Version Control i Git kao primjer Version Control sustava

Tadić, Josip

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike

Josip Tadić

VERSION CONTROL I GIT KAO PRIMJER VERSION CONTROL SUSTAVA

Završni rad

Pula, 2019.

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike

Josip Tadić

VERSION CONTROL I GIT KAO PRIMJER VERSION CONTROL SUSTAVA

Završni rad

JMBAG: 60198321030306140002, redoviti student

Studijski smjer: Informatika

Predmet: Programsko inženjerstvo

Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija

Mentor / Mentorica: doc. Dr. Sc. Tihomir Orehovački

Pula, prosinac 2019.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani _____, kandidat za prvostupnika ekonomije/poslovne ekonomije, smjera _____ ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA

o korištenju autorskog djela

Ja, _____ dajem odobrenje Sveučilištu Jurja Dobrile

u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom

_____

_____

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____ (datum)

Potpis

_____

Sažetak

U ovom završnom radu obrađena je tema kontrole verzioniranjem. Kontrola verzioniranjem ili "version control" je metoda razvijanja računalnih programa i aplikacija, no ne koristi se nužno samo u tu svrhu. Ova metoda omogućila je razvoj mnogih današnjih sustava i računalnih programa. Osim same kontrole verzioniranjem objašnjeni su sustavi i alati koji olakšavaju korištenje ovom metodom. Važnost ovih sustava i njihovih alata te njihove podjele, način primjene i njihove mogućnosti detaljnije su pojašnjeni u radu. Iako u radu ima više navedenih alata za kontrolu verzioniranjem kao primjer sustava odabran je Git. Git kao primjer je odabran iz razloga što ima najširu primjenu i jedan je od najrobusnijih alata za kontrolu verzioniranjem. Git kao sustav ima daleko najveću zajednicu, najveći broj aktivnih korisnika i projekata, te je najčešći izbor alata za kontrolu verzioniranjem unazad zadnjih nekoliko godina. Git se gotovo smatra standardom kontrole verzioniranjem.

Ključne riječi: kontrola verzioniranjem, verzija, sustavi kontrole verzioniranjem, Git

Summary

In this final assignment, the topic of version control is addressed. Version control or source control is a method of developing computer programs and applications, but is not necessarily used solely for that purpose. This method has enabled development of many of today's systems and computer programs. In addition to version control itself, systems and tools that make this method easier to use are explained. The importance of these systems and their tools, as well as their classification, their applications and their capabilities are also explained in the assignment. Although there are several listed versioning tools and systems in the assignment, Git was chosen as an example of the system. Git was chosen as an example because it has one of the widest application capabilities and is one of the most robust version control tools available. Git as a system has by far the largest community, the largest number of active users and projects, and is the most common choice of version control tools over the last few years. Git is almost considered the standard of versioning control.

Keywords: version control, version, version control systems, Git

SADRŽAJ

1. UVOD

Premda su se prvi oblici version control sustava javili još 80-tih godina prošlog stoljeća, pravi porast broja korisnika zabilježen je unazad zadnjih 10-tak godina. Sam rast broja korisnika uvelike je potaknut rastom informatičke zajednice. Osim rasta zajednice tehnologija je danas dostupnija u višoj mjeri nego što je to bilo 80-tih godina prošloga stoljeća. Današnji version control sustavi ne zahtijevaju mnogo računalnih resursa te ukoliko ne uključuju velik broj multimedijskih datoteka rade gotovo besprijekorno. Današnji poslovi koji se vežu uz računalno programiranje, i donekle uz baze podataka, poboljšali su svoju efikasnost zahvaljujući ovim sustavima. Mnoge aplikacije i mnogi sustavi razvijeni su uz pomoć version control sustava. Današnji poslovi koji su vezani uz računalno programiranje u većoj mjeri zahtijevaju poznavanje načina rada version control sustava. Neovisno o kojem se programskom jeziku radi korisnici se služe version control sustavima. Osim porasta u popularnosti i važnosti poznavanja ovih sustava pri zapošljavanju, u porastu su funkcije i mogućnosti koje pružaju version control sustavi. Uz porast funkcionalnost nažalost raste i kompleksnost samih sustava, te prvi susret sa version control sustavom iz tog razloga može biti malo zastrašujuć. Iako korisnicima treba više vremena da se priviknu na ovakav način rada, ali njegovo učenje itekako se isplati. Jednom kada se počnu služiti version control sustavima ubrzo shvaćaju svu njegovu moć. Version control sustavi po svojoj prirodi korištenja također povećavaju suradnju timova, i samih korisnika unutar timova. Osim povećanja suradnje unutar timova i osoblja unutar timova, Version control sustavi povećavaju kvalitetu rada, svaki od korisnika može vidjeti rad svojih kolega, te njihovu povijest promjena, što je u računalnom programiranju vrlo bitno pogotovo u aspektu stabilnosti. Osim u profesionalnom svijetu rada version control sustavi odlično služe za vježbu budućih računalnih programera, a njihovo znanje uvelike povećavaju šansu za zapošljavanje. Velika zajednica korisnika spaja ljude iz profesionalnog svijeta, sa osobama koje tek počinju svoje karijere u tehnološkoj industriji. Na ovaj način sudionici zajednice imaju odličnu priliku za vježbu, te imaju priliku raditi sa alatom koji se koristi u većini tehnoloških poduzeća. Jedan od razloga zašto su version control sustavi toliko korišteni su

njegove široke primjene. Git kao version control sustav daleko prednjači po broju korisnika iako je sintaksom znatno kompliciraniji od svojih konkurenata. Ispred početnika koji se tek upoznavaju sa ovakvim principom rada preostaje odluka sa kojim će sustavom raditi. Git kao sustav je robusniji od svojih konkurenata, a svoje funkcionalnosti je širio kako se širila potreba i rast popularnosti version control sustava.

