Sveučilište u Mostaru
Fakultet strojarstva i računarstva

Programsko inženjerstvo – vježbe

Asistent: Davor Škobić

1. Osnove rada sa GIT-om

Osnovni pojmovi i procedure:

- Clone, pull, push, fetch, checkout, add, reset
- Razrješavanje konflikata, grananje i spajanje.

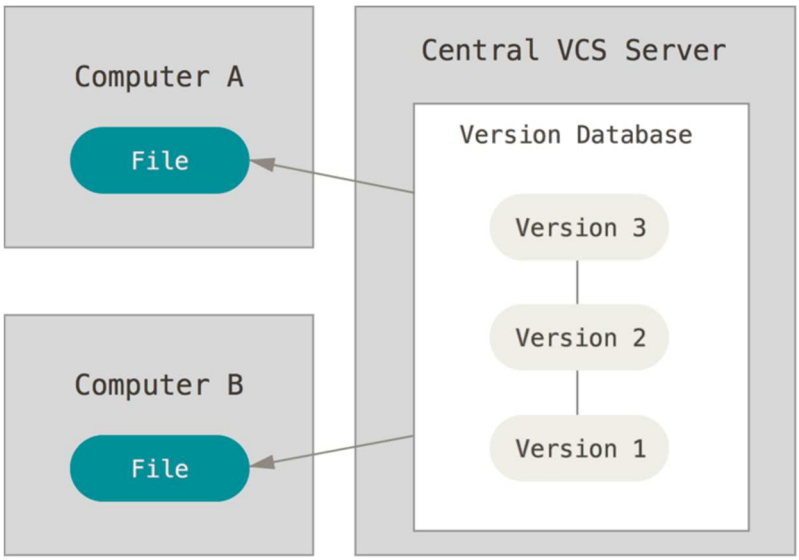
1.1. Što je GIT?

U svojoj osnovnoj definiciji GIT je alat za kontrolu verzije (verzioniranje) datoteka u projektima razvoja programske podrške (softvera). Svaki razvojni projekt u svojoj osnovi sastoji se od programskih datoteka čijim izvršenjem omogućavamo izvršavanje određenih zadataka koji su potrebni za funkcioniranje određenog segmenta informacijskog sustava. Prilikom razvoja programske podrške prikupljaju se korisnički zahtjevi (koji se mogu mijenjati), implementiraju zamišljena rješenja, te u tom procesu sudjeluje veći broj ljudi.

Kako bi veći broj ljudi mogao efikasno sudjelovati u razvoj programske podrške, osmišljeni su alati za verzioniranje programskog koda. Sa tim alatima omogućva se povijesno čuvanje programskog koda, te se na jednostavan način programski kod dijeli, vraća na prethodno stanje, te grana u različitim pravcima razvoja.

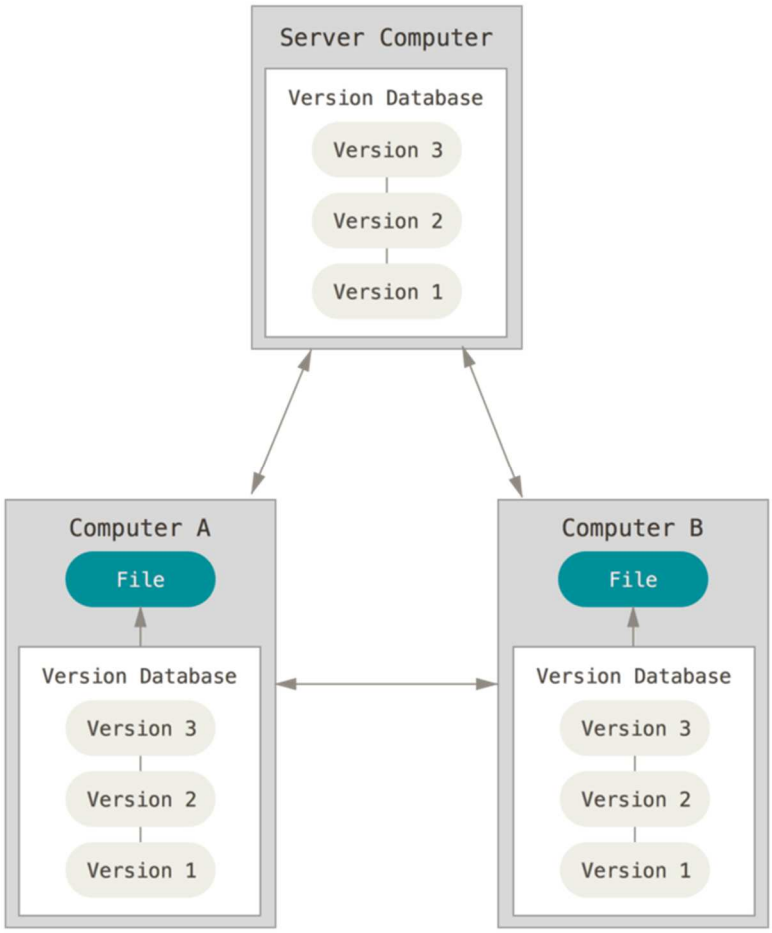
U osnovi alate za kontrolu verzije dijelom u dva segmenta:

