Git

Šta je git objašnjeno na jednostavnom primjeru

Git je alat za kontrolu verzija. Omogućuje nam da pratimo istoriju razvoja projekta, verzionišemo ga i olakšava kolaboraciju sa drugim učesnicima na projektu.

Jedna od brojnih mogućnosti koje ti git nudi jeste istorija izmjena. Svaki editor ima ugrađenu mogućnost *„undo“* operacije pomoću koje možeš da se poništi prethodna akcija. Većina editora omogućava da se poništi i više prethodnih akcija. Ovako posmatrano, editor pamti istoriju izmjena. Git takođe omogućuje istoriju izmjena, ali na malo drugačiji način. Razlika između git-a i editora je to što editor pamti akcije koje su se desile automatski, a git ne, već programer sam pravi svoje verzije (commit u git-u). 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 jeste praćene sadržaja. Rekli smo da git ne pamti akcije koje su se desile, već git prati sadržaj i prikazuje sve promjene koje su se desile. Npr. Programer je u editoru dodao novu liniju koda u jedom fajlu, 10 linija u drugom i napravio novi fajl. Git će prikazati sve ove izmjene, gdje programer onda bira šta želi da bude nova verzija. Nova verzija ne mora da sadrži izmjene, već može da sadrži samo neki podskup. Kad se napravi nova verzija potrebno je da se da opis zašto je napravljena ta verzija (taj skup izmjena). To se radi kako bi kasnije kad bi se trebalo vratiti na nju lako mogli naći koja nam 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 je potrebno da se krene sa radom. Git ima odličnu dokumentaciju koja će biti linkovana na svim mjestima. Toplo preporučujemo literaturu koja je navedena pri kraju dokumenta. Ideja ovog dokumenta je da služi kao uvod u priču o gitu.

Uvod

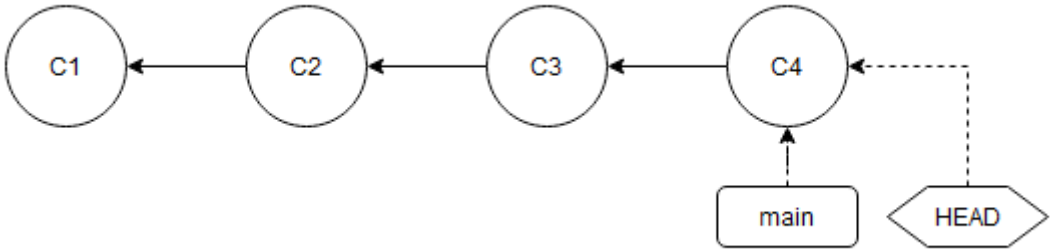
Git je distribuirani sistem za kontrolu verzija. Prednosti:

- 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.

Za početak commit se može posmatrati kao jedna verzija projekta. Git je napravljen tako da kad se kreće između commit-ova izgleda kao da se kreće između verzija 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 se dobla verziju projekta za neki commit, 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. Git dodatno radi razne optimizacije kako bi ova operacija bila brza.



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 omogućuju da više ljudi paralelno radi na projektu, jedna osoba može da šalje svoje grane drugim članovima tima, oni mogu da nastave rad dalje, itd. Dodatno, grane mogu da se spajaju. Na taj način se integriše posao koji je razvijan paralelno na više grana u jednu granu koja će onda sadržati sve izmjene. Glavna grana u projektu se obično zove `main` ili `master` i ona se inicijalno kreira kad napravimo novi git repozitorijum.

Instalacija

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

Instalacija je prilično jednostavna, savjet je da se pažljivo prate korake instalacije jer se tu bira podrazumijevani editor za rad sa git-om. Inicijalna vrijednost je Vim i preporuka je da se odabere neki drugi editor ukoliko niste upoznati sa Vim-om. Nije problem ukoliko se odabere pogrešan editor, lako se može zamijeniti kasnije (uputstvo).

Konfiguracija

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

Par stvari je potrebno da se odmah konfigurišu: (Ukoliko radite u učionici u kojoj više studenata dijeli isti laptop, ne preporučujemo da radiš ovo podešavanje sad, već nakon što napravite novi git

repozitorijumm pokreni ovu komandu unutar njega (repozitorijuma) bez flag-a `--global`. Ovako će vaše 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`.