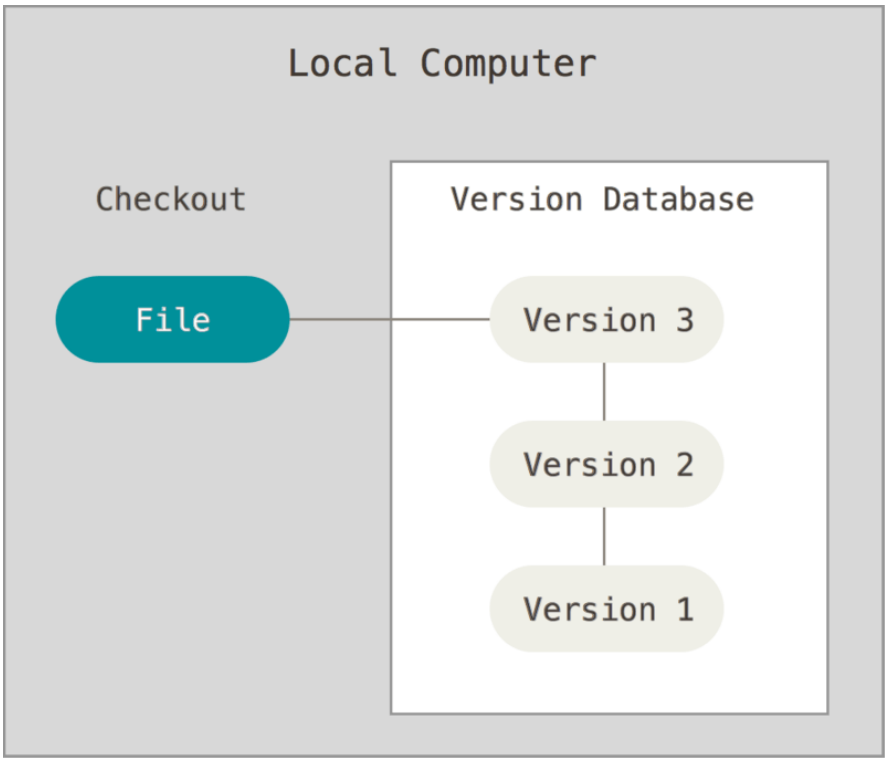
2 Version control

2.1 Uvod u Version control

Što je zapravo "kontrola verzije" (eng. Version control) i iz kojih je razloga nama to bitno. Kada i zašto se uopće javila potreba za version controlingom? Pojam version control se pojavio kao potreba za smislenim sustavom organiziranja. Version control postoji u primitivnim oblicima otkako i samo pisanje, ali danas se posebno ističe pri razvijanju informacijskih tehnologija. Primitivniji oblici koji se čak i danas koriste su numeracije izdanja knjiga, dok najviše koristi od version controlinga imaju računalni programeri. Mogućnosti koje pruža version controlling su mnogobrojne, te neke od njih su od presudne važnosti računalnim programerima, najvažnije značajke version controllinga omogućuju ljudima da nesmetano rade istovremeno na istom dokumentu ili razvoju programa, da prate njegov razvoj kroz vrijeme, da se točno vidi tko je što radio, te da u slučaju da se nešto dogodi sve skupa može vratiti na prijašnju ili neku od verzija koja nam odgovara.

Kontrola verzija je sustav upravljanja i pregledavanja informacija koji se često koristi za praćenje razvijanja programskog koda, dokumenata, računalnih programa(najčešće razvijanja internetskih stranica i njihovih sustava, te mobilnih aplikacija). Način na koji version control prati tijek razvijanja jest da tijekom vremena bilježi sve promjene u datoteku ili skup datoteka, tako se u budućnosti uvijek možete vratiti određene verzije.

Primitivni način ovakvog sustava je moguće primijeniti za bilo kakav dokument, a one se jako lako primjenjuju, pogotovo ako ih radimo za vlastiti development, gdje radi samo jedna osoba. Sve što treba je svaku verziju koju smo napravili spremiti, napraviti njenu kopiju zapisati vrijeme spremanja i verziju, i kako bi si malo olakšali nekakav opis. Najveća prednost ovakvog primitivnog sustava, je u tome što u svakom trenutku možemo krenuti ponovno od nekih prijašnjih verzija. Ovakav način primjene nema smisla često verzionirati, nego tek nakon nekih većih promjena ili dodavanja funkcionalnosti i testiranja. U slučaju puno verzija dokumenti mogu postati vrlo nepregledni i na taj način trošimo puno vremena te samim time ovaj sustav gubi smisao.



Slika 1: Lokalni version control sustav

Izvor: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Ljudi koji rade u timovima, a ne koriste neki od oblika kontrole verzija, često mogu stvoriti probleme. Jedan od većih problema koji nastaju ne korištenjem version controla je to što nitko od ljudi koji rade na projektu ne znaju točno koje su izmjene napravljene, što samo po sebi ima puno potencijala za stvaranje novih problema: konflikti (više promjena istoga djela programskog koda), neke od promjena na drugim dijelovima programskog koda može utjecati na funkcionalnost cjeline, te bi u većini slučajeva bi imali više dokumenata od kojih ne bi znali koji je konačan, a ručno pregledavanje bi bilo vrlo mukotrpno. U današnje vrijeme razvijanja software-a, rad bez neke vrste verzioniranja gotovo je nezamisliv i djelomično besmislen pogotovo za projekte na koje su priključeni veći brojevi učesnika.

Za upravljanje kontrolom verzija postoje već gotovi sustavi koji uvelike olakšavaju version controling, ali u svakome od njih princip je sličan. Napravimo inicijalnu instancu programa koja ima verziju 1.0, te nakon toga na zahtjev dodajemo promjene, pa slijedi verzija 1.1. Nakon verzije 1.1 trebamo napraviti još promjena funkcionalnosti pa slijedi verzija 1.2. U slučaju da nešto prestane raditi nakon detaljnijeg testiranja uvijek možemo vratiti na prijašnju

verziju(u ovom slučaju 1.1), te nastaviti dalje od te verzije svjesni svojih pogrešaka, te ih ispraviti.