- Centralizirani sustavi za kontrolu verzije – npr. *Subversion* (SVN) i *Team Foundation Server* (TFS),
- Distribuirani sustavi za kontrolu verzije – npr. *Git*.



Slika 1 Centralizirani sustav za kontrolu verzije

Centralizirani sustav za kontrolu verzije je sustav koji se i danas primjenjuje sa prednošću što svi korisnici u određenom trenutku imaju uvid u rad drugih korisnika (onaj dio koji je postavljen na centralni poslužitelj). Administratori imaju jednostavniju kontrolu nad cijelim sustavom u smislu jednostavnije autorizacije korisnika. Kao nedostatke možemo istaknuti jedinstvenu točku kvara (*single point of failure*), u slučaju kvara centralnog poslužitelja dolazi do gubitka podataka i pristupa programskom kodu ranijih verzija, te korisnicima ostaju samo trenutne verzije spremljene na njihovim radnim jedinicama.



Slika 2 Distribuirani sustav za kontrolu verzije

Distribuirani sustavi se ističu time što na klijentskim radnim jedinicama preslikavaju cijeli središnji repozitorij. Tako u slučaju gubitka središnjeg poslužitelja, trenutna i sve prethodne verzije programskog koda su sačuvane na klijentskim uređajima. Procesom kloniranja, bilo koji od klijentskih repozitorija može se vratiti na središnji poslužitelj. Glavni ciljevi kod razvoja Git-a kao distribuirano rješenja bili su: brzina, jednostavan dizajn, podrška za nelinearni razvoj programske podrške (paralelne grane razvoja), potpun distribucija, mogućnost efikasnog rukovanja velikim projektima.

Zbog nepostojanja pravog centralnog poslužitelja, nije moguće voditi numeraciju iteracija programskih datoteka (npr. 1,2,3,4...), već se za svaki *commit* datoteke na lokalni poslužitelj predaje kopija cijele datoteke, te se radi izračun *hash* vrijednosti za tu datoteku koristeći SHA-1 algoritam. Na taj način se osigurava integritet podataka i jedinstvenost zapisa.

1.2. Git – osnovne naredbe i pojmovi

Osnove naredbe sa GIT-om su:

- Clone
- Push
- Pull
- Fetch
- Checkout
- Add
- Reset

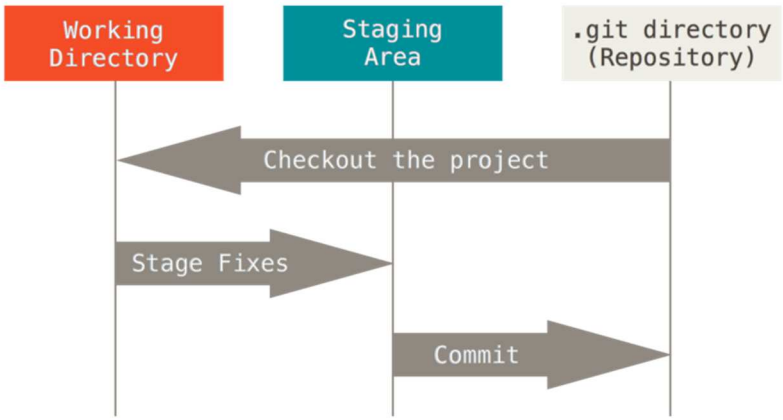
Također, potrebno je posebnu pažnju posvetiti sljedećim procesima:

- Razrješavanje konflikata,
- Grananje i spajanje

Prije objašnjenja ovih pojmova potrebno je razumjeti tri glavna stanja u svim Git projektima:

- **Commited**
- **Modified**,
- **Staged**.

Commited označava da su podaci sigurno spremljeni u lokalnoj bazi podataka. *Modified* znači da je datoteka izmjenjena, ali da nije izvršen *commit* u bazu podataka. *Staged* označava datoteku koja je izmijenjena, te je označena za spremanje u bazu podataka kod idućeg spremanja u lokalnu bazu podatka (*commit*).



Slika 3 Tri stanja datoteke

Git direktorij (repozitorij) je lokacija gdje se nalaze svi podaci o našem projektu koje preuzimamo sa centralnog repozitorija (naredba *clone*).

