Git - distribuirani sistem kontrole verzija

Dizajn i razvoj softvera

Slavko Kovačević[1]

[1]Elektrotehnički fakultet
Univerzitet Crne Gore

februar 2023. godine

Sadržaj

1. Uvod
2. Instalacija
3. Konfiguracija
4. Inicijalizacija prvog repozitorijuma
5. Predaja izmjena
6. Dobre i loše poruke predaje
7. Log predaja
8. Arhitektura
9. Integritet podataka
10. Pokazivač na glavu
11. Dodavanje fajlova
12. Ažuriranje
13. Pregled razlika
14. Brisanje
15. Premještanje i preimenovanje
16. Poništavanje izmjena
17. Uklanjanje nepraćenih fajlova
18. Ignorisanje fajlova
19. Zaključak

Uvod I

Kontrolom verzija bavimo se svakodnevno, a da toga nismo ni svjesni. Uzmimo za primjer da radimo seminarski iz nekog predmeta i pišemo ga u Word-u. Napravili smo početnu verziju i dali joj ime verzija1.docx. Nakon određenih izmjena dokument smo sačuvali kao verzija2.docx, a onda ga poslali profesoru koji nam je proslijedio komentare koje smo usvojili i nakon čega smo dokument sačuvali pod nazivom final1.docx.
Ponovnim čitanjem uočili smo još sitnih greški i završili s verzijom final2.docx. Iz primjera se pokazuje da vođenje računa o tekućoj verziji nije zahvalan posao, a situacija se komplikuje ako na dokumentu radimo u paru ili s više kolega.

Uvod II

Git je softver koji vodi računa o promjenama fajlova i foldera, specijalizovan za upotrebu u programiranju. Git nije prvi softver ovakve namjene.
Važno je naglasiti da je Git distribuirani sistem kontrole verzija. Za razliku od Git-a, četiri njegova najveća prethodnika koristili su model centralnog repozitorijuma - centralizovano mjesto gdje se master kopija koda čuva. Programer klonira kod, napravi određene izmjene i podnosi ih centralnom repozitorijumu.
Git ne funkcioniše na taj način već svakom korisniku omogućava da ima svoj repozitorijum time što ne čuva verzije dokumenata niti razlike između njih - Git zapravo enkapsulira sve izmjene u cjeline i razmjenjuje ih između korisničkih repozitorijuma. Korisnik broj 1 može imati skup cjelina A, B, C, D i E dok korisnik broj 2 može posjedovati cjeline A, D i F.

Uvod III

Nijedan od ovih repozitorijuma nema pravu verziju i nije više u pravu od drugog, on naprosto posjeduje određene cjeline i može ih razmijeniti. Međutim, u praksi se Git ne ponaša ovako. Po pravilu, uvijek postoji centralni master repozitorijum s kojim ostali korisnici komuniciraju, ali to nije dio Git arhitekture već stvar odluke. Učesnici jednog projekta dogovore se da će sve izmjene usklađivati s centralnim repozitorijumom (koji se nalazi na proizvoljnom softver hostingu kao što je GitHub), ali je važno razumjeti da ih Git na to ne primorava.

Uvod IV

Git je, pored svih svojih prednosti u odnosu na nekadašnju konkurenciju (koja više ne postoji), nastao slučajno. Potrebno je bilo donijeti odluku koji će softver za kontrolu toka biti korišten za praćenje verzija Linux kernela i odlučeno je da to bude BitKeeper. BitKeeper se plaćao, ali je imao neplaćenu verziju koja je ukinuta 2005. godine. Kako je sam Linux kernel softver otvorenog koda (zajednica volonterski doprinosi razvoju) i u potpunosti besplatan, bilo je neprihvatljivo da za potrebe kontrole njegovih verzija bude plaćan drugi softver.

Uvod V

Linus Torvalds, tvorac Linux kernela, odlučio je da razvije svoj softver koji je dobre stvari preuzeo od BitKeeper-a, a usput ispravio njegove mane. Nazvao ga je Git potpuno nasumično - to je kombinacija tri karaktera koju je moguće izgovoriti, a koja nije bila već postojeća Unix komanda. Kao i Linux kernel, Git je također softver otvorenog koda - ubrzo je postao popularan zbog svojih kvaliteta, ali i zbog mogućnosti da bude unaprijeđen od strane zajednice. Nedugo zatim pojavio se i GitHub 2008. godine i premotajmo 15 godina, Git je standard kontrole verzija.