2.2 Sustav za kontrolu verzija

Sustav za kontrolu verzija (eng. VCS - Version Control Systems) je naziv za software-ske alate koji nam uvelike olakšavaju upravljanje i korištenje version controllinga. posljednjih nekoliko desetljeća napretkom tehnologija i napretkom tehnika programiranja u Sustavima za kontrolu verzija došlo je do velikih napredaka, a neki alati su više zaživjeli od svojih konkurenata. Sustavi kontrole verzija (VCS) se u većini slučajeva pokreću kao samostalni alat, ali Sustavi kontrole verzija također mogu biti ugrađeni u razne vrste programskih alata, kao što su na primjer: uređivači teksta i proračunske tablice, razni web dokumenti, i te u razne sustave za upravljanje sadržajem(eng. CMS – Content management system), koji onda prati cijelu njegovu povijest. VCS je također poznat i kao SCM (eng. Source Code Management) alat ili kao RCS (eng. Revision Control System). Veći dio popularnijih VCS sustava koji su danas dostupni su otvorenog koda, i besplatni za sve vrste upotreba. Sustavi i alati za upravljanje kontrolom verzija mogu se podijeliti u dvije skupine: Centralizirani VCS i Distribuirani VCS.

2.3 Centralizirani sustavi za “version control”:

Prve potrebe za sustav kontrole verzijama su se javile 1970-tih. Prvi izumljeni sustavi su bili Centralni sustavi za kontrolu verzijama. Zašto baš centralizirano? Jer je jednostavno, a ono što je prvo bilo potrebno, jedno mjesto svatko može spremiti, prijaviti i provjeriti razlike. Centralizirani sustavi za kontrolu verzija se temelje na središnjoj ili glavnoj kopiji dokumenata. Taj glavni dokument ili skupina dokumenata može se nalaziti bilo gdje, ali zbog opasnosti od gubitka podataka i sigurnosti poželjno bi bilo negdje na nekom internetskom poslužitelju. Dakle svi koji rade na ovakvom sustavu, svoje

promjene izvršavaju direktno na glavnoj kopiji dokumenta. Na taj način uvijek imamo jednu vremensku liniju koja bilježi sve promjene spremljene na glavnu kopiju. Ovakav način upravljanja verzijama nešto je jednostavniji od distribuiranog, ali zasigurno ima svoje i nedostatke u odnosu na distribuirane sustave.

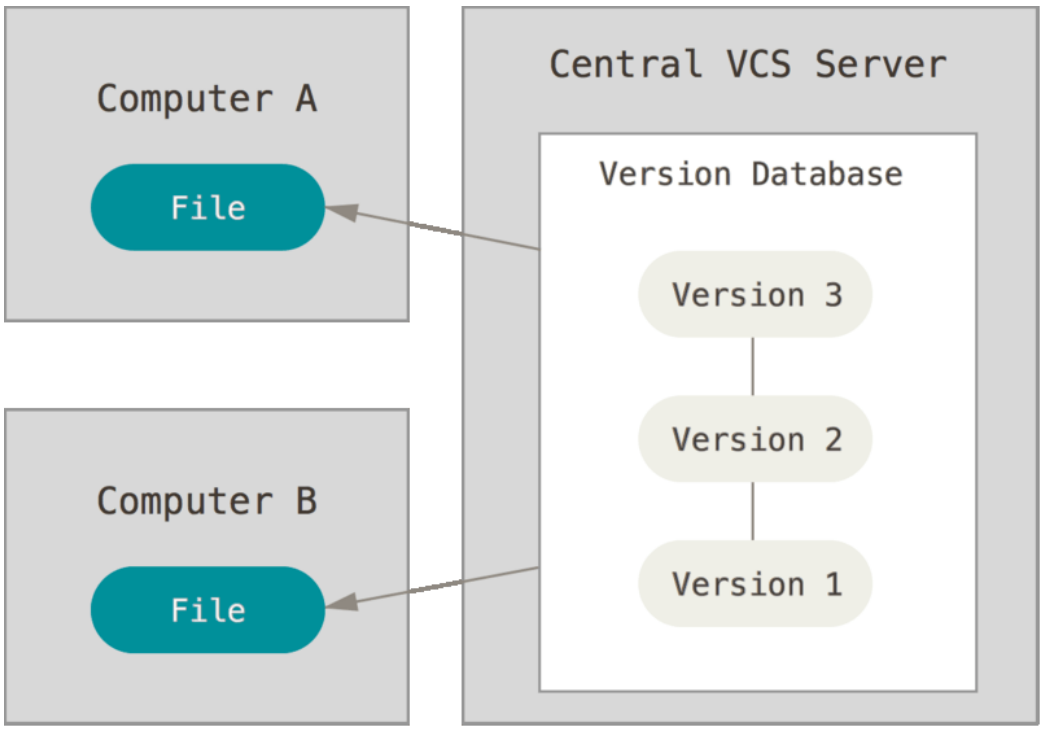
Promjene se jednostavno bilježe na glavnoj kopiji dokumenta. Ostali sudionici nakon bilježenja mogu vidjeti nastalu promjenu, ali ju nemaju kod sebe. Ostali sudionici također mogu "povući" zadnju verziju, a sustav za kontrolu verzija automatski će ažurirati sav promijenjen sadržaj bilo kojeg promijenjenog dokumenta. Iako je u današnje vrijeme rijedak slučaj da se radi samo na jednom dokumentu, pogotovo u timovima sa više ljudi. Većina suvremenih sustava za upravljanje verzijama se bavi promjenama više dokumenata od jednom, to jest prati skupinu promjena. Sve promjene se u ovom i u većini drugih slučajeva tretira zajedno kao skup. Na primjer ako se radi o zaglavljima programskog dokumenta sustav posebno treba paziti na odgovarajuću datoteku kojoj zaglavlje pripada.

Centralizirani version control rješava većinu problema opisanih u prijašnjim poglavljima. Programeri(ali ne isključivo samo programeri) više nemaju potrebu zadržavati razne kopije dokumenata na svojim diskovima ili udaljenim servisima, jer alati za kontrolu verzija prate glavnu (središnju) kopiju i mogu dohvatiti zadnje dostupnu verziju, ili bilo koju verziju koja im je potrebna tokom rada.

