

Git - sistem za kontrolu verzija

1. Uvod

Sistem za kontrolu verzija (Version Control System - VCS) omogućava softverskim timovima da održavaju i upravljaju svojim izvornim kodom. Git je najpopularniji danas i nastao je od strane Linusa Torvalds-a u aprilu 2005. godine nakon promene politike licenciranja BitKeeper-a koji je do tada korišćen za razvoj Linux kernel-a. Ono što je karakteristika ovog VCS-a jeste da je akcenat na praćenju modifikacija koda, a ne fajlova.

Developer-i radeći u timovima konstantno dodaju nov kod i modifikuju postojeći, pri čemu se može desiti da rade na potpuno različitim delovima projekta, ali je češći slučaj je da se rad preklapa u nekoj meri (manjoj ili većoj). Kontrola verzija treba da omogući lakše rešavanje problema konflikata, tako što će identifikovati delove koda koji su menjani od strane više ljudi i da pomogne u razrešavanju istih. Da li će konflikt biti razrešen kako treba, zavisi od samih programera, odnosno svaka nova promena može izazvati probleme u kodu koji je pisan i bio funkcionalan u prošlosti.

Osnovne osobine Git-a su:

- jako dobra brzina,
- skalabilnost,
- jednostavan dizajn,
- veliki broj operacija je lokalne prirode,
- dobra podrška za izuzetno velike projekte,
- praćenje sadržaja umesto samih fajlova.

2. Arhitektura

Karakteristika Git-a je da ima Three-tree arhitekturu, odnosno postoje tri različita sloja gde se promene mogu nalaziti sa njegovog stanovišta (slika 1):

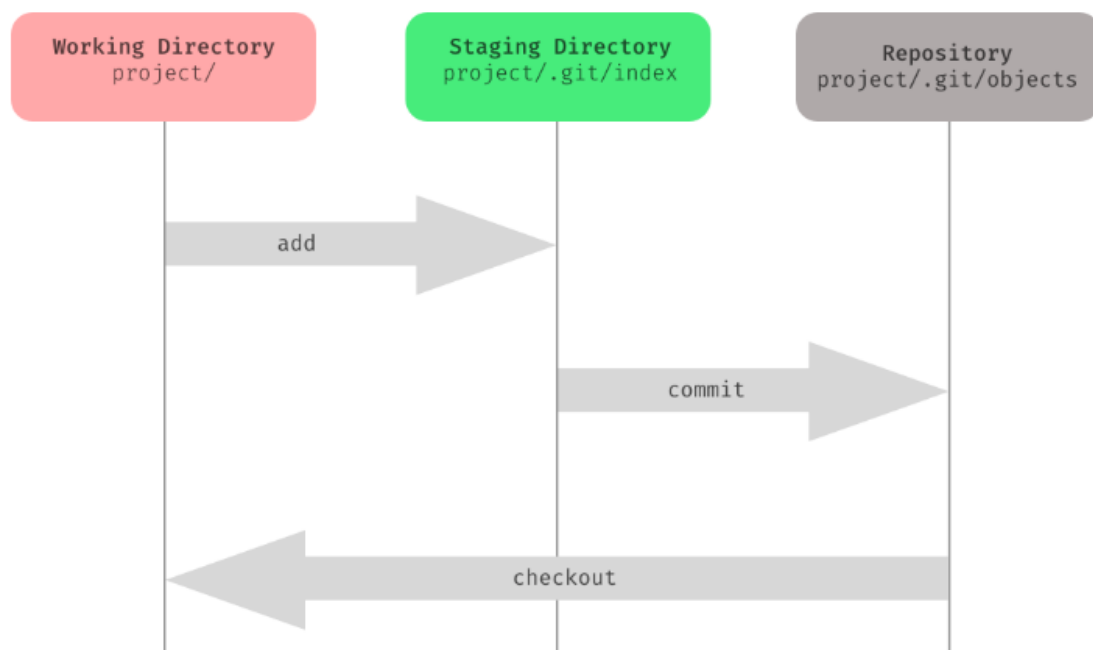
- Working directory (radno stablo) - Predstavlja projektni direktorijum koji se nalazi na fajlsistemu računara pri čemu se ovde mogu nalaziti izmene koje još nisu trajno zabeležene (commit-ovane).
- Index (pripremna zona) - Koristi se za pregled i dodatno spremanje narednog commit-a. Predstavlja fajl, a ne skup direktorijuma.
- Repository (lokalni repozitorijum) - Nakon svakog commit-a, čuvaju se promene kao commit, tree i blob objekti.

Pripremna zona i repozitorijum se nalaze unutar skrivenog foldera pod nazivom `.git` unutar radnog stabla. Na slici 1 su redom bojama označena sva 3 sloja.



Slika 1 - Troslojna arhitektura

Svaku promenu datoteke ili dodavanje i brisanje istih iz radnog direktorijuma (sloj označen crvenom bojom) je neophodno stage-ovati u pripreмноj zoni (zelena boja), tj. vrši se sređivanje za predstojeći commit, odnosno kreiranje objekata u lokalnom repozitorijumu (siva boja). Međutim sam proces ne mora biti isključivo jednosmeran. Iz repozitorijuma možemo preuzeti sadržaj bilo kog prethodnog commit-a koji će biti prenet u radno stablo. Tok je prikazan na slici 2.



