (eng. *remote*) repozitorijum. Udaljeni repozitorijum je postavljen na mrežno dostupnom računaru, koji može biti instaliran na nekom internom serveru ili je moguće koristiti internet platforme koje pružaju postavljanje git repozitorijuma. Repozitorijumi postavljeni na ovakvim platformama su dostupni putem interneta. Popularne platforme ovog tipa su GitHub (www.github.com) i GitLab (www.gitlab.com).

Čest scenario rada sa Git sistemom je da se najpre kreira udaljeni repozitorijum. Nakon toga, potrebno je na lokalnom računaru napraviti kopiju udaljenog repozitorijuma. Sve izmene u kopiji će kasnije biti moguće poslati na udaljeni repozitorijum. Takođe, nove izmene na udaljenom repozitorijumu mogu biti preuzete u lokalni repozitorijum. Pravljenje kopije udaljenog repozitorijuma se naziva kloniranje repozitorijuma i vrši se sledećom naredbom:

```
git clone <identifikator udaljenog repozitorijuma>
```

U nastavku je primer korišćenja clone naredbe koja kreira *helloworld* projekat sa udaljenog repozitorijuma postavljenog na GitHub platformi:

```
git clone https://github.com/petar/helloworld.git
```

Ova naredba u trenutnom folderu kreira novi folder koji predstavlja git repozitorijum. Sadržaj foldera će odgovarati sadržaju udaljenog repozitorijuma koji se klonira.

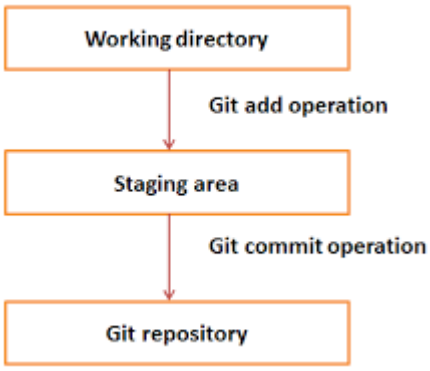
Kloniranjem je u konfiguraciji repozitorijuma uspostavljena veza između lokalnog i udaljenog repozitorijuma, tako da je moguće kasnije vršiti sinhronizaciju sadržaja ova dva repozitorijuma. U slučaju da je lokalni repozitorijum nastao inicijalizacijom (putem naredbe init), a ne kloniranjem, moguće je naknadno postaviti udaljeni repozitorijum sa kojim se ovaj lokalni repozitorijum sinhronizuje. Primer naredbe koja vrši ovu operaciju je dat u nastavku.

```
git remote add origin https://github.com/petar/helloworld.git
```

Postavljanje izmena na repozitorijum

U toku rada, u lokalnom folderu koji predstavlja git repozitorijum vrše se izmene nad fajlovima. Izmene nad fajlovima se ne postavljaju automatski u git repozitorijum. Umesto toga, potrebno je eksplicitno određeni skup izmena dodati kao novu verziju u repozitorijum. Tada se pojavljuje novi *commit* u git grafu.

Dodavanje izmena na repozitorijum se vrši u dve faze. Fajlovi koji čine izmenu se najpre moraju dodati u tzv. *staging area*, a nakon toga se fajlovi koji su u tom prostoru, mogu dodati u repozitorijum. Ovo je ilustrovano na slici.



* preuzeto sa tutorialspoint.com

Kao što je i navedeno na slici, postavljanje fajlova u *staging area* se vrši operacijom *add*. Primer korišćenja ove naredbe za postavljanje jednog fajla je u nastavku:

```
git add proba.txt
```

Ukoliko želimo da dodamo sve izmenjene fajlove, to možemo uraditi sledećom naredbom:

```
git add -A
```

Fajlovi koji se nalaze u *staging area* se u repozitorijum dodaju operacijom *commit*, koja je ilustrovana sledećim primerom.

```
git commit –m "Opis izmene"
```

Vidimo da parametar –m specificira poruku koja opisuje nove verziju koja se ovom prilikom postavlja u repozitorijum. Opisi izmena su veoma važni zbog kasnijeg praćenja istorije izmena.

