Git

U kojoj god firmi da budeš radio, čekaće te git. Ako pak budeš radio u nekoj firmi u kojoj ne rade git, zapitaj se da li želiš da radiš tu.

prof. dr. Igor Dejanović

Šta je git objašnjeno na jednostavnom primjeru

Jedna od brojnih mogućnosti koje ti git nudi jeste istorija izmjena. Zamisli neki editor, bilo koji. Svaki editor ima ugrađenu mogućnost „*undo*“ pomoću koje možeš da poništiš prethodnu akciju. Većina editora ti omogućuju da poništiš i više prethodnih akcija i vratiš se u prošlost. Git ti nudi istu stvar ali u kodu. Razlika između git-a i editora je to što editor za tebe pamti akcije koje su se desile, a git ne pamti, već ti sam praviš svoje verzije. Verzija bi bila neko stanje projekta koje želiš da sačuvaš u istoriji, na koje se možda želiš vratiti u budućnosti.

Druga bitna stvar vezana za git je praćene sadržaja. Rekli smo iznad da git za tebe ne pamti akcije koje su se desile. Međutim, git zato prati sadržaj i prikazuje ti sve promjene koje su se desile. Npr. u editoru si dodao novu liniju koda u jedom fajlu, dodao si 10 linija u drugom i napravio novi fajl. Git će ti prikazati sve ove izmjene, gdje ti onda od ovih izmjena možeš napraviti novu verziju. Novu verziju ne moraš praviti od svih izmjena, možeš da odabereš neki podskup. Kad praviš novu verziju potrebno je da opišeš zašto si napravio te izmjene, kako bi kasnije kad bi se htio vratiti na nju lako mogao naći koja ti verzija treba.

Napomena: Ovo objašnjenje, ali i samo uputstvo, je veoma pojednostavljeno. Git ima dosta više mogućnosti od ovdje navedenih. Naredbe i komande su objašnjene tek toliko koliko ti je potrebno da kreneš sa radom. Git ima odličnu dokumentaciju koja će biti linkovana na svim mjestima. Toplo preporučujem literaturu koja je navedena pri kraju dokumenta. Ideja ovog dokumenta je da te uvede u priču o gitu, a kasnije ćeš sam širiti svoje znanje i git mastery.

Uvod

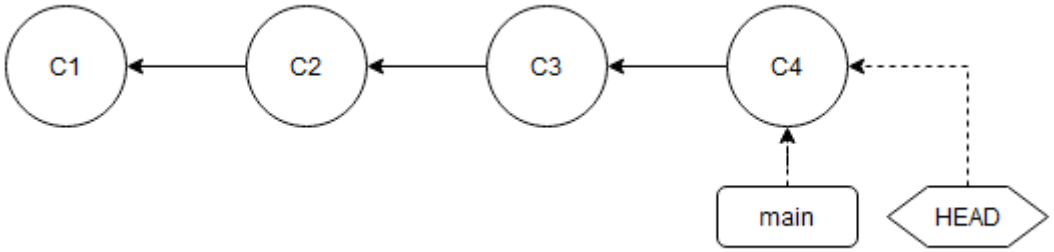
Git je distribuirani sistem za kontrolu verzija. Prednosti:

- brz i skalabilan
- jednostavan dizajn
- odlična podrška za paralelan rad
- ne prati fajlove već sadržaj

Git istoriju modeluje pomoću usmjerenog acikličnog grafa sa promjenama. Možemo se kretati kroz graf, širiti ga, granati, itd. Svaki čvor u grafu predstavlja jedan commit.

Da za početak pojednostavimo priču, commit možeš posmatrati kao jednu verziju tvog projekta. Git je napravljen tako da kad se krećeš između commit-ova izgleda kao da se krećeš između verzija tvog projekta. Ipak, ispod haube ovo nije tako. U paragrafu ispod imaš kratko objašnjenje kako on zapravo radi.

Za one koji žele da znaju više: Commit predstavlja samo skup izmjena. Da bi dobio verziju tvog projekta u nekom commit-u, potrebno je da se primjene svi commit-i (svi skupovi izmjena) od prvog (inicijalnog commit-a koji predstavlja početno stanje tvog projekta) do trenutnog commit-a i tek tada dobiješ trenutno stanje.