Instalacija

- Pretpostavićemo da svi studenti koriste Windows OS
- Git se može preuzeti sa sljedećeg linka: https://git-scm.com/download/win
- Nakon što se instaler preuzme pokrenuti ga i odabrati predložene opcije

Uspješnost instalacije provjerimo tako što u komandnom interpretatoru izvršimo sljedeću naredbu:

```
PS C:\Users\skovacevic> git --version
git version 2.33.0.windows.2
```

Dobili smo odgovor koju verziju Git-a imamo instaliranu što znači da je instalacija uspješno izvršena.

Konfiguracija

Git se može konfigurisati na tri nivoa:

- Sistem (system) koji se odnosi na sistem
- Korisnik (global) koji se odnosi na trenutnog korisnika
- Projekat (local) koji se odnosi isključivo na tekući projekat

Preciznije, Git konfiguracione informacije u vidu fajla (.gitconfig) može čuvati na tri mjesta. U konfiguracionom fajlu možemo npr. zapisati naše ime i mejl adresu:

```
PS C:\Users\skovacevic> git config --global user.name "Slavko Kovacevic"
PS C:\Users\skovacevic> git config --global user.email "skovacevic@ucg.ac.me"
```

Sve trenutne konfiguracije možemo provjeriti sa:

```
PS C:\Users\skovacevic> git config --list
```

Inicijalizacija prvog repozitorijuma I

Kako bismo inicijalizovali novi repozitorijum, potrebno je da prvo napravimo projekat (odnosno folder) u kojem će se naš repozitorijum nalaziti. Po dogovoru, sve naše foldere čuvaćemo u folderu Java koji se nalazi u folderu Documents. U folderu Java napravimo folder projekat. Međutim, ovom folderu treba pristupiti preko komandne linije. To ćemo uraditi upotrebom komande cd:

```
PS C:\Users\skovacevic> cd Documents\Java\projekat
```

Sada smo spremni da inicijalizujemo repozitorijum:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git init
Initialized empty Git repository in C:/Users/skovacevic/Documents/Java/
projekat/.git/
```

Inicijalizacija prvog repozitorijuma II

Git je za nas napravio folder .git koji će voditi računa o kontroli verzija. Važno je napomenuti da su fajlovi i folderi koji počinju s tačkom podrazumijevano skriveni. Naučiti kako ih je moguće prikazati.
Tim fajlovima svakako možemo pristupiti pomoću komandne linije. Sada možemo npr. provjeriti sadržinu konfiguracionog fajla za naš projekat:

```
PS C:\Users\skovacevic\Documents\Java\projekat> cat .git/config
[core]
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
```

Predaja izmjena

Prije negoli predamo prvu izmjenu, potrebno je nešto i promijeniti u repozitorijumu. Kako je repozitorijum trenutno prazan napravimo jedan tekstualni fajl fajl.txt i u njega upišimo jednu liniju teksta. Sada, Git-u treba dati naredbu da evidentira da je došlo do izmjene:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git add fajl.txt
```

ali Git i dalje ne prati izmjene sve dok ih ne predamo repozitorijumu pomoću sljedeće komande:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git commit -m "Initial commit"
```

gdje smo sa -m dodali poruku predaje.
Ova tri koraka će se ponavljati iznova i iznova:

- napravimo izmjenu
- dodamo izmjenu
- predamo izmjenu repozitorijumu uz odgovarajuću poruku

Dobre i loše poruke predaje

Uopšteno, treba se pridržavati sljedećih pravila:

- Poruke predaje trebaju biti sažeci od ne više od 50 karaktera.
- Poruke predaje su nastavak “This commit...”.
- Poruke predaje se ne završavaju tačkom.
- Prvo slovo poruke predaje je uvijek veliko.
- Poruke predaje pišu se u trećem licu.
- Poruke predaje pišu se na engleskom jeziku u prezentu, npr. “Fix a bug in...”.
- Poruke predaje ne smiju biti generičke, npr “...Add new feature”.

Kako poruke predaje ostaju vječno sačuvane godinama i čitaju ih vaše kolege (a i vi) njihov zadatak prevashodno je da nedvosmisleno daju odgovor na pitanje koju izmjenu je određena predaja donijela.

Log predaja

Sve dosadašnje predaje možemo izlistati pomoću:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git log
commit 1cd2a4b89be37ea0b18a8aab585673cdeb6f2ecd (HEAD -> master)
Author: Slavko Kovacevic <skovacevic@ucg.ac.me>
Date: Fri Feb 24 15:34:31 2023 +0100

    Initial commit
```