Ovakvo postavljanje izmena u dve faze omogućuje logičko grupisanje izmena nad fajlovima. Tako se određeni broj fajlova može postaviti kao jedna verzija, a ostatak kao druga. Time se u git grafu dobijaju dva *commit* čvora, pa je kasnije lakše pronaći željenu verziju u istoriji izmena.

Ignorisanje fajlova

Određeni fajlovi koji se skladište u fajl sistemu ne trebaju da se skladište u samom repozitorijumu. Ovo se uglavnom odnosi na neke privremene fajlove koje softverske aplikacije generišu. Takođe, fajlovi koji se generišu na osnovu drugih fajlova se ne moraju skladištiti u repozitorijumu. Na primer, u Java projektu, .class fajlovi nastaju na osnovu izvornih .java fajlova, pa je dovoljno u repozitorijumu skladištiti samo .java fajlove.

Zbog lakšeg upravljanja fajlovima u repozitorijumu, moguće je naznačiti da određene fajlove git ignoriše. Ovo se vrši kreiranjem fajla pod nazivom `.gitignore` u kojem se navode šabloni imena fajlova koje treba git da ignoriše. Najčešće se .gitignore fajl postavlja u korenski folder repozitorijuma. Tada će se ignorisanje odnositi na sve fajlove u repozitorijumu. Primer sadržaja `.gitignore` fajla je dat u nastavku:

```
*.class
~$*.docx
~*.tmp
```

Fajl iz primera specificira da će Java class fajlovi, kao i Microsoft Word privremeni fajlovi biti ignorisani.

Stanje i istorija repozitorijuma

Kao što smo videli, sadržaj fajl sistema se ne poklapa uvek sa stanjem u git repozitorijumu. Fajlovi u direktorijumu se mogu nalaziti u jednom od tri stanja:

- samo u fajl sistemu
- dodati u *staging area*
- postavljeni u repozitorijum

Takođe, u svakom trenutku jedna verzija iz repozitorijuma je tekuća (ona na koju ukazuje referenca HEAD).

Naredbom *status* moguće je proveriti stanje repozitorijuma. Ovo se odnosi na stanje fajlova, koja je tekuća verzija i slično. Dat je primer korišćenja *status* naredbe i mogući izlaz:

```
$ git status

On branch master

Your branch is up-to-date with 'origin/master'.



Untracked files:

   (use "git add <file>..." to include in what will be
committed)
```

U prikazanom primeru, tekuća verzija je na grani master. *Staging area* je prazna i ne postoje izmene koje bi operacijom *commit* mogle biti postavljene na repozitorijum. Sam fajl sistem sadrži jedan novi fajl pod nazivom *test.txt* koji se ne nalazi u repozitorijumu i nije još uvek dodan u *staging area*.

Obzirom da git evidentira sve verzije u repozitorijumu, moguće je pregledati istoriju verzija. Ovo se vrši naredbom *log*. Primer naredbe i izlazaje dat u nastavku.

```
$ git log

commit 0abbf9bc7d33c81c91984a8927e343e797f20381

Author: Goran Savic <savic071@gmail.com>

Date:   Wed Jun 14 09:47:48 2017 +0200


    Dodavanje materijala za termin 23
```

Primer prikazuje dve verzije koje repozitorijum sadrži. Vidimo da je za svaku verziju ispisan identifikator (naziva se heš verzije), autor, datum postavljanja i opis izmene.

Kao što smo objasnili, fajl sistem prikazuje fajlove iz određene verzije sa repozitorijuma. Ta tekuća verzija definisana je HEAD referencom. Komanda *checkout* omogućuje pomeranje HEAD reference na proizvoljnu verziju. Potrebno je naznačiti na koju verziju se HEAD pomera. Jedan način da se to naznači je navođenjem heš vrednosti verzije. Dat je primer korišćenja naredbe checkout, koja pomera HEAD referencu na drugu verziju iz ranije prikazane istorije verzija.