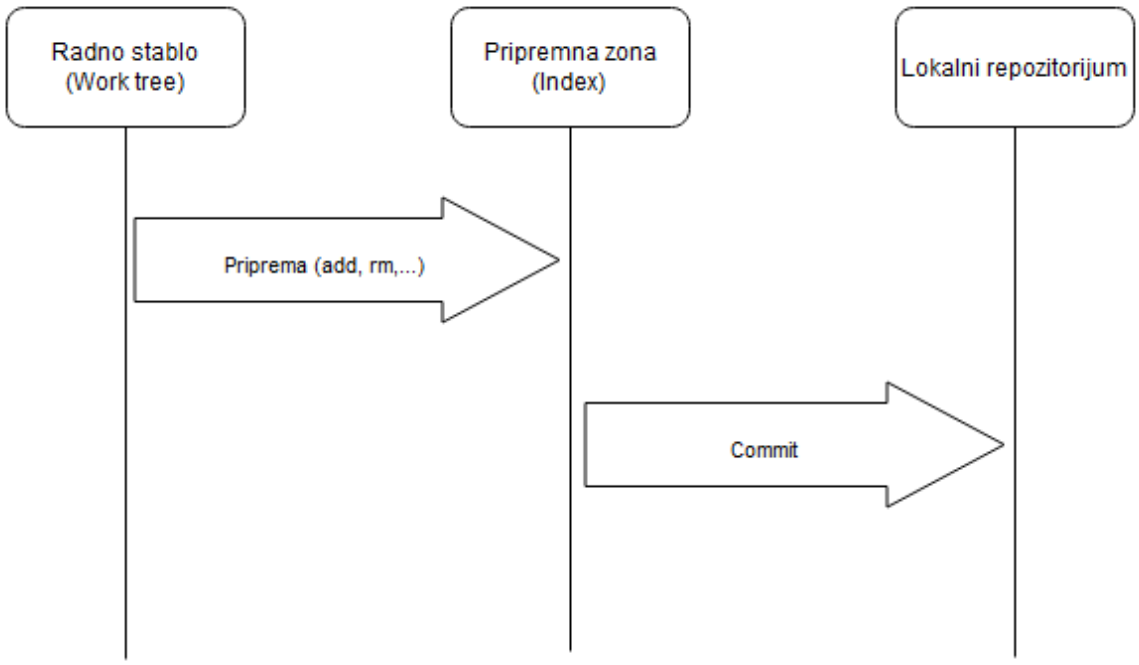
Praćenje sadržaja

Git radi tako što prati promjene svih fajlova unutar repozitorijuma (tvog projekta).

Git prikazuje koje smo sve izmjene načinili u projektu (recimo: dodali smo 10 linija koda u jedan fajl, izbacili 4 linije iz drugog, napravili novi fajl, itd.). Od svih izmjena koje nam prikazuje, potrebno je da odaberemo izmjene koje želimo da budu dio idućeg commit-a. Odabrane izmjene prebacimo u *pripremnu zonu* (*index*).

Kad su sve izmjene koje idu u idući commit u *index*-u zoni, sa *commit* naredbom pravimo novi commit. Taj commit će sve promjene koje su bile u index zoni smjestiti u *lokalni repozitorijum* (ovo možemo posmatrati kao da git napravi novu verziju repozitorijuma u koju uključi promjene koje smo commit-ovali).

Ovaj workflow je ilustrativan na slici ispod. Na ovaj način moguće je odabrati samo dijelove koda koje želimo da commit-ujemo 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 editoru napraviti i otvoriti novi projekat
- Pokrenuti `git bash` (ukoliko ste na Windows-u) ili bilo koji terminal emulator (ukoliko ste na nekom od Unix derivata) i pozicionirati se u direktorijum u kom se nalazi vaš projekat
- Inicijalizovati git repozitorijum
  - `git init` - inicijalizuje git repozitorijum unutar trenutnog foldera
- Provjeriti 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. Neki početni `.gitignore` fajlovi se mogu naći na repozitorijumu. Druga opcija je da se generiše ovaj fajl. Može se koristiti ovaj sajt. Potrebno je da se navede IDE ili editor koji koristite za razvoj i programske jezike koje koristite (npr. za c# možete navesti: Rider, Visual Studio, Csharp) . Nakon toga se generiše fajl koji smjestite u korijen repozitorijuma. Nakon što dodate `.gitignore` sve što je navedeno u njemu će biti ignorisano i više se neće pojavljivati u statusu.