Pa uočavamo da svaka predaja ima svoj ID po kojem se razlikuje od drugih predaja. Takođe, predaja ima autora, datum izvršenja i naravno poruku koju smo sami unijeli. Šta nam sve nudi log predaja možemo provjeriti pomoću naredbe:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git help log
```

Predaje se mogu listati po datumu, korisnicima, mejl adresama, a možemo pogledati poslednjih npr. 5 ili vršiti pretragu pomoću regularnog izraza.

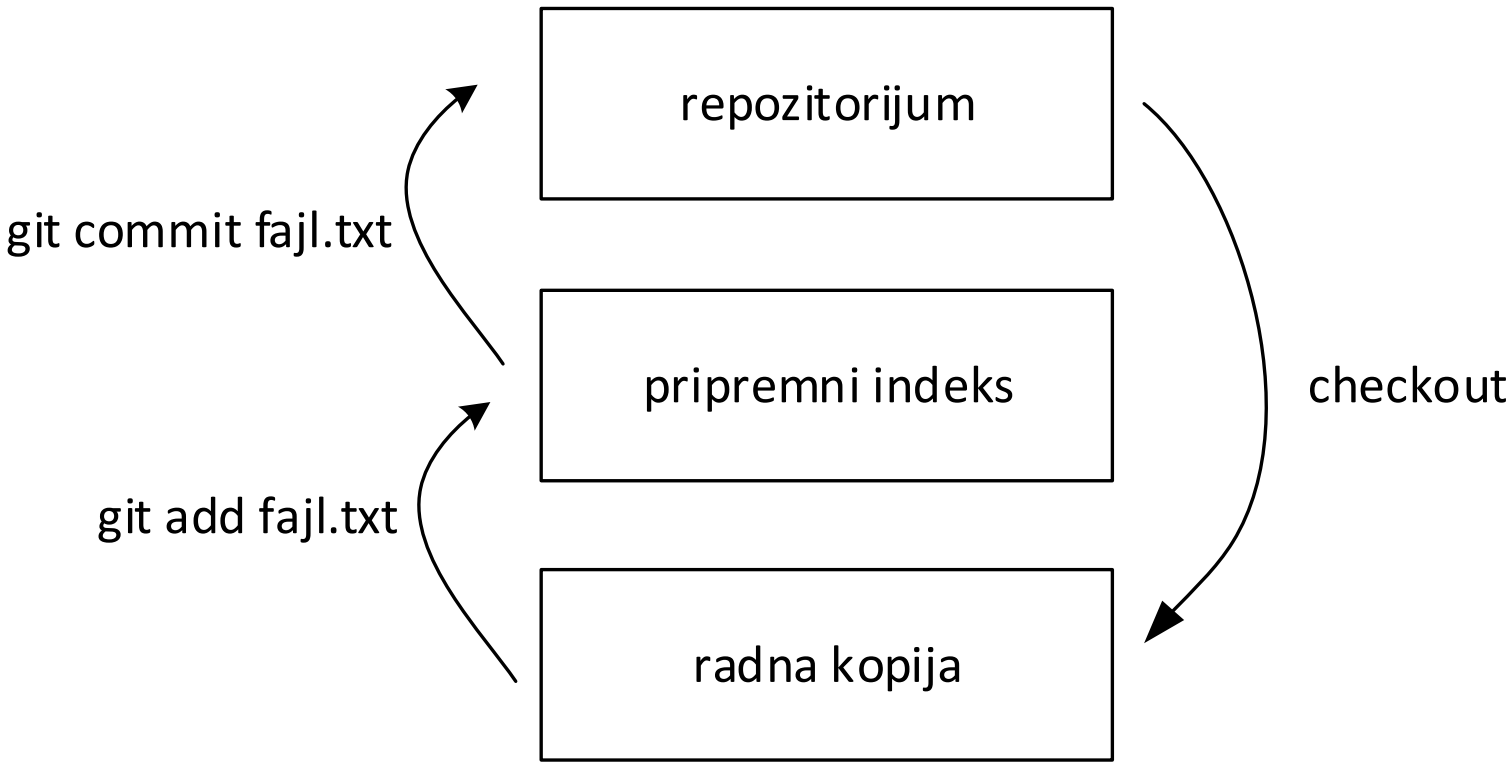
Arhitektura I

Git koristi arhitekturu tri drveta. Ova tri drveta su:

- repozitorijum (repository)
- pripremni indeks (staging index)
- radna kopija (working copy)

Prije svega, vrijedi objasniti da su ova drveta zapravo folderi koji sadrže druge foldere i fajlove, otuda i naziv. Git-ovi prethodnici koristili su arhitekturu s dva drveta, odnosno arhitekturu bez pripremnog indeksa.

Arhitektura II



Slika: Git arhitektura.

Arhitektura III

Pripremni indeks se može preskočiti, ali to nije dobra praksa. Objasnimo to na jednom primjeru. Naš program ima jedan minorni problem. Krenuli smo da ga rješavamo, stigli do pola rješenja, a u tom trenutku javio nam se klijent s prijavom drugog problema visokog prioriteta. Bez pripremnog indeksa predajom izmjena koje utiču na drugi problem, u repozitorijumu bi završilo i polovično rješenje prvobitnog problema što nikako nije dobro - repozitorijumu smo proslijedili nefunkcionalan kod i prekršili pravilo da jedna predaja predstavlja jednu cjelinu.