`git checkout c19a915`

Uglavnom je dovoljno navesti prvih sedam karaktera heš vrednosti pri referenciranju verzije, što je urađeno i u prikazanom primeru. Izvršavanjem *checkout* komande sadržaj fajlova u fajl sistemu se menja tako da bude u skladu sa sadržajem koji su fajlovi imali u verziji na koju se pomeramo. Na ovaj način moguće je istražiti kompletnu istoriju verzija.

Kreiranje grana

Objasnili smo da git graf može da ima više grana. Kreiranje grana omogućeno je *branch* komandom. Ova komanda pravi novu granu koja počinje na mestu na kojem se trenutno nalazi HEAD referenca. Zbog kasnijeg referenciranja, svaka grana ima svoj naziv. Dat je primer kreiranja grane pod nazivom *Feature-1*.

```
git branch Feature-1
```

Iako je grana kreirana, HEAD referenca i dalje pokazuje na granu sa koje smo izvršili grananje. Da bi se HEAD referenca pomerila na novokreiranu granu, potrebno je izvršiti naredbu checkout:

```
git checkout Feature-1
```

Sada se HEAD referenca nalazi na grani Feature-1. Generalno operacijom checkout možemo pomeriti HEAD referencu na bilo koju granu. *Commit* operacije će dodavati nove verzije na ovoj grani git grafa i pri svakoj *commit* operaciji HEAD referenca će biti pomerena na novokreirani *commit* čvor.

Prikaz svih grana koje trenutno postoje u grafu može se dobiti naredbom

```
git branch
```

Grana koju trenutno HEAD referencira je u prikazu rezultata označena znakom *.

Moguće je spojiti izmene definisane na različitim granama git grafa. Operacija *merge* u granu na kojoj smo trenutno pozicionirani ubacuje izmene napravljene na nekoj drugoj grani. Ovo rezultira novim čvorom u git grafu koji sadrži izmene napravljene na obe grane koje se spajaju. Takvi čvorovi u grafu su potomci dva čvora (jer su nastali spajanjem dve verzije, a ne menjanjem jedne verzije kao kod klasičnih *commit* čvorova).

Ako je HEAD trenutno pozicioniran na granu *master*, sledeća naredba će u tu granu ubaciti i izmene koje sadrži poslednji čvor grane *Feature-1:*

```
git merge Feature-1
```

Na prvoj slici u ovom materijalu može se videti kako git graf izgleda nakon što smo sa *master* grane kreirali dve grane *Feature-1* i *Feature-2* i izvršili spajanje tih grana sa *master* granom.

Konflikti

Pri spajanju grana git automatski pronalazi razlike u sadržajima verzija koje se spajaju, tako da kao rezultat dobijamo fajlove koji sadrže izmene iz obe verzije. Ako je u verzijama koje se spajaju izvršena izmena nad istim fajlovima u istim linijama fajla, tada nije moguće automatski izvršiti spajanje sadržaja. Git će u tim slučajevima prijaviti **konflikt** pri spajanju fajla, jer nije moguće odlučiti koja verzija linije fajla treba da se nađe u spojenoj verziji fajla.
Pogledajmo na primeru kada konflikt nastaje i kako se razrešava. U nastavku je prikazan sadržaj istog fajla u verzijama koje se nalaze na različitim granama (levo je verzija sa *master* grane, a desno sa grane *Feature-1*). Vidimo da je linija broj pet izmenjena u obe verzije.

master
test.txt

```
prva
druga
treca
cetvrta
ABCD
sesta
sedma
```

Feature-1
test.txt

```
prva
druga
treca
cetvrta
XYZW
sesta
sedma
```

Ukoliko se sada izvrši spajanje verzije sa grane *Feature-1* u granu *master*, git će prijaviti konflikt u fajlu i označiti specijalnim karakterima u samom fajlu linije koje su u konfliktu. Za ove linije, u sadržaj fajla će biti ubačene verzije linija sa obe grane koje se spajaju. Pogledajmo, kako bi izgledao fajl iz prethodnog primera u konfliktu koji bi nastao nakon spajanja.

master
test.txt

```
prva
druga
treca
cetvrta
<<<<<<< HEAD
ABCD
=======
XYZW
>>>>>>> Feature-1
sesta
sedma
```

Vidimo da fajl sadrži obe verzije linije. Za svaku liniju je znakovima > odnosno < označeno odakle je linija preuzeta. Na master grani se trenutno nalazi HEAD referenca, pa je ova linija označena sa HEAD, dok je linija sa grane *Feature-1* označena imenom te grane. Znakovi ======= razdvajaju linije sa različitih grana.