Prilikom rada sa centraliziranim sustavom za upravljanje verzijama, procedura rada za dodavanje ili izmjenu značajki ili pak ispravljanje grešaka u dokumentu uobičajeno prati slijedeće korake:

1. Preuzeti sve promjene sa glavnog poslužitelja, napravljene od strane drugih sudionika (takozvana "pull" naredba)
2. Provjerite promjene i uvjerite se da funkcioniraju ispravno.
3. Vaše promjene proslijedite na središnji poslužitelj kako bi ih drugi sudionici mogli preuzeti i provjeriti (takozvana "push" naredba)

Ovakav postupak je pojednostavljen način rada sa alatima za kontrolu verzioniranjem, u radnom okruženju ne mora značiti da neće nikada doći do problema. Većina problema koji su već spomenuti u prijašnjim poglavljima se uglavnom javljaju zbog nepažnje ili loše komunikacije među sudionicima timova ili loše komunikacije sa nadređenima. U takvim slučajevima većina alata odrađuje svoj dio posla, na način da onemogućuju spremanje promjena dok se ne riješe svi konflikti, to jest dijelovi programskog koda koji su izmijenjeni od više sudionika.



Slika 2: Centralizirani version control sustav

Izvor: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Neki od poznatijih centraliziranih sustava za version control su:

1. CVS – Jedan od prvih sustava za version control
2. SVN
3. Perforce

Međutim, Centralizirani version control ima i ozbiljne nedostatke. Najuočljiviji problem je što postoji jedan oslonac koji lako može postati točka neuspjeha, a nju predstavlja centralizirani poslužitelj. U slučaju da taj poslužitelj nije dostupan neko vrijeme, tijekom tog vremena nitko ne može surađivati, pa tako ni spremati ispravne promjene na bilo čemu trenutno rade. Ako se memorija na kojem je pohranjena središnja baza podataka ošteti, a nisu spremljene odgovarajuće sigurnosne kopije, postoji mogućnost da se izgubi apsolutno sva povijest projekta ili pak dio cijelog projekta. Onog što bi u tom trenutku preostalo su pojedinačni projekti ili njegovi dijelovi na lokalnim uređajima sudionika.

2.3 Distribuirani sustavi za "version control":

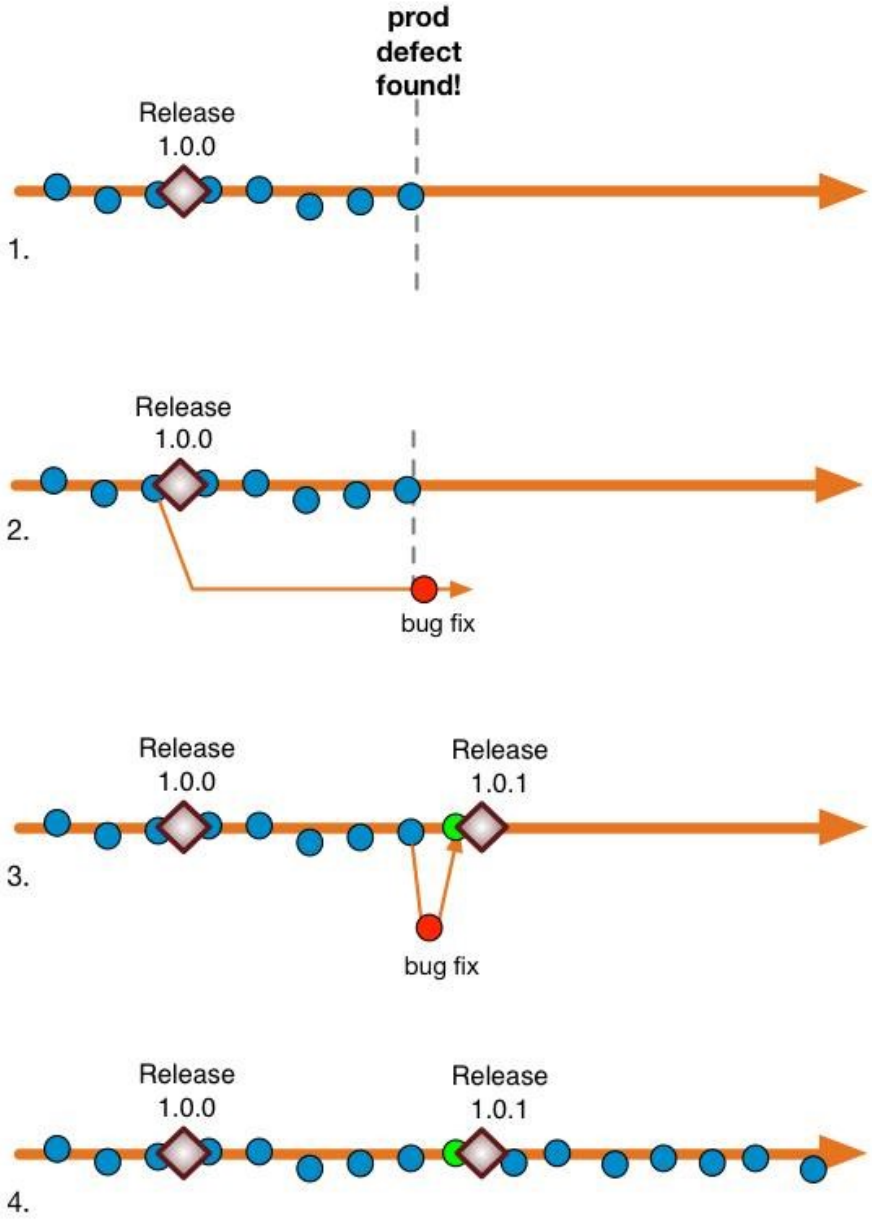
Iako u raznim sustavima za version control terminologija može u manjoj mjeri razlikovati, prije pojašnjavanja pojma Distribuiranih sustava za "version control" potrebno je razjasniti par pojmova i naredbi kako bi lakše razumjeli cjeli koncept, a oni su:

2.3.1 Osnovni pojmovi

1. "Baseline" ili "trunk":

Glavna grana developmenta(dokumenta), koja je jedinstvena, te sve ostale izmjene kreću sa nje (eng. Master branch) U kontekstu razvijanja softvera baseline ili trunk se odnosi na niz izdanih stabilnih verzija razvoja, te se na njih oslanja tijek razvijanja softvera. Jedan od pristupa koji se redovito koristi u praksi je odvojiti granu (eng. Branch) na kojemu se tada mogu testirati i implementirati promjene. Nakon testiranih promjena grana se spaja nazad na Baseline ili Trunk, te na taj način tijek razvoja na glavnoj grani tijeka razvijanja softvera. Vrh glavne grane u pravilu predstavlja zadnju stabilnu verziju koju nazivamo glavom (eng. "Head" ili "Tip"). U slučaju kada bi programeri radili samo na jednoj liniji tijeka razvijanja koja bi ujedno u tom slučaju bila glavna grana, svaka promjena koja bi bila pohranjena bi predstavljala glavu. Zbog

takvog načina rada češće se događa da glava glavne grane bude nestabilna verzija. Ovakva praksa nije efikasna jer upravo iz tog razloga postoji grananje i spajanje.