Radna kopija i pripremni indeks mogu sadržati samo po jednu verziju fajla, odnosno tačnije rečeno po jednu cjelinu koja predstavlja izmjene, dok su sve ostale izmjene sačuvane u repozitorijumu. Repozitorijum te izmjene ne čuva kao verzije već kao predaje.

Integritet podataka I

Da odredi ID predaje, Git računa kontrolni zbir pomoću SHA-1 algoritma. Kontrolni zbir za iste podatke je uvijek isti. Drugim riječima, labela koju Git daje svojim predajama fundamentalno je vezana za ono šta je u njima. Kontrolni zbir se računa u odnosu na sve prethodne podatke. Ukoliko se informacije promijene, promijeniće se i kontrolni zbir. U proračun ulaze i podaci o podacima - te nije moguće promijeniti poruku predaje niti njenog autora. Kad se izračuna nova kontrolna suma na osnovu promjene informacije, ona sadrži sve podatke prethodnih predaja odnosno podataka, time se Git predaje uvezuju (tako što svaka predaja ima znanog roditelja koji predstavlja prethodnu predaju) i omogućava se istorijski pregled između njih. Ukoliko bismo promijenili bilo koju predaju, promijenila bi se i njena SHA vrijednost i time bi lanac bio prekinut.

Pokazivač na glavu

Git posjeduje pokazivač HEAD koji pokazuje na određenu predaju u repozitorijumu. Kako pravimo predaje tako se i pokazivač na glavu mijenja odnosno pomjera na narednu predaju. Pokazivač na glavu uvijek pokazuje na vrh trenutne grane u repozitorijumu. O pokazivaču na glavu brine sam Git te ćemo njemu pristupati veoma rijetko, ali vrijedi znati šta Git radi za nas. Uočimo da u folderu .git postoji fajl koji se zove HEAD. Pogledajmo šta se u njemu nalazi:

```
PS C:\Users\skovacevic\Documents\Java\projekat> cat .git/HEAD
ref: refs/heads/master
```

Pogledajmo na šta to referencira glava:

```
PS C:\Users\skovacevic\Documents\Java\projekat> cat .git/refs/heads/master
1cd2a4b89be37ea0b18a8aab585673cdeb6f2ecd
```

Kao što vidimo, radi se o heš vrijednosti odnosno ID-u prijave. Koje? Sjetimo se komande git log.

Dodavanje fajlova I

Prije negoli se detaljno upoznamo sa git add naredbom, naučimo sljedeću:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
nothing to commit, working tree clean
```

koja nam prikazuje trenutno stanje radne kopije i pripremnog indeksa. Iz priložene poruke vidimo da je radno drvo čisto i da nema izmjena za predaju. Napravimo sada još dva fajla u folderu projekta, fajl2.txt i fajl3.txt i upišimo u njih proizvoljan tekst. Ponovimo prethodnu naredbu:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        fajl2.txt
        fajl3.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Dodavanje fajlova II

Naši fajlovi označeni su kao untracked odnosno ne prate se. To znači da ovi fajlovi nisu u repozitorijumu, Git ne zna ništa o njima i da dodatne izmjene koje napravimo nad njima neće biti praćene. Takođe, Git nam daje sugestiju da fajl prebacimo u pripremno drvo. Uradimo to za jedan od dva:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git add fajl2.txt
```

Pa ponovo provjerimo stanje:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file: fajl2.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        fajl3.txt
```

Dodavanje fajlova III

Vidimo da je Git prebacio drugi fajl u pripremno drvo i govori nam kako da ga pomoću git commit predamo repozitorijumu. Predajmo fajl:

```
PS C:\Users\skovacevic...> git commit -m "Add second file to the project"
[master 86ac89f] Add second file to the project
 1 file changed, 1 insertion(+)
 create mode 100644 fajl2.txt
```

Vrijednost 86ac89f je skraćena SHA vrijednost koju je Git dodijelio ovoj predaji. Ponovimo sada proces i za treći fajl. U mom slučaju, skraćena hex SHA vrijednost njegove predaje je dd4bb43. Očekujemo da glava repozitorijuma pokazuje na ovu predaju jer je poslednja.

Dodavanje fajlova IV

Pogledajmo sada kompletan log predaja:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git log
commit dd4bb430a105f12b3632fc9c6569dd8bf98a29f2 (HEAD -> master)
Author: Slavko Kovacevic <skovacevic@ucg.ac.me>
Date: Sat Feb 25 10:16:49 2023 +0100

    Add third file to the project

commit 86ac89f4bce099bb12046c6c04e284b5a3be6752
Author: Slavko Kovacevic <skovacevic@ucg.ac.me>
Date: Sat Feb 25 10:12:55 2023 +0100

    Add second file to the project

commit 1cd2a4b89be37ea0b18a8aab585673cdeb6f2ecd
Author: Slavko Kovacevic <skovacevic@ucg.ac.me>
Date: Fri Feb 24 15:34:31 2023 +0100

    Initial commit
```

Ažuriranje I

Otvorimo prvi, pa drugi fajl i napravimo nekoliko izmjena u njima. Provjerimo stanje:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
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: fajl.txt
        modified: fajl2.txt

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

Za razliku od dodavanja fajlova, gdje je pisalo da su oni nepraćeni, sada su oni spremni za pripremu odnosno Git nam daje do znanja da ih prati. Drugim riječima, verzija fajla u našem radnom drvetu nije ista kao ona u pripremnom drvetu ili repozitorijumu. Proces dodavanja ovih izmjena identičan je procesu dodavanja novih fajlova. Sjetite se da Git prati cjeline izmjena, a ne same verzije fajlova.

Ažuriranje II

Ukoliko želimo dodati više fajlova odjednom možemo to uraditi navođenjem jednog pa drugog fajla po imenu, a ukoliko želimo dodati sve izmjene odjednom, radimo to pomoću tačke:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git add .
PS C:\Users\skovacevic\Documents\Java\projekat> git add fajl.1txt fajl2.txt
```

Provjerimo stanje:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified: fajl.txt
        modified: fajl2.txt
```

Vidimo da su naša dva fajla u pripremnom drvetu, spremna da se predaju repozitorijumu. Izmijenimo sada treći fajl.

Ažuriranje III

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified: fajl.txt
        modified: fajl2.txt

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: fajl3.txt
```

Ne zaboravite da uvijek radimo s tri drveta. Prva dva fajla su u pripremnom drvetu, dok je treći u radnom. Predajmo sada prvi i drugi fajl pomoću git commit. Sada su izmjene nad dva fajla zajedno predate repozitorijumu. Sami ponovite proces za treći fajl.