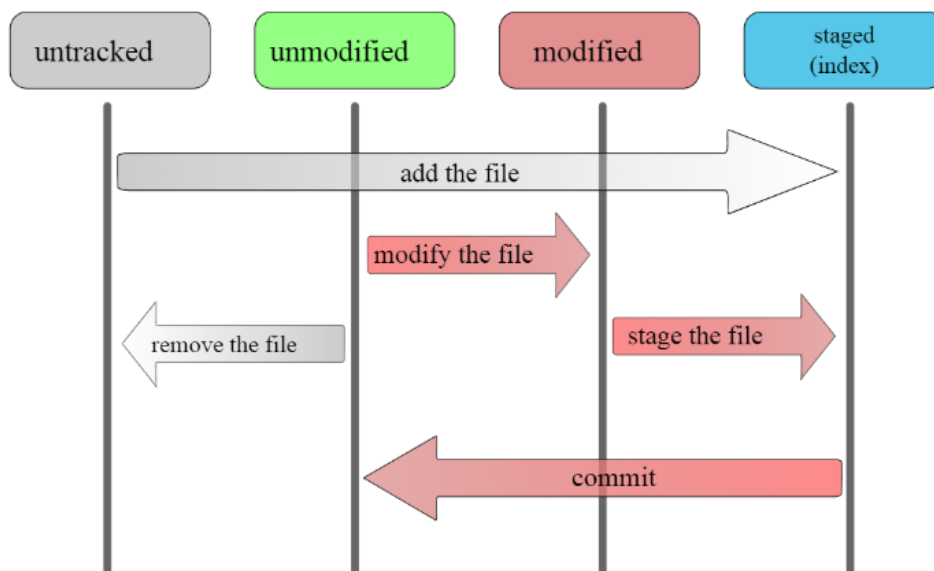
Slika 2 - Tok promena kroz slojeve

Fajlovi u Git repozitorijumu se mogu naći u više stanja (slika 3):

- Untracked - Fajlovi o kojima Git ne vodi računa. Najčešće su to generisani fajlovi na osnovu izvornog koda.
- Tracked - Ovde razlikujemo i dva podstanja:
 1. Unmodified - Fajlovi koji se prate, ali nisu modifikovani.
 2. Modified - Prelazi se iz prethodno opisanog stanja usled nastajanja neke

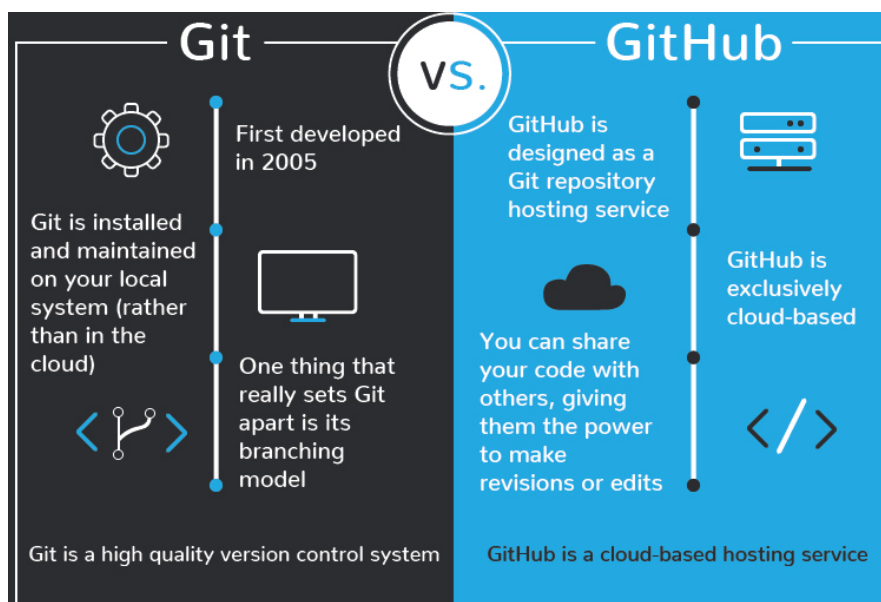
modifikacije.

3. Staged - Označava da fajl treba biti uključen u naredni commit. Nakon commit-a, fajlovi ponovo prelaze u unmodified stanje.



Slika 3 - Stanja fajlova i tranzicije

Git nam omogućava da verzioniramo i pratimo izvorni kod lokalno, ali takođe obezbeđuje i deljenje svih promena sa udaljenim (remote) repozitorijumom. GitHub i GitLab su najpopularniji servisi za hostovanje udaljenog repozitorijuma na cloud-u. Pored standardnih funkcionalnosti, pružaju još puno različitih opcija koje dodatno olakšavaju razvoj kompleksnih projekata u timovima.



Slika 4 - Razlike između Git-a i Github-a

3. Upotreba Git-a

Kao hosting rešenje biće korišćen GitLab u primerima, ali se možete opredeliti i za GitHub. Da bi mogli da beležimo promene u našem kodu, neophodno je najpre lokalno instalirati Git, a zatim napraviti udaljeni repozitorijum gde će biti postavljane i praćene sve promene na projektu.