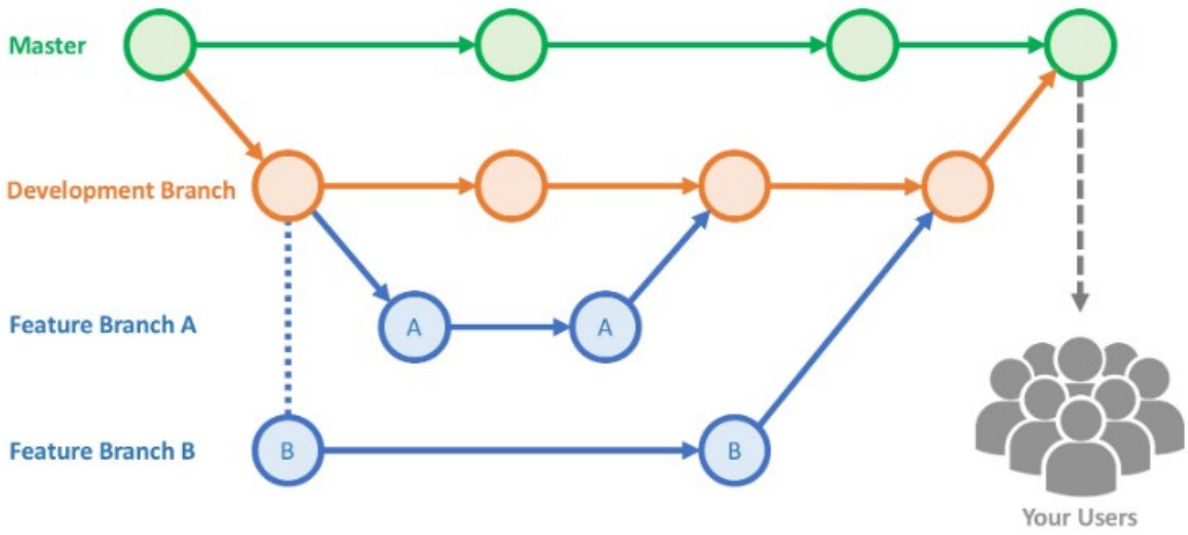


Slika 3: Grafički prikaz tijeka razvoja na glavnoj grani razvoja

Izvor: https://paulhammant.com/2014/09/29/shades-of-trunk-based-development/

2. “Branch”:

Grana development nezavisna od glavne, ali je njena kopija u trenutku nastajanja. Teoretski ih može biti beskonačno. Način na koji je korištenje Brancheva zamišljen je sljedeći: Programeri glavnu granu developmenta kopiraju te stvaraju svoju granu(eng. Branch) iz koje onda svatko od njih može nastaviti raditi što je zamislio. Ova mogućnost je jako moćna jer omogućuje svakome od programera da nesmetano rade na svom dijelu projekta ili modula. Svaki od brancheva se može individualno testirati nakon implementacije, što znatno olakšava tijek razvijanja softvera. Ovakav način “paralelnog” razvoja izuzetno je koristan u slučajevima gdje su timovi sastavljeni od većeg broja različitih stručnjaka koji rade na projektu. Programeri nakon razvijanja softvera mogu poslati nove funkcionalnosti kolegama koji će testirati nove promjene, a ako se potvrdi validnost promjena, one će se spojiti(eng. Merge) nazad na glavnu granu razvoja.

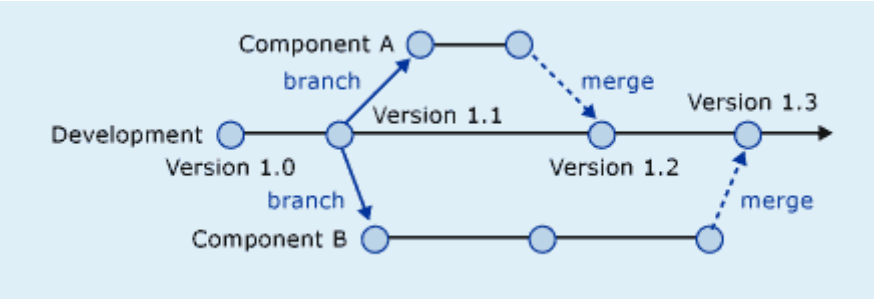


Slika 4: Grananje i master branch

Izvor:https://github.com/launchdarkly/featureflags/blob/master/6%20-%20Flags%20vs%20Branching.md

3. “Merge”:

Merge-ati bi u prijevodu značilo sjediniti ili spojiti. Pojam merge koji još možemo nazivati i integracijom, nebi postojao bez grananja. Merganje ili spajanje je funkcija version control sustava koje omogućuje spajanje više grana. Promjene sa dvije grane se spajaju u jednu, a prilikom spajanja bitno je detaljno obaviti provjere kako ne bi došlo do konflikta koji bi mogli zaustaviti spajanje i na taj način bi mogli stvoriti dodatne probleme.



Slika 5: Spajanje grana sa “merge” funkcijom

Izvor: https://blog.codinghorror.com/software-branching-and-parallel-universes/

4. “Commit”:

Commit znači prijava ili spremanje promjena u radnoj kopiji, što znači da u trenutku commit-a, na našoj grani naš commit postaje glava te grane. Nakon commit-a pomoću naredbi “update” ili “checkout” možemo se vratiti na prijašnje commit-ano stanje ukoliko se nađemo u neželjenoj situaciji. Uz commit-anje preporučeno je naznačiti koje su promjene napravljene kao bi se lakše mogli organizirati i kako bi ostali sudionici projekta mogli pregledati commit-ane promjene.