Ukoliko ste dobro kreirali `.gitignore`, u ispisu `git status`-a će nestati `bin/` i `obj/` folderi ukoliko radite u C# programskom jeziku.

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. *Savijet*: dok analizirate ove naredbe, posmatrajte sliku 2 i vizualizujte šta tačno radi svaka od ovih naredbi.

- `git add .` - dodaće sve fajlove iz trenutnog direktorijuma i svih poddirektorijuma rekurzivno u index zonu. Umjesto tačke možete 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 <putanja do fajla>` – uklanja fajl iz index-a i iz radnog stabla (koristiti flag `--cached` da bi fajl uklonio samo iz index-a)
- `git restore –staged <putanja do fajla>` – 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 prvoj liniji unesite poruku za taj commit. Poruka treba da sadrži informaciju zašto ste napravili izmjene koje se commit-uju. Nakon što napišete poruku, sačuvajte taj fajl i zatvorite ga (ili zatvorite editor). 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).
- `git commit -m „opis poruke“` – radi isto što i komanda iznad ali bez otvaranja editor-a, već se poruka definiše pomoću `-m` flag-a

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. (*savijet*: za izlazak iz git log-a koristi slovo *q*)

Možemo uočiti da graf ima samo jedan commit koji smo upravo napravili i da smo mi 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.

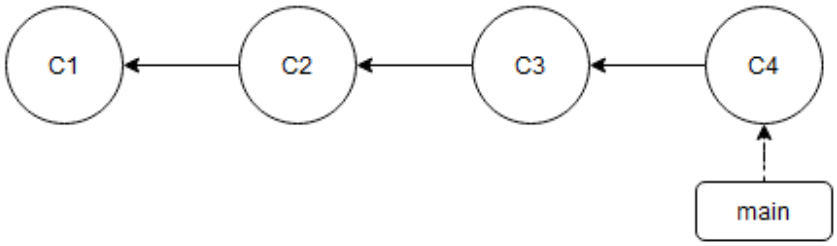
Nakon što ste se uvjerili da funkcionalnost radi, vratite se u `git bash` i ponovite korake od ranije: `git status` da vidite izmjene, zatim sa `git add <fajl>` dodajte fajlove koje želite da commit-ujete i sa `git commit` napravite commit i unesite smislenu poruku. Uvjerite se da je sve u redu sa `git log` i opciono provjerite stanje repozitorijuma sa `git status`.

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

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

Implementirajte parsiranje stringova po razmaku i validirajte da ima tačno 3 ulazna parametra. Commit-ujte.

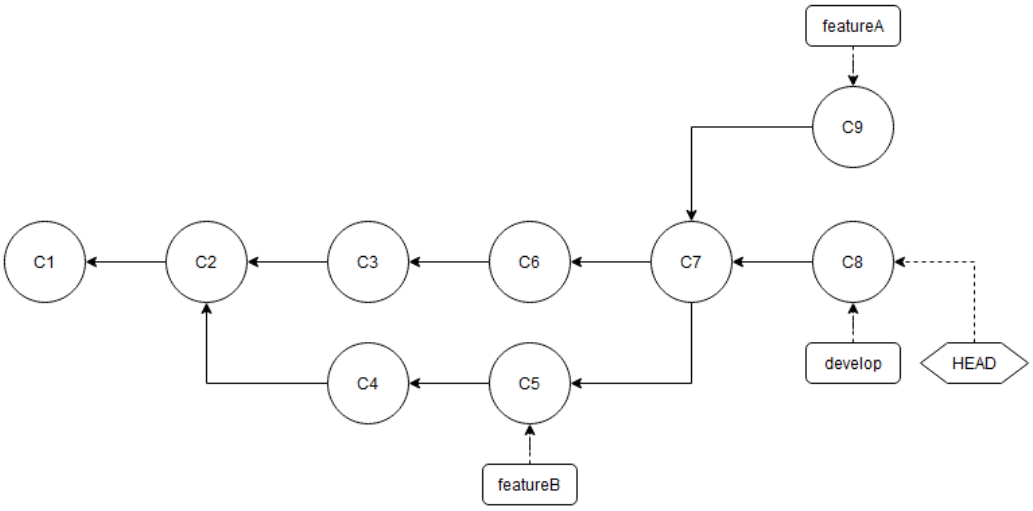
Ako pogledate log u obliku grafa, trebalo bi da liči na graf sa slike:



Opciono: Isprobajte komandu `git checkout` pomoću koje se možemo kretati git grafom. Ova komanda kao parametar prima hash commit-a i prebacuje se na njega. Upamtite na kojoj ste grani razvoja trenutno (vjerovatno se zove `main` ili `master`, uskoro ćemo se detaljnije baviti granama). Prebacite se na neki od ranijih commit-ova i pogledaj šta se dešava sa projektom u editoru. Kad budete htjeli da se vratite nazad, uradite checkout na `main`/`master`.

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.



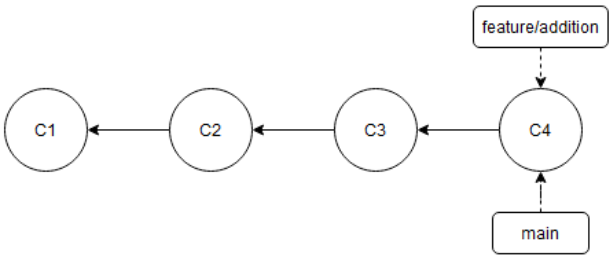
Na slici iznad je prikazan proširen graf sa više grana. Sad vidimo da više commit-ova može imati istog roditelja (C3 i C4 imaju roditelja C2). Ovo su alternativni tokovi razvoja – grane. Grane su prikazane u pravougaonicima sa zaobljenim ivicama. Pored toga, jedan commit može imati i dva roditelja (C7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

Pravljenje grana i prelazak na granu

Vraćamo se projektu:

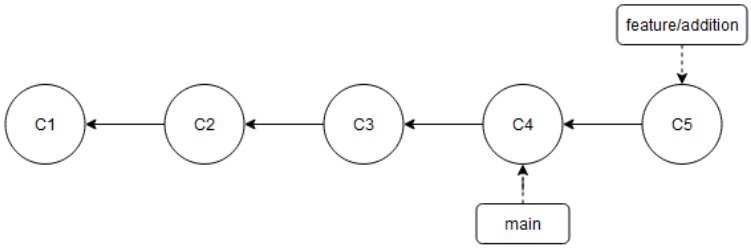
- započećemo razvoj nove funkcionalnosti na novoj grani:
- `git branch feature/addition` će napraviti novu granu sa nazivom `feature/addition`
- da bi se prebacili na granu koju smo upravo kreirali koristimo komandu `git checkout feature/addition`

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



Na novoj grani parsirajte stringove u brojeve, uvjerite se da sve radi uredu i commit-ujte.

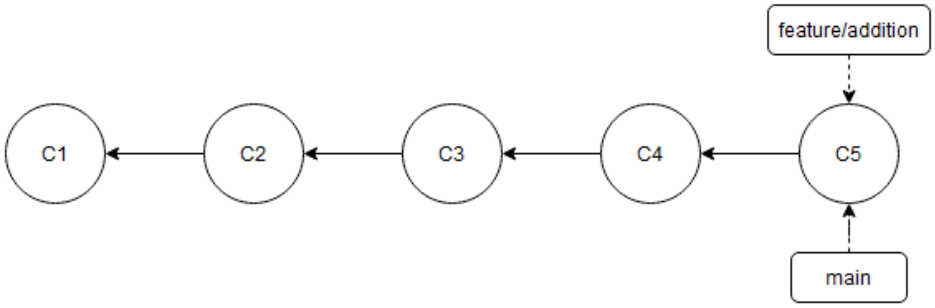
Dodajte funkciju za sabiranje brojeva i integrišite je unutar glavne petlje. Uvjerite se da radi kako treba i commit-ujte.