3.1 Instalacija

Potrebno je preuzeti installer za Windows sa [link-a](#) i putem wizard-a izvršiti instalaciju. Nije neophodno ništa menjati u već postavljenim opcijama u wizard-u, već samo obratiti pažnju da je čekirana opcija za integraciju sa Windows explorer-om. To omogućava pokretanje terminala na vrlo jednostavan način. Dovoljno je otvoriti željeni direktorijum i zatim nakon desnog klika izabrati **Git Bash Here**, što će otvoriti elegantniji terminal od **command prompt-a**, koji je već pozicioniran na ispravnoj putanji. Za Linux, dovoljno je izvršiti komandu u terminalu **apt-get install git** za Debian-based distirbucije, dok je za RPM-based distirbucije neophodno izvršiti komandu **dnf install git**.

3.2 Kreiranje udaljenog repozitorijuma

Nakon registracije i logovanja na [GitLab](#), na prikazu za sve repozitorijume, treba odabrati opciju za kreiranje novog projekta, nakon čega je neophodno popuniti osnovne podatke. Dovoljno je definisati naziv projekta i podesiti da je isti privatan.

Projects New project

Blank project | Create from template | Import project | CI/CD for external repo

Project name
Test

Project URL
https://gitlab.com/ greenenvy

Project slug
test

Want to house several dependent projects under the same namespace? [Create a group](#).

Project description (optional)
Description format

Visibility Level ?
☒ Private
Project access must be granted explicitly to each user.
☐ Public
The project can be accessed without any authentication.

☒ Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project Cancel

Dodatno, moguće je definisati README fajl. Kada neko otvori stranicu projekta na GitLab-u, prikazaće mu se detaljan opis projekta. Namena ovog fajla je da upozna korisnika sa projektom, odnosno da mu da instrukcije kako da projekat upotrebi, pokrene itd. Kako će ta datoteka izgledati, zavisi od programera, ali je dobra praksa da opis bude što bogatiji. U kontekstu predmeta, u njemu će biti navođeno uputstvo za asistenta i profesora oko pokretanja projekta i sadržaćе sve dodatne napomene za koje bude bilo potrebe. Na prethodnoj slici, umesto manuelnog kreiranja, dovoljno je čekirati poslednju opciju koja će kreirati fajl koji se kasnije može editovati u svakom commit-u.

3.3 Osnovni workflow

Nakon kreiranja repozitorijuma, možemo raditi sa njim na našem lokalnom računaru. Neki osnovni tok podrazumeva:

1. Kloniranje udaljenog repozitorijuma (ako isti ne postoji) na lokalni računar (ovo se generalno radi jednom).
2. Vršiti izmena fajlova, kreiraju se nove grane itd.
3. Promene se smeštaju u pripremnu zonu (indeks).
4. Vršiti se trajno beleženje promena, odnosno izvršava se commit operacija.
5. Lokalne promene se push-uju na udaljeni repozitorijum.

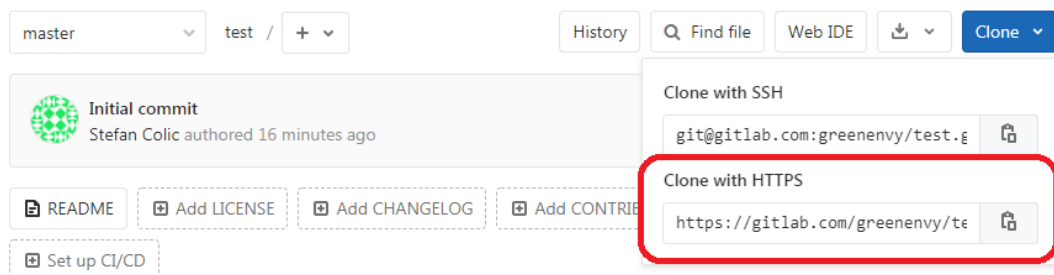
Sem kloniranja koje se najčešće izvršava samo jednom (na jednom lokalnom računaru), operacije od dva do pet se ponavljaju proizvoljan broj puta.

3.3.1 Kloniranje repozitorijuma

Ova operacija se obavlja izvršavanjem komande u okviru terminala:

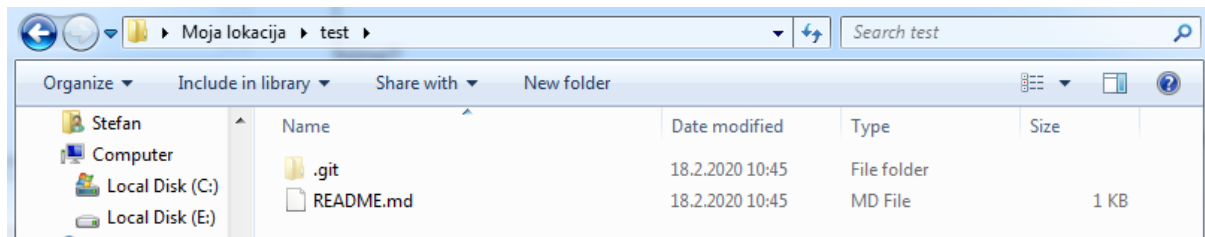
git clone <url repozitorijuma>

Na početnoj stranici projekta, moguće je videti kako izgleda url projekta. Dodatno je neophodno otvoriti terminal na željenoj lokaciji u fajl sistemu (desni klik -> Git Bash Here) i izvršiti navedenu komandu sa adekvatnim URL-om.



```
Stefan@Stefan-PC MINGW64 ~/Desktop/Moja lokacija
$ git clone https://gitlab.com/greenenvy/test.git
Cloning into 'test'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 204 bytes | 14.00 KiB/s, done.
```

U konkretnom primeru sa slike, biće kreiran **test** folder (naziv diktiran prilikom kreiranja na GitLab-u) unutar foldera **Moja lokacija**, u kome će se naći pomenuti README fajl unutar radnog direktorijuma i skriveni **.git** direktorijum u kome se nalaze index fajl i sam repozitorijum kao skup commit, tree i blob objekata (i još neke dodatne stvari).