Slika 1. Primjer jednog git grafa

Na slici iznad su krugovima sa oznakama C1, C2, C3 i C4 označeni commit-ovi. Commit predstavlja skup izmjena. C1 je inicijalni commit i on predstavlja početno stanje projekta. Svaki commit ima referencu na svog prethodnika i tako dobijemo graf. Grane su na slici označene sa pravougaonicima sa zaobljenim ivicama i unutar njih su imena grana. Na slici trenutno ima samo jedna grana `main`. Pored commit-a i grana na slici možemo uočiti i `HEAD` koji se nalazi u dijamantu. `HEAD` nam govori gdje se trenutno nalazimo unutar grafa (u ovom slučaju nalazimo se na commit-u C4 na grani `main`).

Spomenuli smo grane, pa je red i da ih kratko objasnimo. Grane su tokovi razvoja. Možeš ih pojednostavljeno posmatrati kao zasebne kopije tvog projekta. Kad radiš na jednoj grani, ostale kopije (grane) su netaknute. Možeš lako da se prebaciš sa jedne grane na drugu i počneš razvoj nove funkcionalnosti, itd.

Mogućnosti grana se najbolje ogledaju u timskom radu. **Uz pomoć grana svi članovi tima mogu zajedno da rade na istom projektu u isto vrijeme**. Ovo je jedna od velikih prednosti korišćenja gita. Grane ti omogućuju da paralelno radiš na projektu sa tvojim članovima tima, možeš da šalješ svoje grane (posao koji si odradio) drugim članovima tima, oni mogu da nastave rad gdje si stao. Po vrh svega, možeš da spajaš grane i tako integrišeš posao koji je razvijan paralelno u jednu granu koja će onda sadržati sve izmjene.

Instalacija

Git možeš preuzeti na: https://git-scm.com/downloads

Instalacija je prilično jednostavna, savjet je da pažljivo pratiš korake instalacije jer tu biraš podrazumijevani editor za rad sa git-om. Inicijalna vrijednost je Vim i preporuka je da odabereš neki drugi ukoliko nisi upoznati sa Vim-om. Nije problem ukoliko odabereš pogrešan editor, lako ga možeš zamijeniti kasnije (uputstvo).

Konfiguracija

Git je veoma konfigurabilan. Konfiguracija se može podesiti na globalnom nivou, na nivou korisnika i na nivou projekta. Tako možeš imati različita podešavanja kad radiš na različitim projektima. Za detalje pogledaj https://git-scm.com/docs/git-config.

Par stvari je potrebno da odmah konfigurišeš: (Ukoliko radiš u učionici u kojoj više studenata dijeli isti laptop, ne preporučujem da radiš ovo podešavanje sad, već nakon što napraviš novi git repozitorijum pokreni ovu komandu unutar njega (repozitorijuma) bez flag-a `--global`. Ovako će tvoje ime i email biti podešeni samo za taj tvoj projekat, a ne za cijeli sistem.)

- ime korisnika: `git config --global user.name "Marko Marković"`
- email korisnika: `git config --global user.email marko@example.com`