Working directory (tree) je radni prostor u kojemu se nalazi otvoren jedan *checkout* ili bolje reći jedna grana našeg projekta na kojem radimo. Ti podaci se izvlače iz kompresirane baze podataka u Git direktoriju, te spremaju se na lokalni disk kako bi se mogli uređivati.

Staging area je datoteka koja se generalno nalazi u Git direktoriju i sprema informacije o tome što će se naći u idućoj *commit* naredbi.

Tok poslova kod Git-a izgleda ovako:

1. Izvršite izmjenu u vašem radnom direktoriju.
2. Izvršite *stage* datoteka.
3. Izvršite *commit* datoteka iz *stage* prostora i pohranjujete ih u vaš Git direktorij.

Za rad sa Git repozitorijama koriste se već nabrojane osnovne naredbe. Svaka od tih naredbi ima svoje podnaredbe koje ćete upoznati tokom korištenja ovog alata.

Clone naredbu koristimo u slučajevima kada želimo preuzeti postojeći repozitorij sa poslužitelja. Kao što je već opisano, kod distribuiranih sustava kao što je Git, preuzima se cjelokupni sadržaj poslužitelja, ne samo trenutna verzija projekta, nego cijela povijest tog repozitorija od samog njegovog početka, što nam daje mogućnost lokalnog rada na projektu bez kontaktiranja središnjeg poslužitelja.

Fetch naredbu koristimo u slučaju kada sa poslužitelja želimo preuzeti sve one datoteke koje nedostaju na našem lokalnom repozitoriju (preuzimanje svih datoteka, *branch*-ova), ali se u tom trenutku ne vrši spajanje (***merge***) sa trenutnim projektom.

Ukoliko želimo da prilikom preuzimanja novih ažuriranja sa centralnog poslužitelja uradimo automatsko spajanje (***merge***) projekta, to činimo sa naredbom **pull**.

U trenutku kada uradimo određenu funkcionalnost, te je želimo objaviti na centralnom poslužitelju koristimo naredbu **push**. Ukoliko je od našeg preuzimanja projekta (*clone*) netko drugi izvršio *push* naredbu na taj projekt, tada će naša *push* naredba biti odbijena. Prvo moramo preuzeti posljednje promjene na projektu, spojiti ih sa našim projektom, te tek onda možemo uraditi *push*.

Checkout naredbu koristimo kada želimo prebaciti se sa jedne grane na drugu. Svaki projekt može imati više grana razvoja, o čemu će kasnije biti riječ.

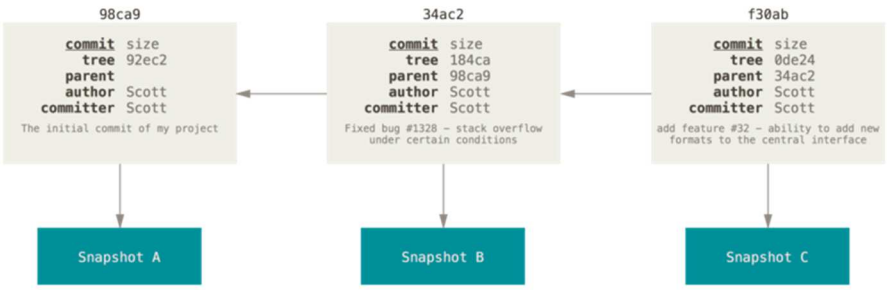
Kada ne želimo postaviti projekt na repozitorij, ali želimo sačuvati promjene kako bi bile spremne za iduću *commit* koristimo naredbu **add**.

U slučaju da želimo vratiti projekt na repozitoriju na prethodno stanje, koristimo naredbu **reset**.

//opisati ostatak naredbi i koncepte grananja, spajanja i razrješavanja konflikata.

Grananje u Git-u je svojstvo koje sadrže svi sustavi za kontrolu verzije koda, no Git se izdvaja jednostavnošću vlastitog rješenja iz tog područja, te se korištenje ove funkcionalnosti ohrabruje u svrhu olakšanja rada korisnicima.

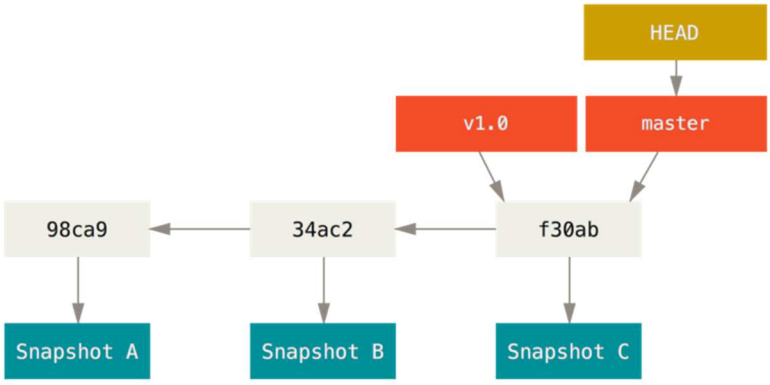
U Git sustavu ne čuvaju se samo promjene u odnosu na izvorišnu datoteku, već se radi na čuvanju cjelokupne snimke određenog sustava. Kada uradimo *commit* određenih promjena Git stvara novi pokazivač koji pokazuje na skup izmjenjenih datoteka, te na pokazivač na prethodni snimak tih datoteka.