5. “Pull”:

Pull-ati znači sve spremljene (push-ane) promjene preuzeti sa nekog od branch-eva. Pull se koristi kada netko od sudionika na projektu commit-a promjene, a mi ih u tom trenutku želimo imati kod sebe.

6. “Push”:

Push-ati znači sve pohranjene promjene(pomoću commit-a) spremiti na neki od internet servisa koji koristimo, kako bi bile prisutne u nekom od branch-eva.

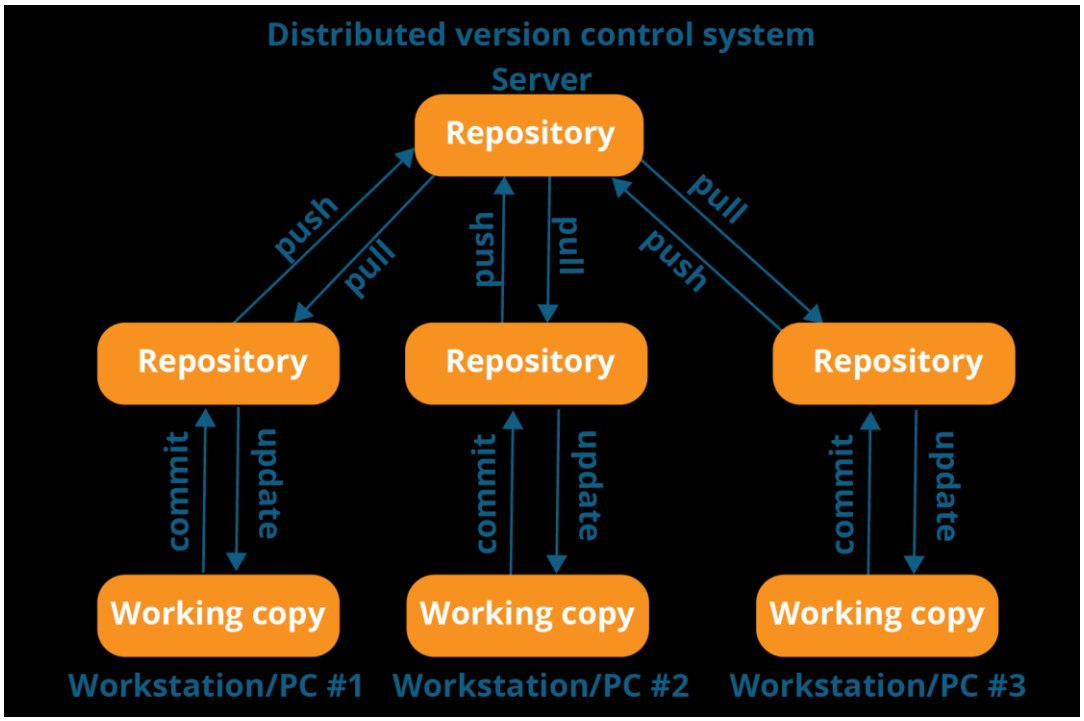
7. “Repository”:

Repozitorij je mjesto gdje se nalaze naši dokumenti, datoteke, programski kod ili projekt. Može biti podešen lokalno na našem računalu ili negdje na nekome od internetskih poslužitelja što je češći slučaj.

2.3.2 Način rada

Distribuirani sustavi za “version control” nemaju potrebu oslanjati na središnji poslužitelj za pohranu svih postojećih verzija datoteka projekta, iako bi to u teoriji mogli. Umjesto toga, svaki sudionik klonira kopiju spremišta i u svojoj memoriji posjeduje kompletnu povijest cijelog projekta. Ova kopija također sadrži i sve meta-podatke originala. Nakon kopiranja svaki sudionik ima potpunu slobodu rada na svojem branch-u(svojoj kopiji), gdje nesmetano i neovisno od drugih sudionika projekta može raditi svoje promjene i testiranja.

Također ovakav sustav nam rješava i neke druge probleme koji su bili vezani za centralizirane sustave za version control, zato što svaki sudionik ima u pozadini spremljene sve druge promjene koje mogu utjecati na njegov programski kod, a svaki od sudionika radi na svojem branch-u tako da ih može biti poprilično velik broj.



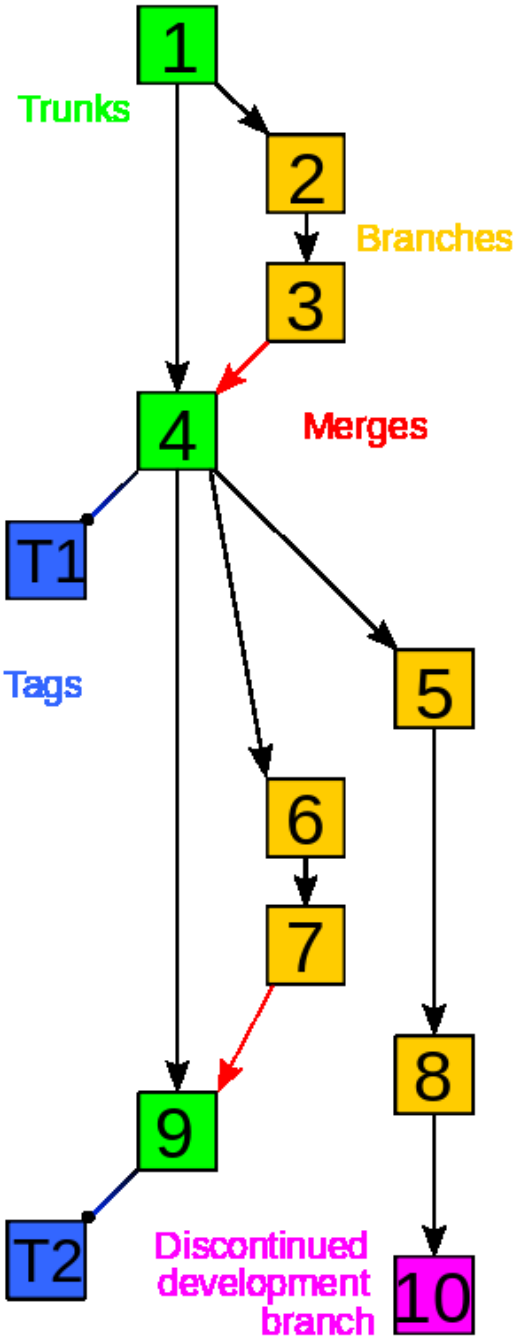
Slika 6: Distribuirani Version control sustav

Izvor: https://www.edureka.co/blog/what-is-git/

Ovakav način rada može se činiti komplicirano, ali u većini slučajeva u radu sa dobrom organizacijom i raspodjelom zadataka, rijetko se javljaju problemi. Velika većina programskih projekata i njegove dokumentacije, sastoji se od običnih tekstualnih datoteka, i određenog broja slika i multimedijalnih zapisa. U današnje vrijeme diskovni prostor je prilično jeftin, tako da pohrana velikog broja kopija datoteka, tako da dostupnost memorije stvara minimalne probleme. Moderniji sustavi isto tako koriste sustave za komprimiranje datoteka kako bi se što je više moguće uštedilo na prostoru.