Rešavanje konflikta se svodi na ručno uređivanje sadržaja fajla tako što se obrišu specijalni karakteri i za konfliktne linije unese željeni sadržaj. Može se ostaviti sadržaj iz obe linije, samo iz jedne od linija ili uneti potpuno nov sadržaj za te linije. Nakon

uređivanja sadržaja fajla, neophodno je fajl dodati u *staging area* operacijom add i putem operacije *commit* ubaciti izmenu na repozitorijum:

```
git add test.txt
git commit –m "Razresen konflikt"
```

Sinhronizacija sa udaljenim repozitorijumom

Sve operacije koje smo do sada analizirali (osim operacije *clone*) odnose se na rad sa lokalnim repozitorijumom. Kao što smo objasnili, obično postoji udaljeni repozitorijum sa kojim se lokalni repozitorijum sinhronizuje.

Izmene izvršene nad lokalnim repozitorijumom se mogu poslati na udaljeni repozitorijum. Preciznije, u određenu granu udaljenog repozitorijuma je moguće ubaciti sadržaj određene grane lokalnog repozitorijuma. Slanje izmena na udaljeni repozitorijum se vrši operacijom *push* na sledeći način:

```
git push <identifikator repozitorijuma> <ime grane koju šaljemo>
```

Grana udaljenog repozitorijuma na koju se izmene šalju se postavlja u konfiguraciju automatski pri kloniranju repozitorijuma, ali ju je moguće i eksplicitno specificirati. Moguće je izvršiti sinhronizaciju sa bilo kojim udaljenim repozitorijumom, ali se u radu najčešće komunicira sa jednim udaljenim repozitorijumom. Taj repozitorijum ne moramo navoditi preko identifikatora, već možemo korisititi alias *origin*. Pri kloniranju repozitorijuma, kao *origin* se postavlja repozitorijum koji kloniramo. Dat je primer postavljanja lokalnih izmena na udaljeni repozitorijum:

```
git push origin master
```

Prikazana naredba će poslati lokalne izmene sa *master* grane na udaljeni repozitorijum.

U pravilu, veći broj lokalnih repozitorijuma postavlja izmene na udaljeni repozitorijum. Zato je potrebno u lokalni repozitorijum preuzeti izmene koje su se pojavile na udaljenom repozitorijumu. Ovo se vrši operacijom pull na sledeći način:

```
git pull
```

Prikazana operacija preuzima u lokalni repozitorijum izmene sa udaljenog repozitorijuma koji je definisan kao *origin*. Preciznije, operacija pull preuzima izmene sa grane udaljenog repozitorijuma i spaja ih sa trenutnom granom lokalnog repozitorijuma. Tako da je naredba pull analogna izvršavanju naredne dve naredbe:

```
git fetch
```

```
git merge origin/master
```

Naredba *fetch* samo preuzima granu sa udaljenog repozitorijuma. Grana će biti uskladištena pod nazivom *origin/master,* ako je reč o *master* grani udaljenog repozitorijuma. Ova grana još uvek nije spojena sa granom lokalnog repozitorijuma. To se vrši gore prikazanom naredbom *merge*.

Dakle, preuzimanje izmena se repozitorijuma uključuje spajanje grana. Kao i kod bilo kojeg drugog spajanja grana i ovde je moguć konflikt, koji se razrešava na gore pomenuti način.

Pomenimo još da operacija *push* neće uspeti ako je udaljeni repozitorijum pretrpeo izmene od kako je prethodni put preuzet njegov sadržaj u lokalni repozitorijum. Da bi operacija uspela, neophodno je najpre naredbom *pull* preuzeti nove izmene sa udaljenog repozitorijuma. Time će te izmene biti spojene sa izmenama u lokalnom repozitorijumu i tek nakon toga je moguće izvršiti operaciju *push*.