Slika 4 Commit pokazivači

Kada govorimo o grananju, govorimo o promjeni toka u razvoju programskog rješenja, u primjerima gdje želimi pokušati riješiti nešto na drugačiji način, testirati određenu funkcionalnost koja ne mora ući u konačno rješenje ili raditi na ispravci određene pogreške.

Kao što je pojedini commit sačuvan kao pokazivač na određene elemente, na isti način su implementirane grane.



Slika 5 Grana i commit povijest

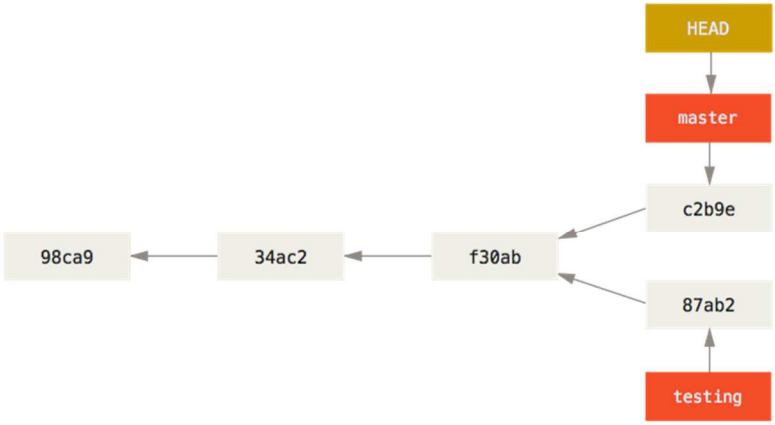
Na slici 5 vidimo povijest *commit* naredbi, koji se definiraju kao serija pokazivača. *Master* je oznaka za glavnu granu svakog programa, dok *v1.0* je oznaka (*tag*) koju možemo koristiti za određivanje određenih važnijih točaka u razvoj programske podrške. HEAD oznaka je pokazivač na granu koja je trenutno u uporabi.

Narebom *branch* stvaramo novu granu u razvoju našeg programskog rješenja.

Slika 6 Grane i HEAD pokazivač

Na slici 6 prikazan je primjer određenog slijeda promjena na programskom kodu, sa stvorene dvije grane, *master* koja ima ulogu glavne grane, te grane *testing* koju koristimo za određene promjene na programskom rješenju.

Naredbom *checkout* prelazimo sa grane *master* na granu *testing* (git checkout testing). Bilo koji *commit* koji napravimo nadalje u programskom rješenju vezat će se za onu granu u kojoj se trenutno nalazimo.

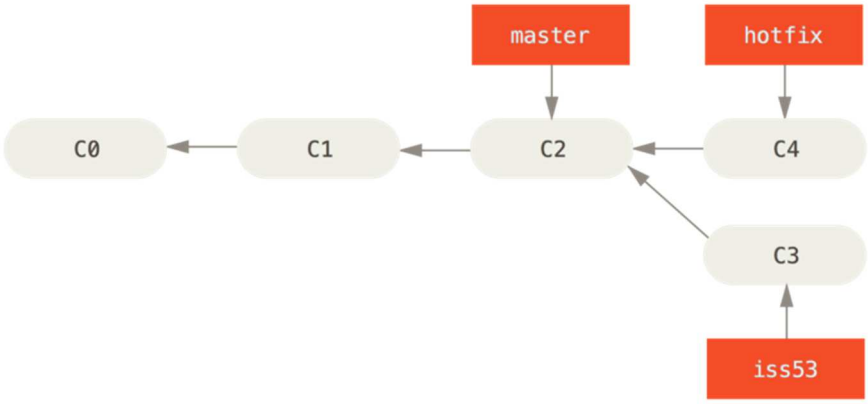


Slika 7 Primjer grananja - dvije grane

Na slici 7 možemo vidjeti situaciju gdje su urađene određene promjene u obe postojeće grane našeg programskog rješenja. Kako bi mogli implementirati neko krajnje rješenje Git omogućava povezivanje više grana u jednu točku.