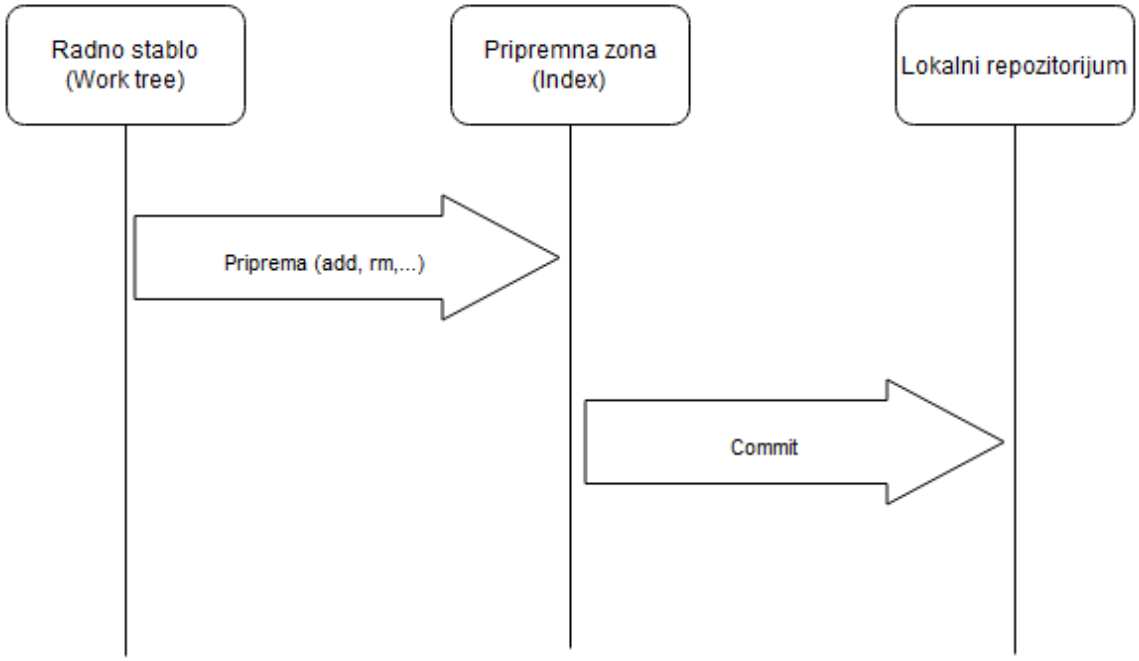
Ime i email koje se ovdje definišu će se koristiti kao ime i email autora commit-ova. Konfiguraciju možeš izlistati sa `git config --list`.

Praćenje sadržaja

Git radi tako što prati promjene svih fajlova unutar repozitorijuma (*radnog stabla*). Ukoliko želimo da git ignoriše neke fajlove potrebno je da napravimo `.gitignore` fajl https://git-scm.com/docs/gitignore.

Kad smo odabrali izmjene koje želimo da spremimo za commit prebacujemo ih u *pripremnu zonu* (*index*).

Kad su svi fajlovi koji idu u commit u *index*-u sa *commit* naredbom pravimo novi commit od fajlova iz *index*-a i smještamo ih u *lokalni repozitorijum*. Ovaj workflow je ilustrativan na slici ispod. Na ovaj način možemo samo dijelove koda da komitujemo i biramo šta od promjena ide u lokalni repozitorijum.



Slika 2. Radno stablo – pripremna zona – lokalni repozitorijum

Praktičan primjer

Sad ćemo na primjeru vrlo jednostavnog kalkulatora objasniti rad sa git-om. Kalkulator prima kao unos string koji je u formatu: "`broj operacija broj`", zatim parsira string, izvrši operaciju i korisniku ispisuje rezultat. Ukoliko korisnik kao unos proslijedi "`kraj`" ili "`exit`" aplikacija se gasi.

- U svom editoru generiši i otvori novi projekat
- Pokreni `git bash` (ukoliko si na Windows-u) ili bilo koji terminal emulator (ukoliko si na nekom od Unix derivata) i pozicioniraj se u direktorijum u kom se nalazi tvoj projekat
- Inicijalizuj git repozitorijum
    - `git init` - inicijalizuje git repozitorijum unutar trenutnog foldera
- Provjeri stanje repozitorijuma `git status` prikaže trenutno stanje, koje uključuje:
    - promjene nad fajlovima koje git prati
    - promjene nad fajlovima koje git ne prati

Možeš uočiti da su se pojavili fajlovi iz projekta, za koje git kaže da ih trenutno ne prati.

Inicijalno git ne prati sadržaj fajlova dok ih prvi put ne dodamo u index, nakon toga on kreće sa praćenjem izmjena.