3.3.2 Rad sa radnim direktorijumom

Nakon što smo udaljeni repozitorijum klonirali kod sebe, neophodno je u terminalu pozicionirati se u sam radni direktorijum, odnosno u ovom slučaju test. Nije potrebno otvarati novi terminal, već u postojećem samo ući u test folder (**cd test**). Datoteke i druge direktorijume kreiramo baš u radnom direktorijumu, dok skriveni **.git** folder nije potrebno dirati manuelno. Njegov sadržaj će biti kreiran i modifikovan izvršavanjem različitih komandi u terminalu.

Nakon što je sve prevučeno, mogu se kreirati razni projekti (WPF, Spring Boot itd.), odnosno mogu se dodavati, brisati i modifikovati postojeći fajlovi i direktorijumi, pri čemu proces razvoja teče standardnim putem. Onog trenutka kada je implementirana neka funkcionalnost, kada je ispravljen neki postojeći bug, kada je odrađena neka količina posla, neophodno je pripremiti datoteke za trajno beleženje (commit) u lokalni repozitorijum.

U bilo kom trenutku je moguće videti trenutni status svakog fajla u odnosu na poslednji zabeležen commit komandom:

git status

Konkretna komanda će prikazati aktivnu granu (o tome više u nastavku materijala), koje datoteke su izmenjene, a nisu pripremljene za dodavanje (stanje modified) i datoteke koje se trenutno ne prate (stanje untracked).

```
Stefan@Stefan-PC MINGW64 ~/Desktop/Moja lokacija/test (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   invert_dict.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        sort_strings.py

no changes added to commit (use "git add" and/or "git commit -a")
```

Pre commit-a je neophodno pripremiti isti, što je moguće obaviti sledećim operacijama:

```
# Dodavanje sadržaja
$ git add MojaKlasa.py
$ git add .

# Interaktivno dodavanje sadržaja
$ git add -p .

# Dodavanje/uklanjanje svih
# novih/modifikovanih/obrisanih fajlova
$ git add -A .

# Uklanjanje iz radnog stabla i indeksa.
$ git rm MojaKlasa.java

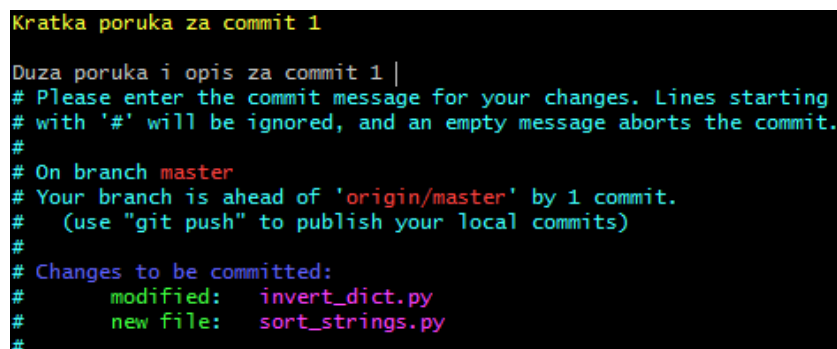
# Uklanjanje iz indeksa uz brisanje iz
# repozitorijuma.
# Fajl ostaje u radnom stablu ali se više ne prati.
$ git rm --cached MojaKlasa.java

# Uklanjanje promena zabeleženih u indeksu
$ git reset HEAD ili samo git reset
```

Nakon uspešne pripreme indeksa, sledi trajno beleženje promena komandom:

git commit

Navedena komanda će otvoriti **Vim** editor.



```
Kratka poruka za commit 1

Duza poruka i opis za commit 1 |
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
# Changes to be committed:
#   modified:   invert_dict.py
#   new file:   sort_strings.py
#
```

U prvom redu je neophodno napisati kratku poruku za sam commit, gde nakon iste može slediti duža verzija poruke sa dodatnim opisom. Nakon popunjavanja poruke i opisa, neophodno je pritisnuti **ESC** taster, uneti **:wq** (write and quit) i pritisnuti **ENTER**. Promene će biti trajno zabeležene i sve datoteke će promeniti stanje u unmodified. Ako je **VIM** editor suviše kompleksan, u instalaciji je moguće navesti neki drugi editor poput **Notepad++**.