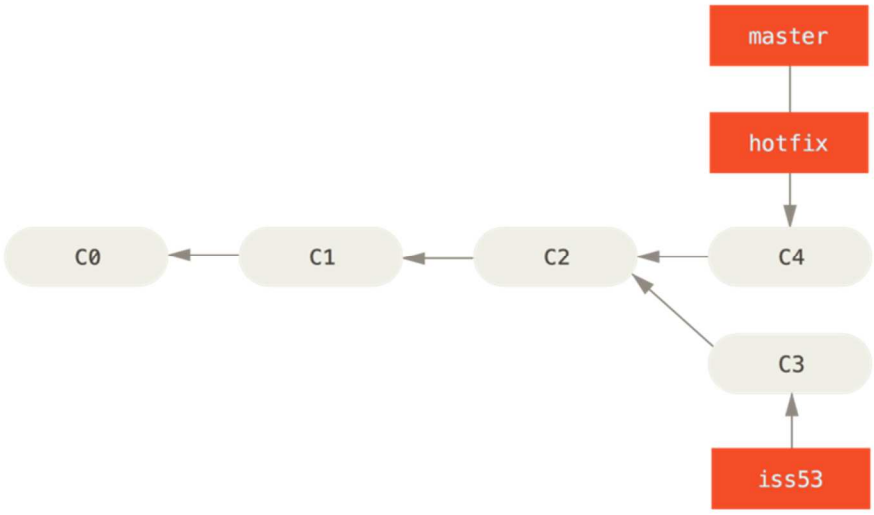
Spajanje grana (*merging*) je bitan proces u upravljanju programskim kodom. Sami proces se razlikuje od situacije do situacije, te može biti jednostavan, a i kompleksan.

Jednostavniji primjer spajanja dvije grane u konačno rješenje je primjer *fast-forward*.



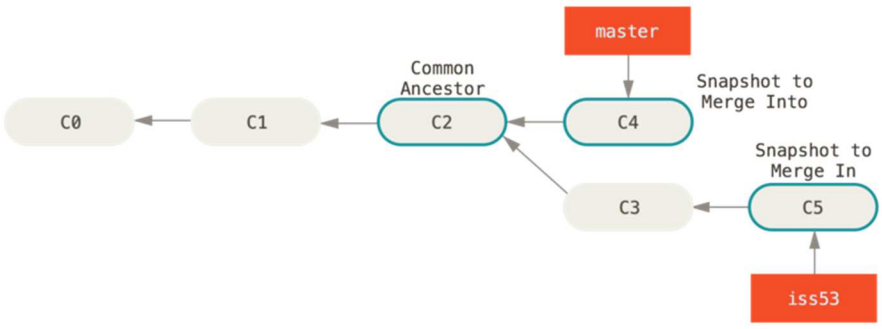
Slika 8 Jednostavno grananje – 1

Na slici 8 imamo primjer glavne grane *master* iz koje si niknule dvije grane *hotfix* i *iss53*. U navedenom slučaju otkrivene su dvije pogreške, *hotfix* koja se odnosi na problem u radu programske podrške, te se taj problem mora brzo riješiti kako bi aplikacija nesmetano radila, dok *iss53* je veći problem čije rješenje nije žurno. Kada u *commit* C4 riješimo aktualni problem, potrebno je izvršiti spajanje grana *master* i *hotfix*. Spajanje vršimo tako što se vratimo u glavnu granu (*git checkout master*), te naredbom *git merge hotfix* izvršavamo spajanje te dvije grane. Git automatski određuje način na koji je najbolje spojiti te dvije grane, te ovdje primjenjuje *fast-forward* metodu gdje jednostavno pokazivač grane *master* pomiče sa *commit* C2 na C4.

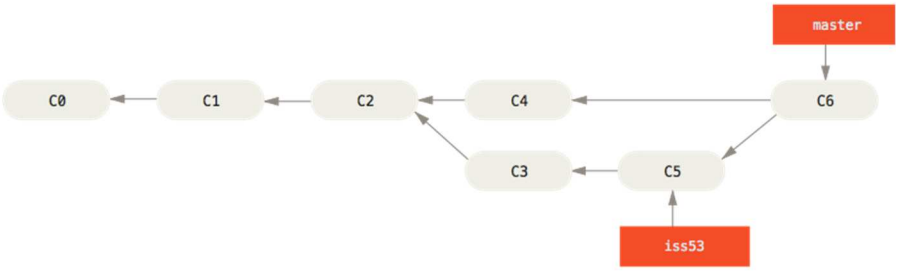


Slika 9 Jednostavno grananje – 2

Kao što vidimo na slici 9, dobijemo sljedeću situaciju. Pošto nam grana *hotfix* više nije potrebna, možemo je obrisati.