Ignorisanje fajlova

Nekad ne želimo da git prati sve fajlove ili foldere iz repozitorijuma. Tu nam u pomoć uskače `.gitignore` fajl. U ovaj fajl je potrebno da navedemo putanje do fajlova ili foldera koje želimo da git ignoriše. Više o ovom fajlu i ignorisanju možeš naći na zvaničnoj dokumentaciji. Neke početne `.gitignore` fajlove možeš naći na idućem repozitorijumu. Druga opcija je da generišeš ovaj fajl. Možeš koristiti ovaj sajt. Potrebno je da navedeš IDE ili editor koji koristiš za razvoj i programske jezike koje koristiš (npr. za c# možete navesti: Rider, Visual Studio, Visual Studio Code, Csharp) . Nakon toga se generiše fajl koji možeš smjestiti u korijen repozitorijuma. Nakon što dodaš `.gitignore` sve što si naveo u njemu će biti ignorisano i više se neće pojavljivati u statusu.

Dodavanje i uklanjanje izmjena u index

Dalje, želimo da dodamo projektne fajlove u index da bi napravili prvi commit. U nastavku su pobrojane najbitnije naredbe za prebacivanje izmjena između index-a i radnog stabla. *Tip*: dok analiziraš ove naredbe, posmatraj sliku 2 i vizualizuj šta tačno radi svaka od ovih naredbi.

- `git add .`- dodaće sve fajlove iz trenutnog direktorijuma i svih poddirektorijuma rekurzivno u index zonu. Umjesto tačke možeš navesti ime jednog fajla, ili više fajlova razdvojenih razmakom.
- `git add <putanja do fajla>` - dodaje navedeni fajl u index. Moguće je dodati više fajlova, gdje se onda navodi putanja do svakog fajla. Putanje se razdvajaju razmakom.
- `git rm` – uklanja fajl iz index-a i iz radnog stabla (koristi flag `--cached` da bi fajl uklonio samo iz index-a)
- `git restore --staged` – vraća fajl iz index-a u radno stablo (ostaju sve promjene u fajlu, samo više neće biti u index-u).

Sa `git add .` prebaci sve fajlove u index.

Pravljenje commit-a

Kada se fajlovi nalaze u index-u, možemo da napravimo prvi commit:

- `git commit` - otvara editor za unos commit poruke. U komentarima ispod je izlistano šta će biti komitovano, a šta ostaje u index-u (ovdje možete provjeriti da li ste odabrali sve što ste željeli).
- Za commit poruku možete unijeti "Initial commit". Imajte u vidu da pišete kvalitetne commit poruke. Commit poruke pišite na engleskom.

Istorija

Možemo provjeriti graf promjena našeg repozitorijuma sa:

- `git log` - prikaže istoriju commit-ova sa informacijama o svakom od njih.

Možemo uočiti da ima samo jedan commit koji smo upravo napravili i mi smo autor tog commit-a.

- `gitk` – grafički alat pomoću kog možemo da pregledamo istoriju

Vraćamo se u editor i razvijamo novu funkcionalnost za naš mini projekat. Potrebno je da omogućimo korisniku da unese neki tekst. Učitaj od korisnika proizvoljan tekst i ispiši ga.

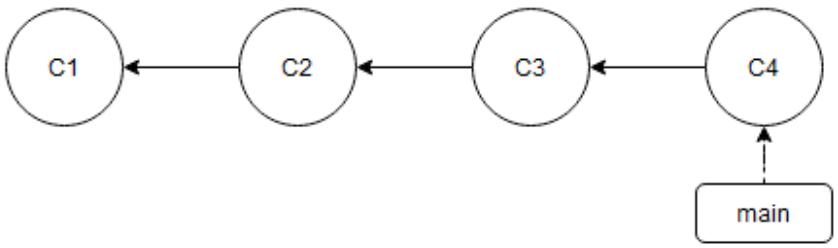
Nakon što si se uvjerio da funkcionalnost radi, vrati se u `git bash` i ponovi korake od ranije: `git status` da vidiš izmjene, zatim sa `git add <fajl>` dodaj fajlove koje želiš da commit-uješ i sa `git commit` napravi commit i unesi smislenu poruku. Uvjeri se da je sve u redu sa `git log` i opciono provjeri stanje repozitorijuma `git status`.

Idući korak je da dodaš petlju u kojoj će kalkulator primati naredbe. Ukoliko korisnik unese `exit` petlja se završava. Commit-uj.

Proširi prethodnu petlju tako da se petlja završi ukoliko korisnik unese `exit` ili `kraj`. Commit-uj.

Implementiraj parsiranje stringova po razmaku i validiraj da ima tačno 3 ulazna parametra. Commit-uj.