Postoji i kraći oblik commit-a bez otvaranja editora, gde je neophodno uneti samo kratku poruku:

git commit -m <poruka>

Ukoliko indeks nije pripremljen, može se iskoristiti prečica:

git commit -a

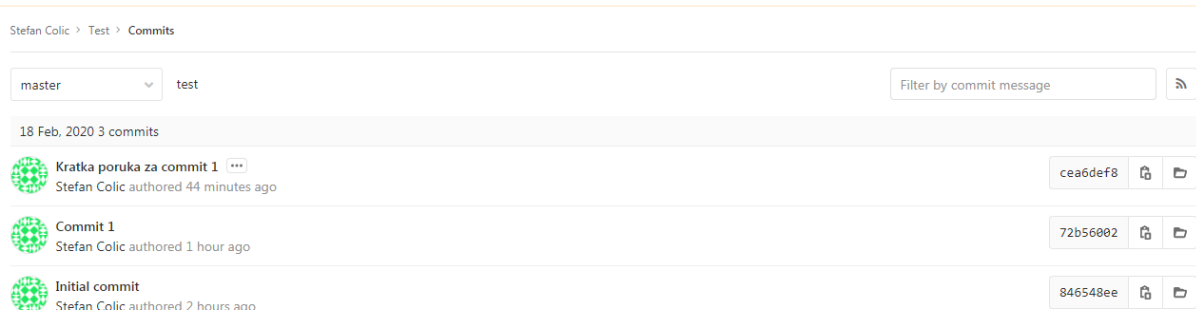
Neophodno je voditi računa koji se fajlovi track-uju, odnosno kao što je u početnom delu teksta naglašeno, ne želimo da na GitLab pošaljemo mašinski generisane fajlove, bilo da su to neki build artifacts ili fajlovi koje generiše IDE. Za tu svrhu se kreira .gitignore fajl u kome se može navesti šta će sve biti ignorisano. Najčešće prilikom inicijalnog kreiranja projekta, generiše se već popunjen .gitignore fajl, pa tu nije neophodno praviti puno izmena. Primer jednog .gitignore fajla:

```
1 /target/
2 !.mvn/wrapper/maven-wrapper.jar
3
4 ### STS ###
5 .apt_generated
6 .classpath
7 .factorypath
8 .project
9 .settings
10 .springBeans
11 .sts4-cache
12
13 ### IntelliJ IDEA ###
14 .idea
15 *.iws
16 *.iml
17 *.ipr
18
19 ### NetBeans ###
20 /nbproject/private/
21 /build/
22 /nbbuild/
23 /dist/
24 /nbdist/
25 /.nb-gradle/
```

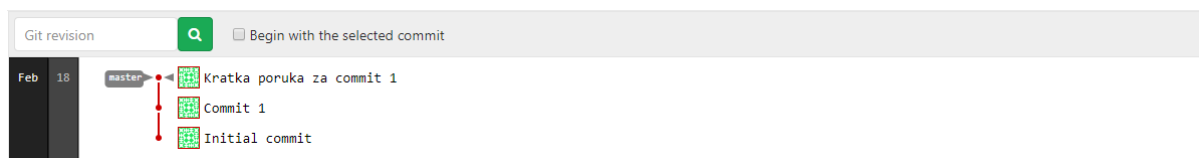
Nakon commit-a, potrebno je poslati lokalne promene na udaljeni repozitorijum. To je moguće odraditi komandom:

git push

Ukoliko nije bilo konflikata (više o tome u nastavku materijala), odlaskom na početnu stranicu projekta na GitLab-u, promene će biti evidentne, odnosno biće prisutan sadržaj koji smo trajno zabeležili. Odlaskom na Repository -> Commits u side panel-u moguće je videti sve commit-e za izabranu granu.

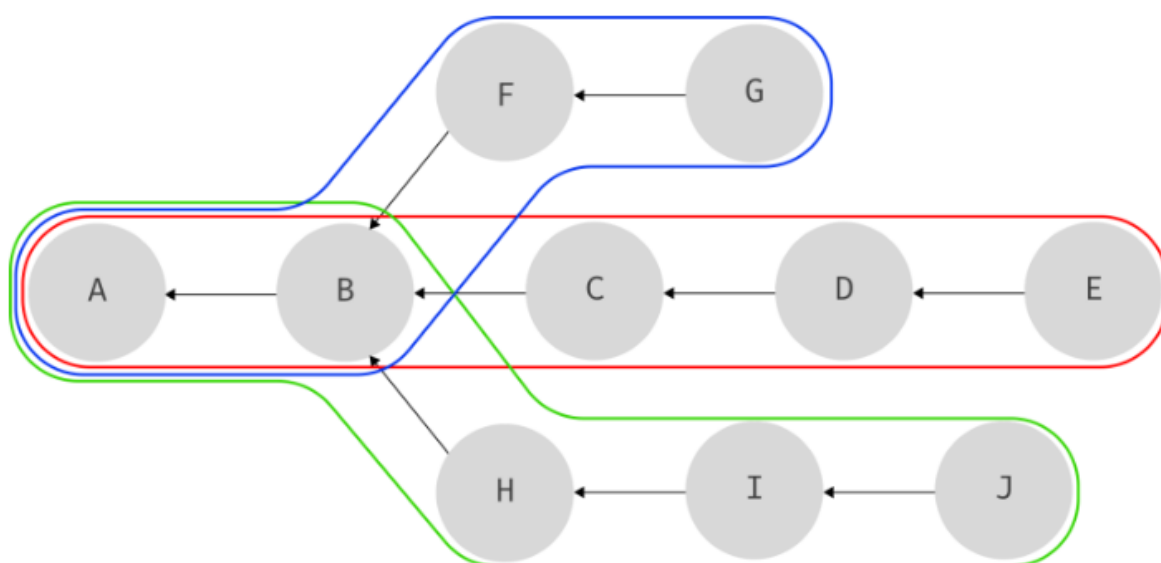


Na Repository -> Graph je moguće videti vizuelni prikaz grafa commit-ova.