Pregled razlika I

Pomoću komande git status možemo se informisati koji je fajl izmijenjen, ali ne i šta je u njemu izmijenjeno. Nerijetko su izmjene suptilne, a kada radimo s fajlovima koji imaju stotine linija koda, nije lako ustanoviti šta je u određenoj predaji promijenjeno.
Otvorimo prvi fajl i dodajmo jedan prazan red i dvije kratke rečenice.
Provjerimo sada razlike:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git diff
diff --git a/fajl.txt b/fajl.txt
index 8df2f24..bef214b 100644
--- a/fajl.txt
+++ b/fajl.txt
@@-1 +1,4 @@
-Prvi fajl projekta.
\ No newline at end of file
+Prvi fajl projekta.
+
+Nova linija.
+Još jedna nova linija.
\ No newline at end of file
```

Pregled razlika II

Jasno vidimo koji je fajl izmijenjen i šta su izmjene odnosno šta smo oduzeli, a šta dodali u fajl. Prva linija fajla naizgled nije mijenjana, ali stoji da je obrisana. Naime, na njenom kraju dodat je karakter za novu liniju pa je ona ponovo dodana. Git prati izmjene na nivou linije. Upozorenje $\mathrm{No\ newline\ at\ end\ of\ file}$ je na mjestu i govori nam da se naš fajl ne završava karakterom za novu liniju što je loša praksa. Zavisno od tekst editora koji koristite, ova poruka može ali i ne mora biti prisutna. Napravimo sada izmjenu i na drugom fajlu, a onda nakon toga dodajmo prvi fajl u pripremno drvo pomoću $\mathrm{git\ add}$. Sada, kada bismo izvršili pregled razlika, Git bi nam prijavio izmjene isključivo na trećem fajlu, odnosno na fajlovima koji su u radnom drvetu.

Pregled razlika III

Da prikažemo razlike koje su u pripremnom drvetu kucamo:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git diff --staged
```

Drugim riječima, komanda git diff izlistava pregled razlika između radnog i pripremnog drveta dok git diff –staged evidentira razlike između pripremnog drveta i repozitorijuma. Moramo uvijek imati na umu da Git arhitektura funkcioniše na osnovu tri drveta!

Brisanje I

Napravimo dva prazna fajla za brisanje, brisanje1.txt i brisanje2.txt. Ukoliko provjerimo stanje pomoću git status vidjećemo da se radi o fajlovima koje Git ne prati, o čemu je ranije bilo riječ. Ukoliko sada obrišemo jedan od dva fajla, on bi naprosto nestao - Git ga nije ni pratio. Međutim, umjesto brisanja, prebacimo ih u pripremno drvo pomoću git add . i uvjerimo se da je sve u redu:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file: brisanje1.txt
        new file: brisanje2.txt
```

Predajmo ih repozitorijumu upotrebom naredbe git commit -m "Add files that will be deleted".

Brisanje II

Prvi način da obrišemo fajl je da ga naprosto obrišemo ili pomjerimo iz projekta. Kako on više nije tu dobićemo sljedeći izvještaj:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted: brisanje1.txt

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

Što znači da je fajl obrisan u radnom drvetu. Brisanje je izmjena i da bismo predali ovu izmjenu repozitorijumu prvo je potrebno proslijediti pripremnom stablu. To radimo pomoću sljedeće naredbe:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git rm brisanje1.txt
rm 'brisanje1.txt'
```

Sada možemo izmjenu predati repozitorijumu.