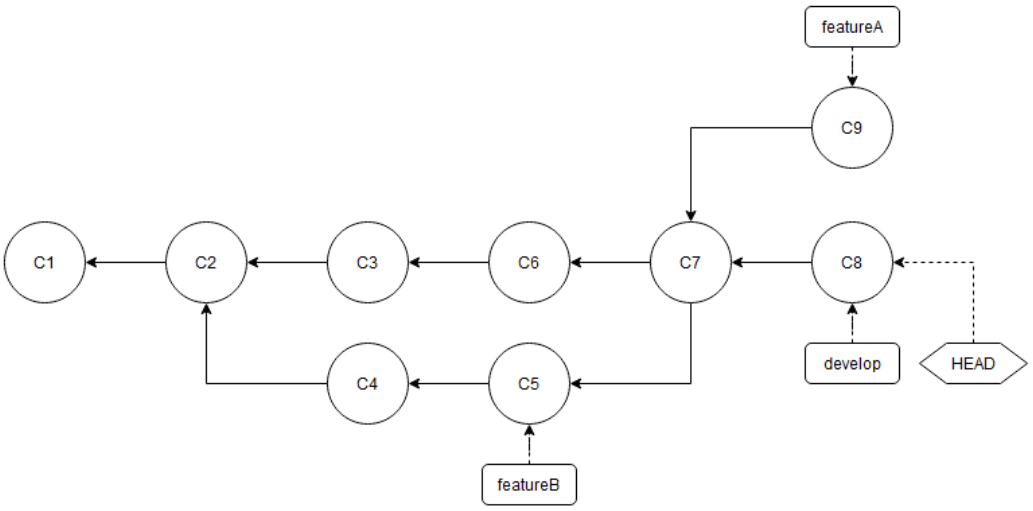
Ako pogledaš log u obliku grafa, trebao bi dobiti nešto slično kao na slici ispod:



Isprobaj komandu `git checkout` pomoću koje se možemo kretati git grafom. Ova komanda kao parametar prima hash commit-a i prebacuje se na njega. Prebaci se na neki od ranijih commit-ova i pogledaj šta se dešava sa projektom u tvom editoru.

Grane

Pravimo kraku pauzu od projekta da bi se osvrnuli na koncept grana u gitu. Grana predstavlja alternativni tok razvoja. To znači da paralelno možemo raditi na dvije (ili više) funkcionalnosti bez ikakvih problema i kad smo završili samo spojimo (engl. *merge*) izmjene na glavnu granu razvoja. Pored ovoga, grane su zgodne jer možemo da isprobamo neku implementaciju i ako ne radi lako odustanemo od nje tako što izbrišemo granu, a izvorni kod aplikacije ostaje netaknut na glavnoj grani.



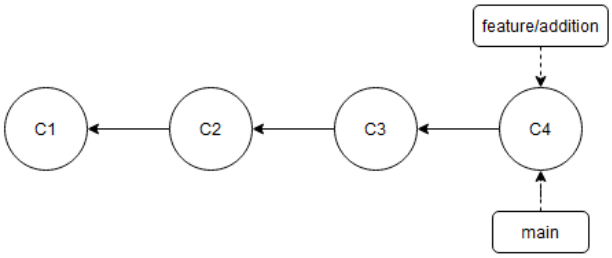
Na slici iznad je prikazan proširen graf sa više grana. Sad vidimo da više commit-ova može imati istog roditelja (C3 i C4 imaju roditelja C2). Ovo su alternativni tokovi razvoja – grane. Grane su prikazane u pravougaonicima sa zaobljenim ivicama. Pored toga, jedan commit može imati i dva roditelja (C7 ima roditelje C5 i C6). Ovaj commit se naziva *merge-commit* i predstavlja spajanje grana. Spajanje grana integriše izmjene sa jedne grane na drugu.W

Pravljenje grana i prelazak na granu

Vraćamo se projektu:

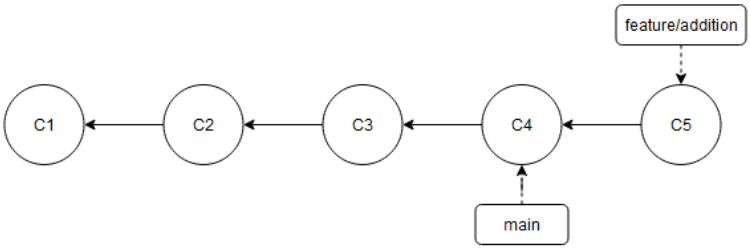
- započećemo razvoj nove funkcionalnosti na novoj grani:
- `git branch feature/addition` će napraviti novu granu sa nazivom `feature/addition`
- da bi se prebacili na granu koju smo upravo kreirali koristimo komandu `git checkout feature/addition`

U `git bash-u` ili terminal emulatoru, možete primjetiti da je naziv grane promijenjen i u samom emulatoru. Izvrši `git log` naredbu i vidi kako trenutno izgleda git graf.



Na novoj grani parsiraj stringove u brojeve, uvjeri se da sve radi uredu i commit-uj.

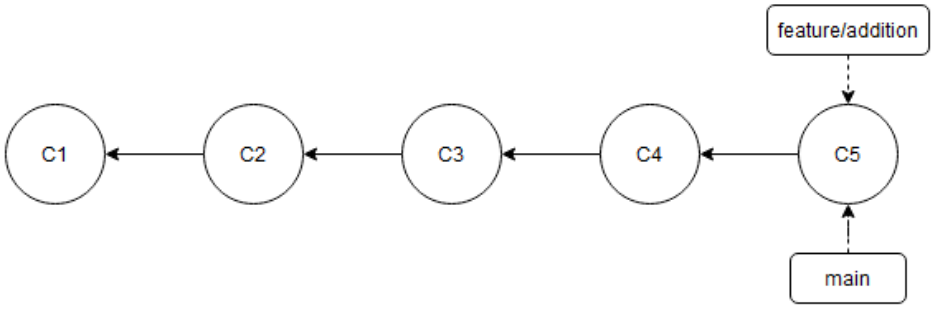
Dodaj funkciju za sabiranje brojeva i integriši je unutar glavne petlje. Uvjeri se da radi kako treba i commit-uj.



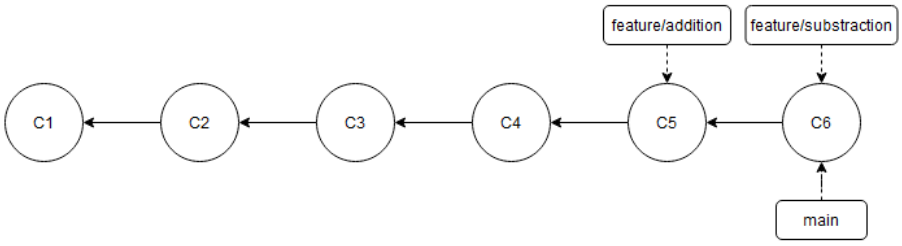
Spajanje grana

Kad smo završili sa razvojem i želimo da novu funkcionalnost integrišemo u glavnu granu radimo *merge*:

- Da bismo spojili `feature/addition` granu na našu glavnu granu (što je kod nas `main`) prvo je potrebno da se prebacimo na `main` granu. Prebaci se na `main` granu (obrati pažnju na editor i sadržaj projekta nakon što si se prebacio).
- Sa `git merge feature/addition` git spaja `feature/addition` granu sa `main`. Pogledaj `git log` i prodiskutuj sa asistentom rezultate.



Na sličan način implementiraj oduzimanje na grani feature/substraction, provjeri da li radi, commit-uj i spoji.



- Izvrši `git log` ili pokreni git-ov integrisani alat za grafički prikaz grafa repozitorijuma `gitk`. Analiziraj sa asistentom alat.

Opciono: poigraj se sa komandama `git show` i `git diff`. Pročitaj dokumentaciju i primjeni ih na repozitorijum.