Slika 10 Jednostavno spajanje - 1



Slika 11 Jednostavno spajanje – 2

Na slikama 10 i 11 vidimo da je došlo do konačnog rješenje problema u grani *iss53*, te da je isto rješenje potrebno spojiti na glavnu (*master*) granu. Kod ovog slučaja radi se na tome da se generira novi *commit*

koji će objediniti zajedničkog pretka (C2), vrh grane master (C4) i vrh grane iss53(C5). Rezultat ovog procesa je commit C6 na kojeg pokazuje *master* grana.

Problem se može pojaviti ukoliko je došlo do izmjena na istim dijelovima datoteka u ova tri promatrana *commit*-a. Git neće izvršiti spajanje, nego će ponuditi načine rješavanja konflikata. Konflikt se može riješiti na način da odaberemo jedno od rješenja ili da ručno odaberemo dijelove datoteka iz svakog od promatranih *commit*-ova.

Ovdje su objašnjene osnovni pojmovi vezani za Git i rad sa njime, koji su potrebni za određeni početak rada sa ovim alatom. Opcije ovog sustava su veće, te je njegova primjena rasprostranjena. U izradi svojih projektnih zadataka ćete koristiti ovaj sustav za kontrolu verzije i predaju rada.

1.3. Primjeri rada sa Git

Za rad sa sustavom Git, potrebno je instalirati Git za onaj operacijski sustav koji koristite (https://git-scm.com/downloads).

Bilo koji folder na našem računalnom sustavu može poslužiti kao repozitorij za naš projekt. U ovom slučaju kroz komandnu konzolu napravit ćemo naš početni repozitorij.

```
$ mkdir test-repozitorij
$ cd test-repozitorij/
$ git init
Initialized empty Git repository in
C:/Users/davor.skobic/Documents/Fakultet/Nastava/Programsko
inženjerstvo/Repozitoriji/test-repozitorij/.git/
davor.skobic@DESKTOP-CK9NDC0 MINGW64 ~/Documents/Fakultet/Nastava/Programsko
inženjerstvo/Repozitoriji/test-repozitorij (master)
```

Naredbom **git init** inicijalizirali smo naš početni repozitorij.

Prilikom rada u Git sustavu, potrebno je učiniti određenu početnu konfiguraciju koja se odnosi na naš identitet, kako bi priliko predaje izvornog koda bilo poznato tko je to učinio. Konfiguracije koje radimo mogu biti globalne (vezane za sve repozitorije na našem sustavu) ili lokalne (vezane za određeni repozitorij. Sve naredbe u Git sustavu imaju sljedeću strukturu **git <naredba> <opcija> <opcija>**.

```
$ git config --global user.name "Davor Škobić"
$ git config --global user.email "davor.skobic@outlook.com"
```

Globalnom naredbom **–global** podesili smo naziv i email korisnika koji će se evidentirati kod svakog našeg zapisa u repozitorij. Također, ovu istu naredbu je moguće pokrenuti za svaki repozitorij posebno bez oznake global, tako da za svaki repozitorij možemo unijeti druge podatke (ukoliko se radi o privatnom, poslovnom ili drugom repozitoriju).

Sa naredbom **git status** vršimo provjeru stanja u našem repozitoriju. Trenutno, nemamo nikavih promjena u istom te stoga dobijemo sljedeću poruku.

```
$ git status
On branch master
Initial commit
nothing to commit (create/copy files and use "git add" to track)
```

Nakon dodavanja obične tekstualne datoteke u naš folder, ponovnim pokretanjem iste naredbe, dobijemo sljedeću poruku.

```
$ git status
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        datoteka.txt
nothing added to commit but untracked files present (use "git add" to track)
```

Na osnovu povratne informacije koju smo dobili, naredbom **git add datotetka.txt** uključujemo predmetnu datoteku u praćenje na našem repozitoriju, te sve promjene na njoj možemo **commit**-ati u naš lokalni repozitorij.

```
$ git add datoteka.txt
$ git status
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   datoteka.txt
```

Uključivanjem određene datoteka u praćenje, mi je dodajemo u *index* područje ili *staging area*. U trenutku kada je datoteka u tom području, ona će se prenijeti u lokalni repozitorij kod izvršenja naredbe *commit*. Ukoliko izvršimo promjenu na predmetnoj datoteci, te ponovno izvršimo naredbu *git status* dobit ćemo sljedeći ispis.