Brisanje III

Da bi se fajl obrisao nije potrebno da ga ručno brišemo. Git će to uraditi za nas. Korak brisanja i pripremanja mogu se spojiti komandom:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git rm brisanje2.txt
rm 'brisanje2.txt'
```

Ovaj fajl nije u kanti, nije nigdje pomjeren, obrisan je trajno. Informacija o njegovom brisanju nalazi se u pripremnom drvetu. Međutim, kada ovu izmjenu predamo repozitorijumu, fajl brisanje2.txt će u formi predaje i dalje biti prisutan i moći ćemo ga vratiti, Git će ga zauvijek (nismo skroz iskreni) čuvati.

Premještanje i preimenovanje I

Kod preimenovanja fajlova važe slična pravila kao i kod brisanja. Promijenimo ručno naziv prvog fajla, fajl.txt u promijenjen.txt i posmatrajmo situaciju:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted: fajl.txt

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

Vidimo da Git ne veže ova dva događaja već nam kaže da smo jedan fajl obrisali, a da je drugi fajl nepraćen jer je tek napravljen iako mi znamo da se radi o jednom te istom fajlu.

Premještanje i preimenovanje II

Dodajmo sada ove izmjene u pripremno drvo pomoću git add i git rm. Provjerom stanja:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed: fajl.txt -> promijenjen.txt
```

Uočavamo da je Git sada zaključio da se radi o istom fajlu i obavještava nas da smo izvršili promjenu naziva. Da bismo razumjeli preimenovanje fajlova pomoću Git-a moramo znati da u Unix sistemima nije moguće preimenovati fajl, moguće ga je samo premjestiti. Međutim, premještanjem se fajl može i preimenovati.

Premještanje i preimenovanje III

Prethodnu tvrdnju potkrijepimo primjerom. Komandom:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git mv fajl2.txt drugi_fajl.txt
```

fajl fajl2.txt premještamo u drugi_fajl.txt čime mu mijenjamo ime. Ukoliko otvorimo folder vidjećemo da se u radnom drvetu ova promjena evidentirala što će nam potvrditi i Git:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed: fajl2.txt -> drugi_fajl.txt
        renamed: fajl.txt -> promijenjen.txt
```

Ne zaboravimo da predamo ove izmjene repozitorijumu.

Poništavanje izmjena I

S vremena na vrijeme dešavaće nam se da napravimo određenu izmjenu na fajlu u radnom drvetu s kojom nismo zadovoljni. Ukoliko smo fajl zatvorili i sačuvali, ne postoji jednostavan način da vratimo pređašnje stanje. Međutim, Git zna kako je taj fajl izgledao prije izmjene jer tu informaciju posjeduje u repozitorijumu. Otvorimo naš promijenjen.txt i obrišimo dio teksta, sačuvajmo izmjenu i zatvorimo fajl. Uvidom u stanje:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
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: promijenjen.txt

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

uočavamo da nam Git sam govori šta da uradimo:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git restore promijenjen.txt
```

Poništavanje izmjena II

Šta se dešava ukoliko je izmjena proslijeđena pripremnom drvetu? Ponovimo eksperiment ponovo. Izmijenimo fajl3.txt, dodajmo ga u pripremno stablo. Uvidom u stanje:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified: fajl3.txt

PS C:\Users\skovacevic\Documents\Java\projekat>
```

vidimo da nam Git opet daje savjet kako da povratimo staru verziju:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git restore --staged fajl3.txt
```

čime je problem riješen.

Poništavanje izmjena III

Naučili smo da je zbog integriteta podataka nemoguće mijenjati prethodne predaje. Ukoliko želimo da vratimo neku staru vrijednost, dobra praksa je da napravimo novu izmjenu koja će novo stanje vratiti na staro. Pa kako da to uradimo? Sjetimo se da ćemo pomoću naredbe $\mathrm{git\ log}$ vidjeti sve prethodne predaje. Nađemo onu koja nam je od interesa (prva predaja), uzmemo prvih nekoliko karaktera njene SHA vrijednosti i kucamo:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git checkout 1cd2a4b -- fajl.txt
```

Sada, predavanjem ove verzije repozitorijumu zapravo poništavamo najnoviju verziju starijom.

Poništavanje izmjena IV

Ukoliko želimo da čitavu jednu predaju poništimo, to možemo uraditi pomoću komande git revert ID nakon čega će se otvoriti Vim editor:

```
Revert "Delete second file for deletion"

This reverts commit 1b056a4b2ed5c977901e208d1c8c88697d8f747c.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
# new file: brisanje2.txt
#
```

Poznavanje Vim-a je van opsega ovog priručnika, ali sve što treba znati jeste da u prvoj liniji imamo generisanu poruku predaje (zadovoljavajuću) i da editor možemo zatvoriti klikom na Esc, a onda smo dužni iskucati :q i kliknuti Enter.