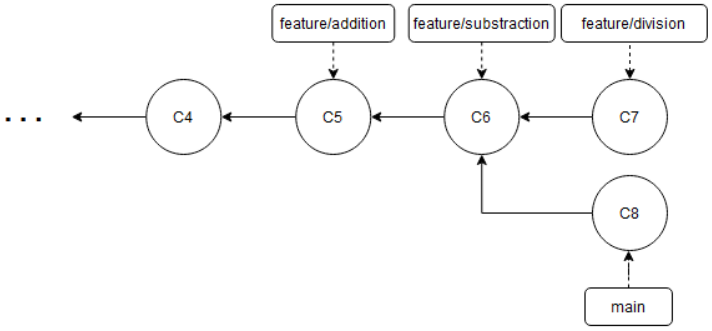
Konflikti

Ponekad se desi da se u toku razvoja moraju mijenjati isti fajlovi i čak iste linije unutar fajlova. Pitanje je kako održati konzistentno stanje repozitorijuma ako se na dvije grane paralelno izmjeni ista linija koda. Git sam može da riješi spajanje ukoliko su različite linije izmjenjene, ali ako je ista linija ili iste linije promijenjena na dvije grane potrebno je da mu pomognemo da razriješi taj problem. Ova pojava u git-u se naziva `merge conflict` i tad git staje sa `merge` naredbom i prepušta nama da riješimo konflikte i dovršimo `merge`.

Vratimo se projektu i hajde da vještački izazovemo jedan konflikt tako što ćemo na `main` grani razviti jednu funkciju koja množi brojeve, a na grani `feature/division` implementirati funkciju koja dijeli brojeve i uvezati je sa ostatkom projekta.

- Prebaci se na `main` granu, ako već nisi na njoj.
- Napravi novu granu `feature/division` i prebaci se na nju, na toj grani implementiraj funkciju za dijeljenje brojeva, commit-uj.
- Vrati se na `main` granu
- Napravi funkciju koja množi brojeve, commit-uj izmjene. Obrati pažnju da se barem jedna linija koda ove funkcije poklapa sa funkcijom na `feature/division` grani.
- Spoji `feature/division` na `main`.

Prodiskutuj sa asistentom rezultat ovog pokušaja `merge`-a.



Razrješavanje konflikata

Dakle konflikti nastanu kad paralelno izmijenimo iste linije koda, pa git nije siguran kako da izvrši merge. Git onda svaki od konflikata označi markerima koji izgledaju ovako:

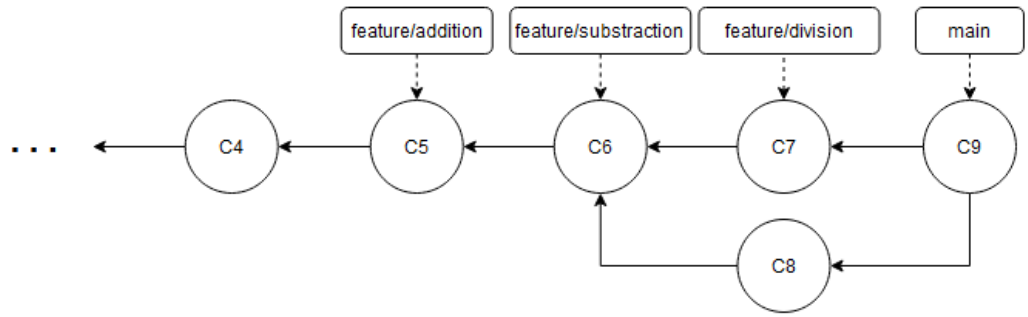
```
<<<<<<< HEAD:file

..... (u ovom prvom dijelu se nalazi trenutna izmjena - to su izmjene na grani na kojoj smo bili kad smo započeli merge)

=======

..... (u drugom dijelu su izmjene na drugoj grani - granu koju smo htjeli da merge-ujemo)

>>>>>>> <naziv grane ili commit-a>:file
```

Potrebno je da svaki od konflikata riješiš, dodaš promjene koje si napravio da bi riješio konflikte u index zonu i komituješ. Ovdje će ti se sama commit poruka popuniti koju možeš zadržati.