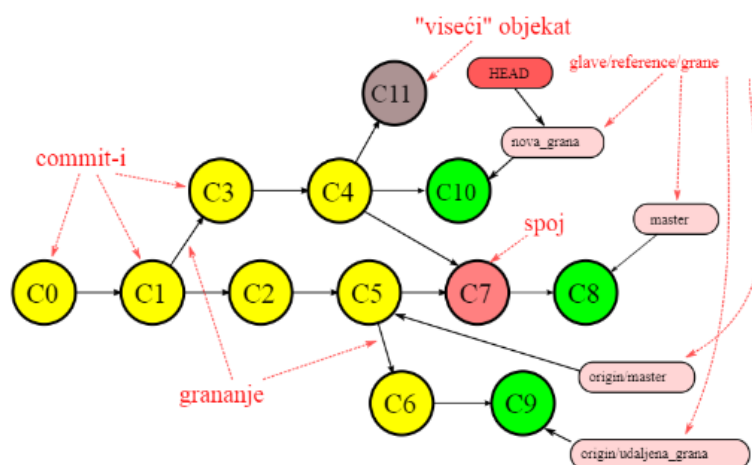
3.3.3 Rad sa granama

Commit-ovi u Git-u obrazuju usmeren acikličan graf, odnosno ukoliko bismo krenuli iz bilo kog čvora, prateći smerove na vezama, nikada ne bismo mogli da dođemo ponovo u istu lokaciju (nema ciklusa). Pri kreiranju repozitorijuma i pri inicijalnom commit-u, graf će sadržati samo jedan čvor koji predstavlja naveden commit i samo jednu granu, pod nazivom **master**. Moguće je kreirati i druge grane, koje predstavljaju alternativne tokove razvoja na kojima se razvija neka funkcionalnost, rešavaju se određeni bug-ovi itd. Usled veoma lakog kreiranja grana i upravljanja istim, ohrabruje se njihova upotreba.



Grane u Git repozitorijumu mogu biti:

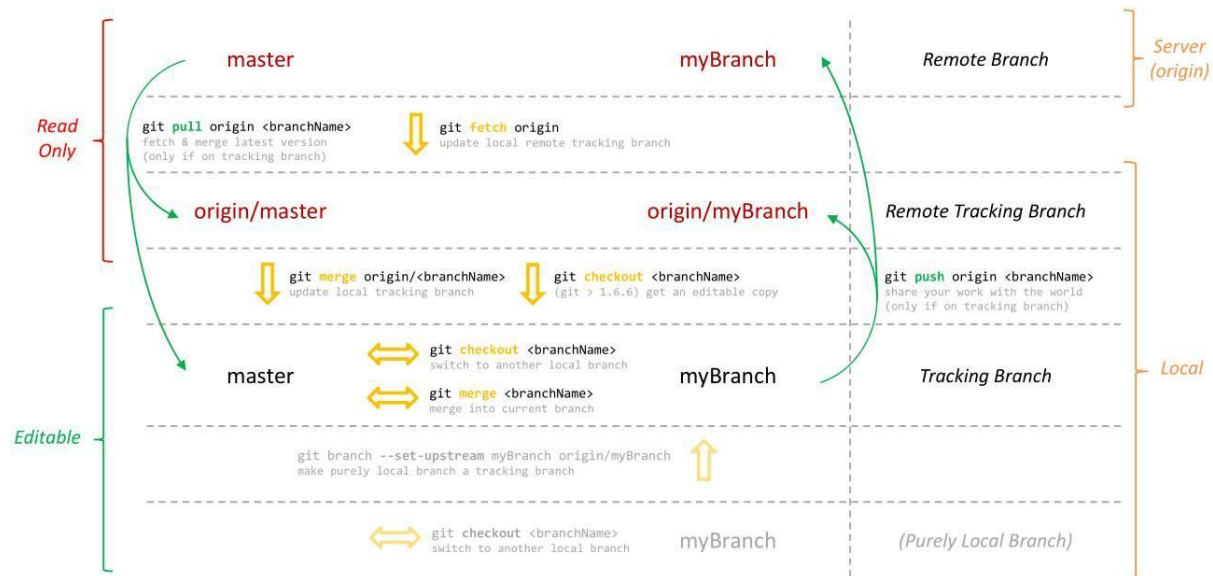
- Lokalne - Nastale u lokalnom repozitorijumu i ne prate druge grane.
- Tracking - Lokalne grane koje prate druge grane. Najčešće prate **remote tracking** grane.
- Remote tracking - Grane nastale u udaljenom repozitorijumu koje se kloniraju u lokalni.



Grane su reference (refs u skrivenom .git direktorijumu) i one samo pokazuju na poslednji commit alternativnog toka. U samom primeru:

- nova_grana - je lokalna grana na kojoj se radno stablo trenutno nalazi (HEAD pokazuje na ovu granu).
- origin/master i origin/udaljena_grana - su remote tracking grane udaljenog repozitorijuma origin.
- master - je lokalna grana koja prati origin/master, odnosno ona je tracking grana.