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   datoteka.txt

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

        modified:   datoteka.txt
```

Git nam poručuje da je prvo stanje datoteke spremno za *commit* na repozitorij, ali da postoji druga verzija iste datoteke, te ukoliko želimo istu pripremiti za repozitorij, moramo je pozvati naredbom *git add* ili je zanemariti naredbom *git checkout*.

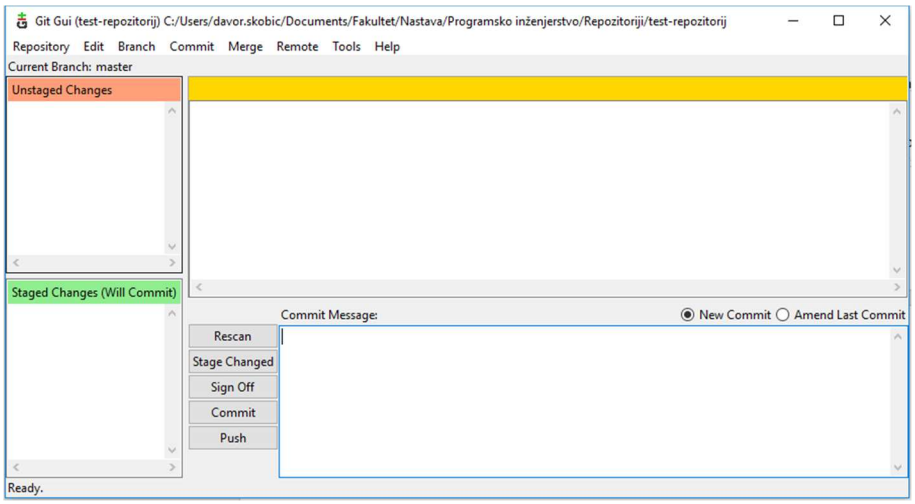
Kada odlučimo da je vrijeme za postavljanje datoteka našeg projekta na repozitoriji isto činimo sa naredbom **git commit –m „komentar“**. Svako postavljanje u git repozitorij traži komentar.

```
$ git commit -m "Prvi commit"
[master (root-commit) 824a58c] Prvi commit
 1 file changed, 5 insertions(+)
 create mode 100644 datoteka.txt
```

Također, važne naredbe su:

- git rm --cached test.pyc – uklanjanje datoteka iz repozitorija, npr. datoteke koje se slučajno ubačene, u ovom slučaju radi se o datoteci test.pyc
- git log – pregled povijesti svih *commit-ova*
- git commit --amend -m "Nova verzija, promijenjen datoteka.txt" – ukoliko smo nakon commit naredbe shvatili da su potrebne dodatne izmjene u posljednjem commitu (kada želimo izbjeći potrebu za dva commit koraka, vezano za jednu logičku izmjenu.

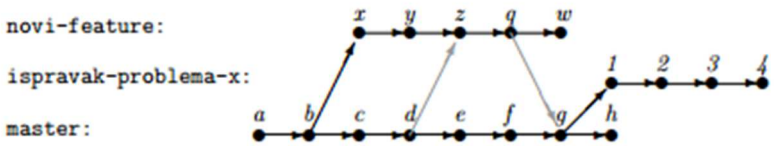
- git gui – naredba koja otvara grafičko sučelje, čime nam je olakšano upravljanje našim repozitorijem.



Slika 12 git-gui

- git clean – naredba za čišćenje datoteka iz direktorija koji nisu trenutno dio repozitorija.

Grananje kao svojstvo je jedna od velikih prednosti Git sustava.



Slika 13 Grananje

Ono što je ovdje važno još jednom spomenuti je sljedeće: svaki je čvor grafa stanje projekta u nekom trenutku njegove povijesti. Svaka strelica iz jednog u drugi čvor izmjena je koju je programer napravio i snimio u nadi da će dovesti do željenog ponašanja aplikacije.

Sa naredbom **git branch** dobijemo prikaz trenutnih grana koje imamo u svom repozitoriju, oznaku u kojoj se trenutno nalazimo (*).

```
$ git branch
* master
```

Dodavanje nove grane radimo sljedećom naredbom.

```
$ git branch probna-grana
```

Ponovnom provjerom statusa grananja dobijemo sljedeći ispis.

```
$ git branch
* master
  probna-grana
```

Prebacivanje sa grane na grane radimo naredbom *checkout*.