Primjer konflikta:

```
1.  <<<<<<<
2.  private double Multiply(double a, double b) {
3.      return a*b;
4.  }
5.  =======
6.  private double Divide(double a, double b) {
7.      return a/b;
8.  }
9.  >>>>>>>
10.
```

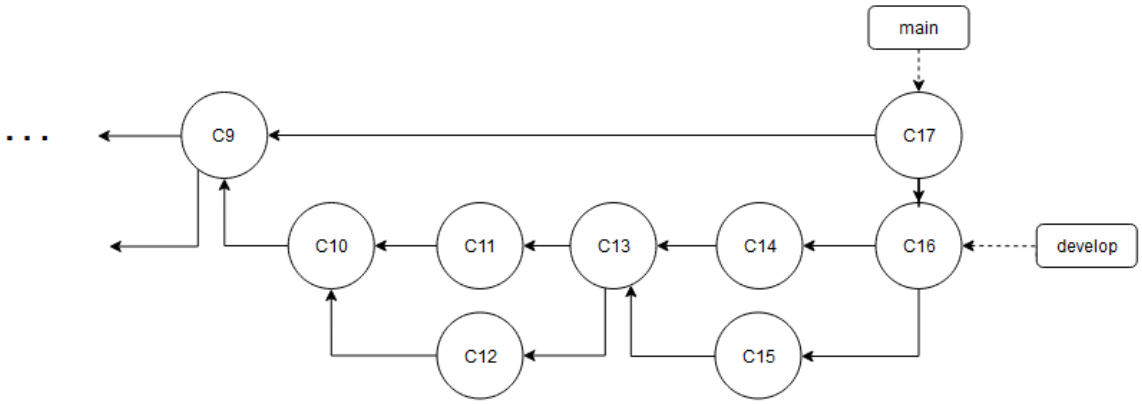
Sličan konflikt bi trebao imati i ti ako si pratio korake iznad. Da bi riješio konflikt treba da odlučiš šta nam od ovih promjena treba. U ovom slučaju želimo da zadržimo obje funkcije, pa prvo ukloni markere ("<<<<", "====", ">>>>") i provjeri da li sve radi kako treba. Ukoliko je sve u redu, dodaj izmjene u index zonu i commit-uj. Nakon commit-a provjeri kako izgleda graf i prodiskutuj stanje grafa sa asistentom.



Vraćamo se na projekat, sad treba da imaš sve izmjene na `main` grani. Vođa tima je odlučio da se od sad na `main` commit-uju samo nova izdanja projekta. Napravi `develop` granu od `main` grane na kojoj ćeš da razvijaš ostatak projekta.

- Implementiraj na feature grani funkcionalnost računanja eksponenta u obliku: `x ^ y`. Na drugoj feature grani implementiraj funkciju koja ispisuje "`Hello, friend!`". Spoji prvu feature granu u `develop`. Spoji drugu feature granu u `develop` i usput riješi konflikte
- Na novoj feature grani dodaj podršku za sabiranje stringova. Ukoliko je korisnik unio dva stringa, kao rezultat sabiranja ispiši konkatenovan string.
- Vrati se na `develop` i napravi novu feature granu koja dodaje podršku za množenje stringova. Ukoliko je jedan parametar unosa string (neka to bude `x`), a drugi broj (recimo `y`), kao rezultat ispiši `y` puta ponovljen string `x` (primjer: za ulaz `a * 3` ispiši `aaa`).
- Spoji obje feature grane i riješi konflikte usput.
- Spoji `develop` na `main`.
- Napravi novu feature granu, na njoj implementiraj istoriju kalkulatora. Kad god se unese nova naredba, potrebno je da je sačuvaš u fajl pod nazivom *history.txt*. Nemoj da commit-uješ *history.txt* fajl. Nakon što si commit-ovao logiku implementacije istorije, u *.gitignore* dodaj *history.txt* fajl i commit-uj.

Na kraju bi tvoj repozitorijum trebao imati strukturu kao na slici ispod.



Literatura i inspiracija

- https://git-scm.com/book/en/v2 - Pro git knjiga (git Biblija), preporučuje se da pročitaš prva 3 poglavlja
- https://missing.csail.mit.edu/2020/version-control/ - video i tekstualni materijali MIT-a o git-u
- http://www.igordejanovic.net/courses/tech/git/ - odlični slajdovi koje je napravio profesor Dejanović, osnovni i napredni pojmovi su detaljno objašnjeni. Dodatno, sve je na srpskom.