Grane i operacije:



Kreiranje grane se može odraditi sa više operacija. Kreiranje nezavisne lokalne grane:

git branch mojabrana

Kreiranje grane koja prati remote tracking granu pri čemu je udaljena grana u lokalnom repozitorijumu predstavljena kao <udaljeni repo>/<ime grane>, gde je udaljeni repo najčešće sa nazivom origin (origin/udaljenagrana):

git branch -t origin/udaljenagrana

Prethodni primer će kreirati tracking granu koja se zove kao i udaljena. Ako želimo da kreiramo tracking granu koja se zove drugačije, radimo:

git branch mojabrana2 -t origin/udaljenagrana

Informacije o granama je moguće videti sledećim komandama:

```
# Spisak lokalnih grana, gde je sa * označena
tekuća
```

```
$ git branch
```

```
# Prikaz svih lokalnih i remote tracking grana
```

```
$ git branch -a
```

```
# Spisak svih remote tracking
```

```
# Spisak lokalnih grana i njihovih upstream grana  
$ git branch -vv
```

Promena grane podrazumeva izmenu radnog stabla tako da odgovara verziji sa grane. Grana se menja komandom na sledeći način:

```
git checkout moja_grana
```

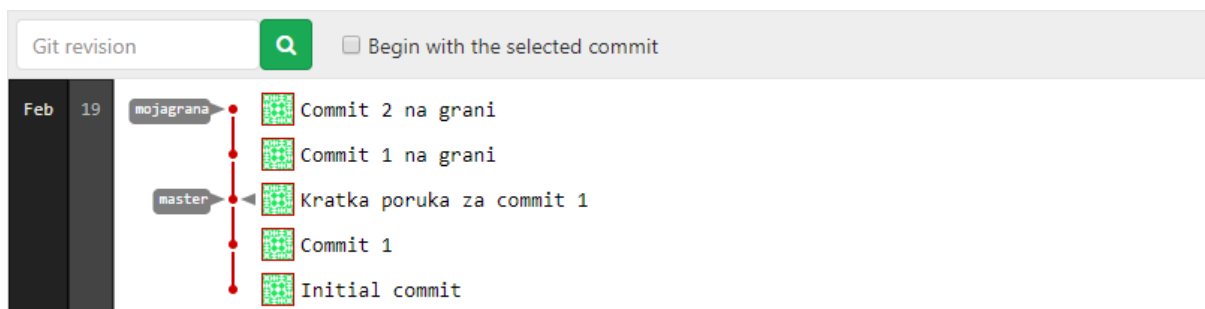
Moguće je kreirati granu i odmah se prebaciti na istu komandom:

```
git checkout -b moja_grana
```

Nakon sređivanja i commit-ovanja, neophodno je poslati promene na udaljeni repozitorijum. Objavljivanje, odnosno kreiranje grane na udaljenom repozitorijumu se radi na sledeći način:

```
$ git push origin moja_grana  
  
# Sa podešavanjem upstream grane  
$ git push -u origin moja_grana
```

Izgled commit-ova na udaljenom repozitorijumu (Graph):



3.3.4 Spajanje grana

Nakon određenog vremena rada na grani, neophodno je izvršiti spajanje grane sa udaljenog repozitorijuma. Komanda za to je:

```
git pull
```

pri čemu se vrši automatsko spajanje grane, s tim što je moguće raditi spajanje i eksplicitno, pri čemu je bitno pozicionirati se na granu na koju se vrši spajanje:

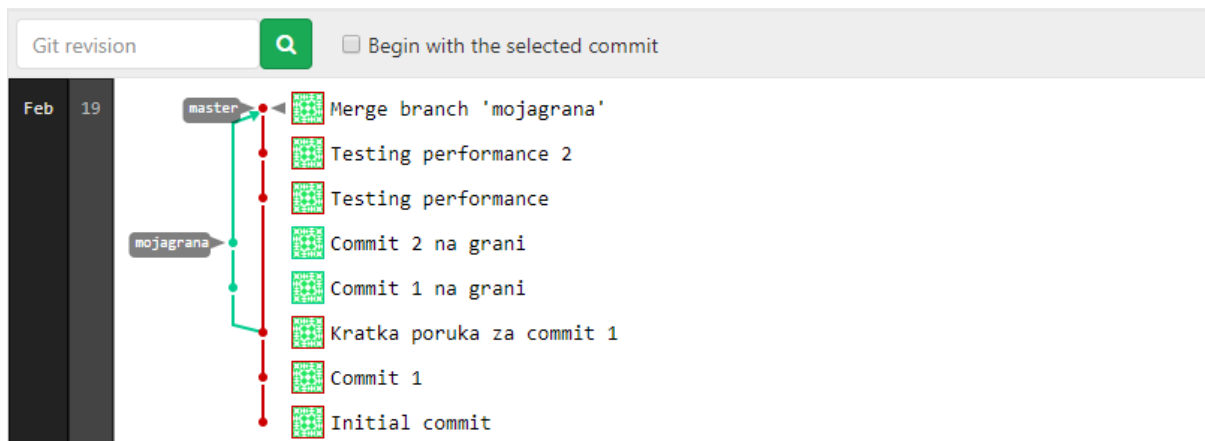
```
git checkout master
```

```
git fetch
```

```
git merge moja_grana
```

Komanda **fetch** će pokupiti promene sa udaljenog repozitorijuma ali neće automatski obaviti spajanje na tekuću tracking granu. Komanda **merge** radi spajanje i ako nema konflikata, automatski radi **commit**. Ako ima konflikta, neophodno ga je razrešiti ručno, zatim dodati promenu i izvršiti **commit**.