Uklanjanje nepraćenih fajlova

Svaki novi fajl koji napravimo je po pravilu nepraćen, ali ga možemo dodati. Takav fajl ne možemo ukloniti pomoću komande git rm zato što Git o njemu ne vodi računa. Napravimo fajl junk.txt. Pokušajmo ga obrisati:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git clean
fatal: clean.requireForce defaults to true and neither -i, -n, nor -f given;
refusing to clean
```

Kako se radi o destruktivnoj operaciji, Git nam izdaje upozorenje. Ne interesuje nas pretjerano git clean -i jer to predstavlja interaktivno čišćenje, korak po korak. Dobra je praksa da prvo uradimo:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git clean -n
Would remove junk.txt
```

jer ova naredba predstavlja takozvani dry run, šta bi bilo kad bi bilo. Sada, kada znamo šta su posljedice, pokrenimo naredbu za čišćenje:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git clean -f
Removing junk.txt
```

Ignorisanje fajlova I

Git se koristi za praćenje fajlova, ali nisu svi fajlovi podesni za praćenje. Veliki fajlovi kao što su slike, snimci, reproducibilni rezultati skripti, logovi, iskompajlirani kod, lokalna podešavanja IDE itd. ne smiju završiti u centralnom repozitorijumu. Mi nismo u obavezi da te fajlove dodajemo u pripremno drvo, ali oni će nam smetati pri radu praveći gužvu. Pa kako da kažemo Git-u da ih se otarasi? Prvo treba napraviti fajl u projektu i dati mu ime .gitignore. Sadržajem ovog fajla (koji predstavlja listu pravila) Git-u sugerišemo koje fajlove da ignoriše.

Ignorisanje fajlova II

Ignorisati fajlove možemo upotrebom sljedećih pravila:

- konkretan fajl dodajemo samo po imenu npr. fajl.txt
- fajlove možemo ignorisati pomoću regularnih izraza
- konkretan fajl možemo izuzeti iz pravila uz pomoć znaka uzvika
- čitav folder možemo ignorisati sa direktorijum/ čime se sve nakon / ignoriše

Kolekciju gitignore šablona možete naći na:
https://github.com/github/gitignore

Ignorisanje fajlova III

Šta ako želimo da ignorišemo promjene nad fajlom koji je Git do tada pratio? Dodajmo jedan od fajlova koje Git prati u .gitignore. Komanda git status reći će nam da je .gitignore promijenjen. Međutim, ukoliko izvršimo izmjenu fajla od interesa, imamo sljedeću situaciju:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git status
On branch master
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: .gitignore
        modified: promijenjen.txt

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

što je neočekivano, Git i dalje prati ovaj fajl bez obzira što smo ga upisali u .gitignore.

Ignorisanje fajlova IV

Da natjeramo Git da u potpunosti zaboravi na fajl koji je prethodno pratio iskoristićemo sljedeću naredbu:

```
PS C:\Users\skovacevic\Documents\Java\projekat> git rm --cached promijenjen.txt
rm 'promijenjen.txt'
```

čime fajl uklanjamo iz pripremnog drveta, ali ne i iz radne kopije. Zanimljivo, git status reći će nam da Git misli da će fajl biti obrisan, ali kad odradimo predaju repozitorijumu on i dalje ostaje na svom mjestu - sve što se dogodilo jeste da je Git prestao da ga prati.

Zaključak

Ovaj poslednji slajd predstavlja kraj početka upoznavanja s Git-om. Premda smo se upoznali s osnovama Git-a, ostalo je još dosta toga da o njemu naučimo. Međutim, znanje iz softverskog inženjerstva najbolje se stiče praksom. Znanje koje se aktivno ne upotrebljava brzo se i zaboravi. Zato vas ohrabrujemo da sa ovim osnovama (koje su sasvim dovoljne da vršite kontrolu verzije na nekom projektu) krenete sa poslom i suočavate se sa realnim, a ne hipotetičkim problemima, samo tako postaćete dobri inženjeri.