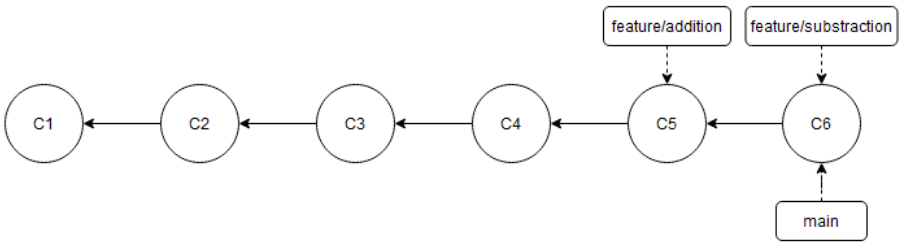
Spajanje grana

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. Prebacite se na `main` granu (obratite pažnju na editor i sadržaj projekta nakon što smo se prebacili).
- Sa `git merge feature/addition` git spaja `feature/addition` granu sa `main`. Pogledajte `git log` i prodiskutujte sa asistentom rezultate.



Na sličan način implementirajte oduzimanje na grani feature/substraction, provjerite da li radi, commit-ujte i spojite.



- Izvršite `git log` ili pokrenite git-ov integrisani alat za grafički prikaz grafa repozitorijuma `gitk`. Analizirajte sa asistentom alat.

Opciono: istražite komande `git show` i `git diff`. Pročitajte dokumentaciju i primjenite ih na repozitorijumu.

Konflikti

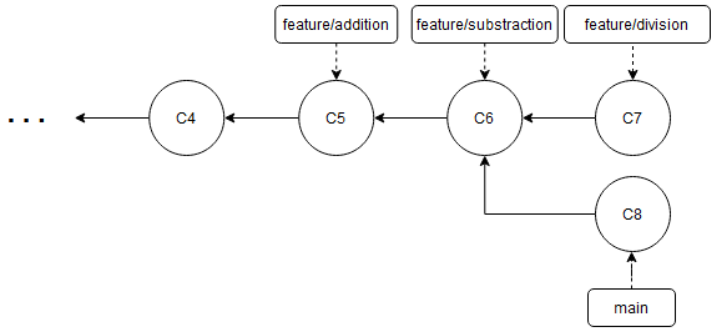
Ponekad se desi da se u toku razvoja moraju mijenjati isti fajlovi ili čak iste linije unutar fajlova. Pitanje je kako održati konzistentno stanje repozitorijuma ako se na dvije grane paralelno izmjeni ista linija koda. Git sam može da riješi spajanje ukoliko su različite linije izmjenjene, ali ako je ista linija ili iste linije promijenjene 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.

- Prebacite se na `main` granu, ako već niste na njoj.
- Napravite novu granu `feature/division` i prebacite se na nju, na toj grani implementirajte funkciju za dijeljenje brojeva, commit-ujte.
- Vratite se na `main` granu
- Napravite funkciju koja množi brojeve, commit-ujte izmjene. Obratite pažnju da se barem jedna linija koda ove funkcije poklapa sa funkcijom na `feature/division` grani.

- Spojite `feature/division` na `main`.