Izgled grafa na GitLab-u nakon spajanja grana:



3.3.5 Razrešavanje konflikata

U idealnom scenariju, prilikom spajanja lokalnih i udaljenih grana, više članova tima neće dirati iste segmente koda. Međutim, vrlo su česti slučajevi kada dolazi do konflikata, odnosno kada su menjane iste linije koda, kada je obrisan fajl koji se u nekom commit-u i dalje nalazi u projektu itd. Git u tim situacijama neće znati šta treba da radi, odnosno, prepušta korisniku apsolutnu kontrolu nad tim delovima koje je neophodno ispraviti. U datoteci, biće označeni kritični delovi. Biće navedeno šta je bio sadržaj lokalnog commit-a, a šta sadržaj nekog commit-a na remote repository-u. Korisnikov zadatak je da edituje fajl u skladu sa tim šta bi sadržaj na toj liniji trebalo da bude, vodeći računa da ne poremeti funkcionalnost projekta.

Primer izgleda fajla sa konfliktima:

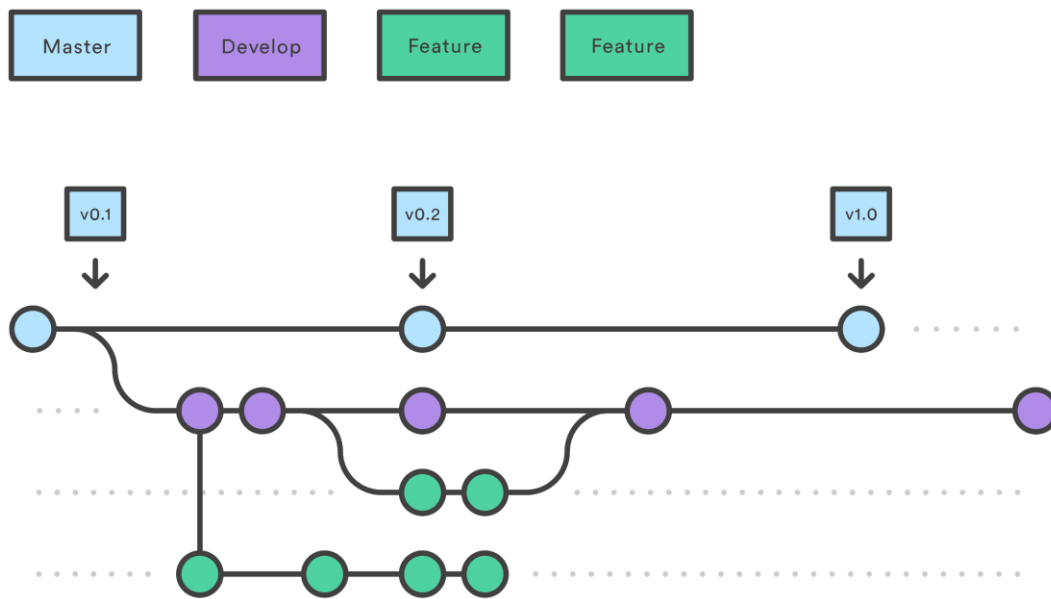
```

1 my_dict = {
2     'Izuku Midoriya': 'One for All',
3     'Katsuki Bakugo': 'Explosion',
4     'All Might': 'One for All',
5     'Ochaco Uraraka': 'Zero Gravity'
6 }
7 <<<<<< HEAD
8   'Gosho Aoyama' : 'Detektiv Conan'
9   =====
10  'Hayato Date' : 'Naruto'
11  >>>>>> f76a27f0f65572043657ba7c78e04e02e7be091c
12 }
```

Nakon razrešavanja svih konflikata, neophodno je stage-ovati sve promene i odraditi regularni commit.

4. Modeli grananja

Postoji više konvencija kako da se definišu grane i više procesa koji opisuju način njihovog korišćenja. Najtrivijalniji način jeste korišćenje samo **master** grane, što se može upotrebiti za vrlo male timove i relativno male projekte. Preporučeno i dovoljno koristiti na projektu je definisanje jedne **develop** grane i veći broj **feature** grana na kojima će biti realizovane funkcionalnosti i one će se spajati na **develop** granu. Spajanje sa **develop** grane na **master** granu raditi kada se ima **stable** verzija koja je skroz funkcionalna. Na **master** grani **mora biti verzija koja radi!**



Vše informacija na sledeća dva linka:

- <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>
- <https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow>

5. Zadatak

Registrovati se na GitLab i napraviti jedan testni repozitorijum u koji će biti dodati ostali članovi tima. Istestirati napomenute komande i dublje analizirati dešavanja radi boljeg razumevanja načina funkcionisanja Git-a, što će olakšati dalju kolaboraciju među članovima.