```
$ git checkout probna-grana
Switched to branch 'probna-grana'
```

Zapamtite, najbolje je prebacivati se s grane na granu tek nakon što smo commitali sve izmjene. Tako će nas u novoj grani dočekati čista situacija, a ne datoteke koje smo izmijenili dok smo radili na prethodnoj grani.

Brisanje grana je omogućeno naredbom *git branch –D <naziv grane>*.

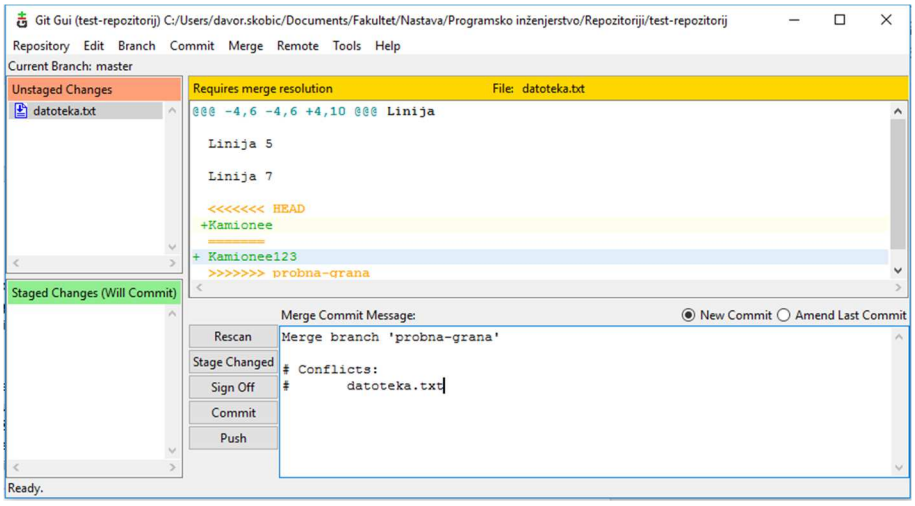
Spajanje grana je jedna od bitnijih stavki kontrole verzije. Grananje koristimo u raznim slučajevima kao što su ispravke bugova, razvoj dodatnih funkcionalnosti, testiranje različitih filozofija razvoja određenih segmenata podrške i sl. Kod procesa spajanja grana radi se o preuzimanju promjena iz jedne grane u drugu. Sve grane nastavljaju svoj životni tok. Kada želimo spojiti dvije grane, pozicioniramo se u onu granu u kojoj želimo izvršiti spajanje. Npr. kada želimo spojiti promjene iz *probna-grana* u našu *master* granu, potrebno se vratiti u *master* granu sa *checkout* naredbom. Nakon toga potrebno je izvršiti *merge* naredbu.

```
$ git merge probna-grana
Updating 9818f0f..3721ce8
Fast-forward
 datoteka.txt | 2 ++
 1 file changed, 2 insertions(+)
```

Razrješavanje konflikata je jedna od čestih pojava prilikom spajanja grana. U slučaju kada u dvijema granama izvršimo izmjenu na istoj liniji koda, Git neće znati automatski spojiti datoteke.

```
$ git merge probna-grana
Auto-merging datoteka.txt
CONFLICT (content): Merge conflict in datoteka.txt
Automatic merge failed; fix conflicts and then commit the result.
```

U ovim situacijima koristite **git-gui** naredbu za rješavanje problema ručnim spajanjem kodova.



Slika 14 git merge

Merge metode:

- *fast-forward,*
- *rebase,*
- *cherry-pick,*
- *squash.*

U procesu razvoja možete koristiti **tag** kao metodu obilježavanja verzija koda.

Preuzimanje udaljenog repozitorija se naziva kloniranje, te se ono radi sljedećom naredbom.

```
$ git clone git://github.com/tkrajina/uvod-u-git.git
```

Na ovaj način preuzimamo trenutno stanje projekta. Ukoliko je došlo do određenih *commitova* od strane drugih korisnika na udaljeni repozitorij, iste preuzimamo ***fetch*** naredbom.

```
git fetch
remote: Counting objects: 5678, done.
remote: Compressing objects: 100% (1967/1967), done.
remote: Total 5434 (delta 3883), reused 4967 (delta 3465)
Receiving objects: 100% (5434/5434), 1.86 MiB | 561 KiB/s, done.
Resolving deltas: 100% (3883/3883), completed with 120 local objects.
From git://github.com/twitter/bootstrap
```

U grupnoj vježbi pokušati uraditi commit na udaljeni repozitorij na kojem je bio commit nekog drugog korisnika. (Primjer sa git fetch, git merge origin/master).

Ukoliko želimo postaviti sadržaj na udaljeni repozitorij koristimo ***git push origin master***. Ukoliko nemamo ovlasti, isto radimo sa ***git pull request,*** gdje od administratora udaljenog repozitorija tražimo da povuče naše izmjene.

Ovdje su objašnjeni osnovni procesi. U navedenoj literaturi možete pronaći opširnije primjere koji vam mogu biti potrebni kod izrade vaših projektnih zadataka.

Literatura – Git

[1] - https://git-scm.com/book/en/v2 - Sve o Git

[2] - https://www.youtube.com/watch?v=egy2r6ReaeI – Using Git in Visual Studio – objašnjenja Git-a i primjeri korištenja unutar Visual Studio

[3] – https://tkrajina.github.io/uvod-u-git/git.pdf - Uvod u Git, Tomo Krajina