Prodiskutujte 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
```

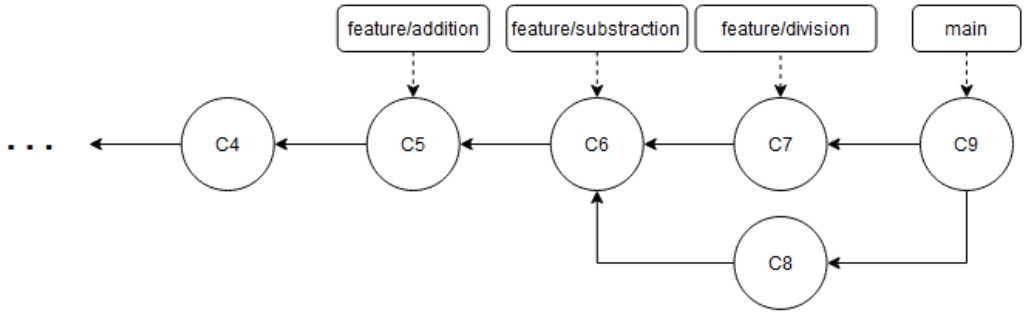
Potrebno je da se svaki od konflikata riješi, dodaju promjene koje su napravljene prilikom riješavanja konflikta u indeks zonu (što označi da je konflikt riješen) i komituje. Ovdje će vam se sama commit poruka popuniti koju možete zadržati.

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 trebali imati i vi ako ste pratili korake iznad. Da bi riješili konflikt treba da odlučite šta nam od ovih promjena treba. U ovom slučaju želimo da zadržimo obje funkcije, pa prvo uklonite markere ("<<<<", "====", ">>>>") i provjerite da li sve radi kako treba. Ukoliko je sve u redu, dodajte izmjene u index zonu i commit-ujte. Nakon commit-a provjerite kako izgleda graf i prodiskutujte stanje grafa sa asistentom.

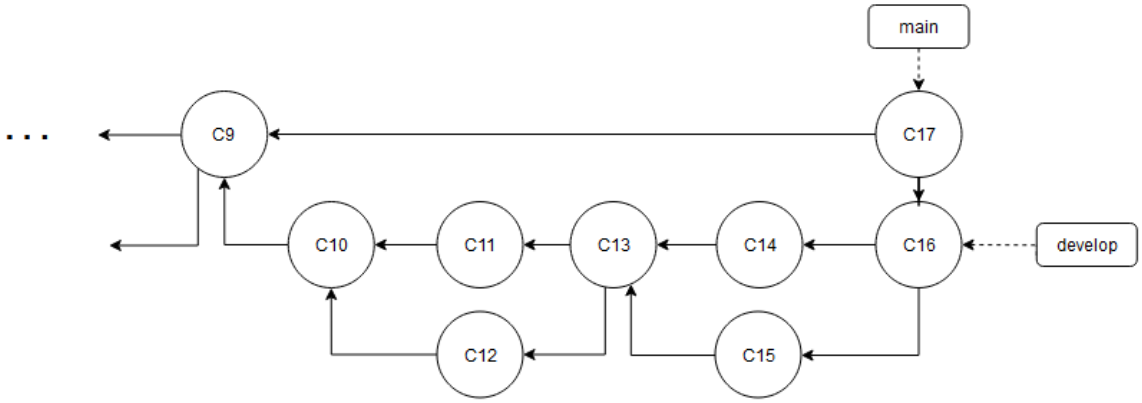
Poenta rješavanja konflikata jeste da napravimo da projekat opet radi kako je očekivano. Da bismo uspješno riješili konlfikt potrebno je da znamo šta treba da radi kod nad kojim se desio konflikt, da uklonimo konflikt markere i napravimo da taj kod opet radi kako je očekivano.



Vraćamo se na projekat, sad treba da imate 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 ćete da razvijate ostatak projekta.

- Implementirajte na novoj grani funkcionalnost računanja eksponenta u obliku: `x ^ y`. Vratite se na develop i napravite još jednu granu i na njoj implementirajte funkciju koja ispisuje "`Hello, friend!`". Spojite prvu granu u `develop`. Spojite drugu granu u `develop` i usput riješite konflikte
- Na novoj grani dodajte podršku za sabiranje stringova. Ukoliko je korisnik unio dva stringa, kao rezultat sabiranja ispišite konkatenovan string.
- Vratite se na `develop` i napravite novu granu koja dodaje podršku za množenje stringova. Ukoliko je jedan parametar unosa string (neka to bude `x`), a drugi broj (recimo `y`), kao rezultat ispišite `y` puta ponovljen string `x` (primjer: za ulaz `a * 3` ispišite `aaa`).
- Spojite obje grane na develop i riješite konflikte usput.
- Spojite `develop` na `main`.
- *Dodatan zadatak*: Prebacite se na develop i napravite novu granu, na njoj implementirajte istoriju kalkulatora. Kad god se unese nova naredba, potrebno je da je sačuvate u fajl pod nazivom *history.txt*. U *.gitignore* dodaj *history.txt* fajl i commit-ujte, a zatim commit-ujte kod koji čuva istoriju u fajl.

Na kraju bi vaš repozitorijum trebao imati strukturu kao na slici ispod (ukoliko ima odstupanja provjerite sa asistentom).



Literatura i inspiracija

- https://git-scm.com/book/en/v2 - Pro git knjiga (git Biblija), preporučuje se da pročitate 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.