2.3.3 Generalni tijek rada

Generalni tijek rada sa Distribuiranim version control sustavima prikazan je na slijedećem grafu:



Slika 7: Generalni primjer tijeka rada sa version control sustavima

Izvor:https://en.wikipedia.org/wiki/Merge_(version_control)#/media/File:Revision_controlled_project_visualization-2010-24-02.svg

2.3.4 Prednosti distribuiranih sustava za “version control”:

Svi sudionici imaju lokalnu kopiju, izmjene koje su napravljene mogu se koristiti na lokalnom stroju. Lokalna kopija je velika prednost jer se ne može narušiti stabilnost i funkcionalnosti master branch-a.

Inkrementalna povijest sadrži sve promjene koje su napravljene, nema potreba za čestim provjerama ili čestim spremanjima promjena.

Ovakav sustav može se u nekoj mjeri koristiti bez upotrebe interneta. Pristup internetu potreban je samo za preuzimanje i pohranjivanje promjena u repozitorij. Nakon navedenih operacija nije bitno gdje se nalazite i imate li pristup internetu.

Ovakav sustav je izgrađen za pohranjivanje promjena, a svaka promjena sadrži svoje opise što uvelike poboljšava preglednost svih promjena.

Kao posljedice merga-anja i kloniranja ne javljaju se redundancije podataka jer se distribuirani sustav brine o tome te korisnik nema briga vezanih za redundanciju.

Lako se upravlja repozitorijima i sustavi za upravljanje promjenama nisu zahtjevni u aspektu računalnih performansi.

2.3.5 Nedostaci distribuiranih sustava za “version control”

U nekim slučajevima znaju se javljati potrebe za sigurnosnim kopijama, iz razloga koje su navedene kao prednosti. U slučaju kada netko radi na dokumentu ili programskom kodu, a nije direktno spojen na internetsku mrežu mi ne možemo znati što se s tim sudionikom događa i dali on posjeduje najnovije promjene.

Iako postoji vrh master branch-a, u određenom trenutku nitko ne posjeduje konačnu verziju dokumenta ili programskog koda pogotovo u timovima sa više ljudi gdje svatko radi svoje promjene, što je čest slučaj.

Svake promjene bilježe se u obliku identifikacijskih oznaka koje nekada znaju biti zbunjujuće jer su najčešće zapisani u obliku nasumičnog skupa znamenki i slova.

I dalje postoji mogućnost konflikata, koji se može dogoditi u slučaju loše komunikacije ili loše raspodjele zadataka od strane nadređenih. Iako se rijetko događa da više ljudi radi na točno istome dijelu dokumenata ili programskog koda, ispravljanje konflikata može biti vrlo mukotrpno.

2.3.6 Version control sustavi

Neki od distribuiranih sustava za version control su:

1. Git – ujedno i najpopularniji sustav, koristi ga najveći broj korisnika.
2. Mercurial – manje popularan, koriste ga Facebook,Mozilla, itd.
3. Bazaar
4. Monotone
5. Fossil

Generalni primjer tijeka rada sa Mercurialom i Bazzar sustavom za version control:

“**Mercurial:**

- **Install:**sudo easy_install-2.5 mercurial
- **Make project directory:**mkdir hgrepo; cd hgrepo
- **Initialize project:**hg init
- **Add a file:** touch foo.txt; hg add foo.txt

- **Commit:**hg commit -m "added foo.txt" commit
- **Grab Shared Repository:**hg clone ssh://example.com//projects/hgrepo
- **Commit Changes Locally:**hg -ci -m "adding a change"
- **Push Changes to Server:**hg push
- **See Pending updates as patches:**hg incoming -p
- **Download updates from Server:**hg pull
- **Apply changes:**hg update
- **Merge conflicts:**hg merge
- **Merge two unrelated remote repositories:**hg pull -f ssh://example2.com//projects/hgrepo

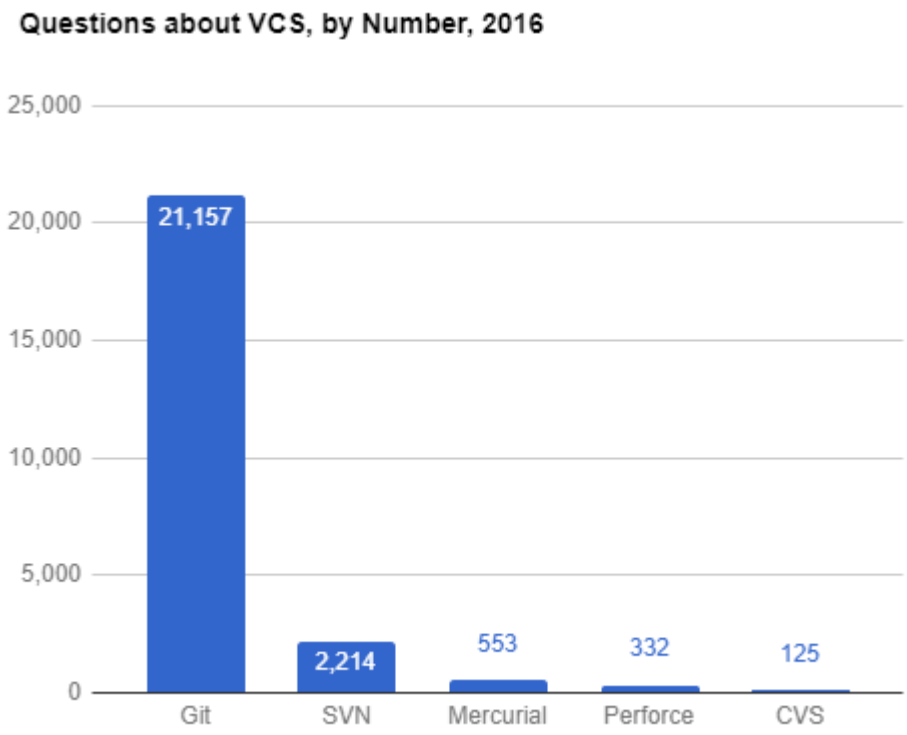
Bazaar:

- **Install**sudo easy_install-2.5 bzr
- **Make project directory:**mkdir bzrrepo; cd bzrrepo
- **Initialize project:** bzr init
- **Add a file:**touch foo.txt; bzr add foo.txt
- **Commit:** bzr commit -m "added foo.txt" commit
- **Grab Shared Repository:**bzr branch bzr+ssh://example.com/projects/gitrepo
- **Commit Changes Locally:**bzr -ci -m "adding a change"
- **Push Changes to Server:**bzr push
- **Download updates from Server:**bzr pull
- **Apply changes:**bzr update
- **Merge conflicts:**bzr merge“

Izvor:
https://web.archive.org/web/20090602084310/http://www.ibm.com/developerworks/aix/library/au-dist_ver_control/

Iz priloženog postupka generalnog tijeka rada sa sustavom Mercurial i Bazaar možemo vidjeti sličnosti naredbi koje se koriste u sustavima za version control.

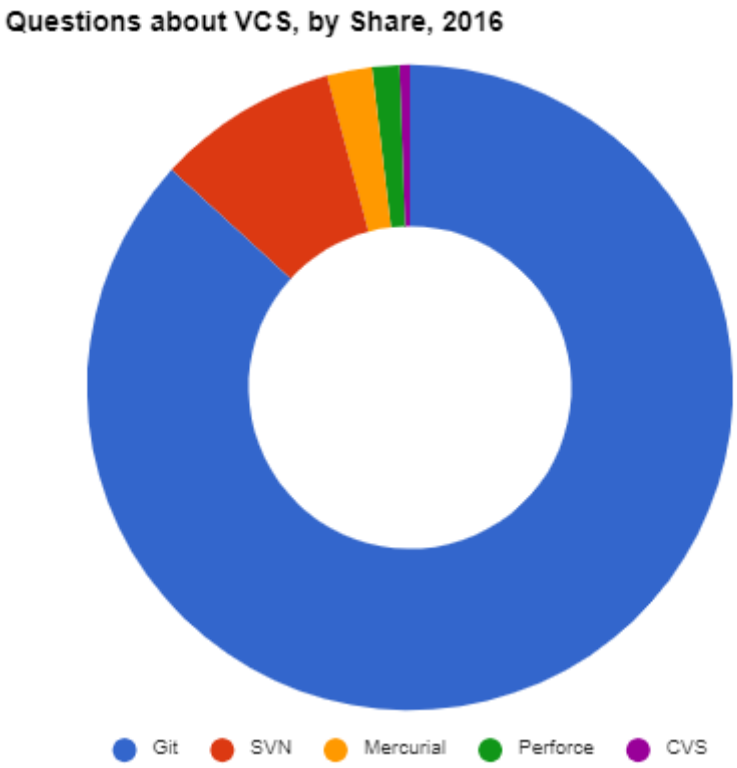
Od svih servisa daleko najpopularniji je Git. Prema podatcima sa web sjedišta Stackoverflow.com servis Git daleko prednjači sa najviše postavljenih pitanja vezanih za taj servis



.

Slika 8: Broj pitanja postavljenih na web središtu stackoverflow.com sortiran prema servisima

Izvor: https://rhodecode.com/insights/version-control-systems-2016



Slika 9: Broj pitanja postavljenih na web središtu stackoverflow.com sortiran prema servisima izražen u postotcima

Izvor: https://rhodecode.com/insights/version-control-systems-2016

U 2018. Godini web portal stackoverflow.com proveo je anketu u vezi korištenja sustava za version control. Rezultati su pokazali da od 74 000 programera njih gotovo 90% koristi Git kao izbor sustava za version control.

Servisi koji pružaju usluge poslužitelja za version control:

“Skladište izvornog koda je arhiva datoteka i web hosting uređaj za izvorni kod softvera, dokumentacije, web stranica i druga djela, dostupna javno ili privatno. Često se koriste u softverskim projektima otvorenog koda i drugim projektima sa više razvojnih programera za održavanje revizije i povijesti verzija ili kontrolu verzija. Mnoga spremišta nude sustav za praćenje grešaka i nude upravljanje verzija, popise adresa i projektnu dokumentaciju na bazi wikija. Autori softvera uglavnom zadržavaju svoja autorska prava kada se

softver objavi na uređajima za hosting koda.”- https://en.wikipedia.org/wiki/Comparison_of_source-code-hosting_facilities#General_information

U tablici su prikazani neki od popularnijih servisa koji pružaju usluge poslužitelja:

Name	CVS	Git	Hg	SVN	BZR	TFVC	Arch	Perforce	Fossil
Assembla	Ne	Da	Ne	Da	Ne	Ne	Ne	Da	Ne
Azure DevOps	Ne	Da	Ne	Ne	Ne	Da	Ne	Ne	Ne
Bitbucket	Ne	Da	do 2020	Ne	Ne	Ne	Ne	Ne	Ne
Buddy	Ne	Da	Ne	Ne	Ne	Ne	Ne	Ne	Ne
CloudForge	Ne	Da	Ne	Da	Ne	Ne	Ne	Ne	Ne
GForge	Da	Da	Ne	Da	Ne	Ne	Ne	Ne	Ne
Gitea	Ne	Da	Ne	Ne	Ne	Ne	Ne	Ne	Ne
GitHub	Ne	Da	Ne	DjelomičNe	Ne	Ne	Ne	Ne	Ne
GitLab	Ne	Da	Ne	Ne	Ne	Ne	Ne	Ne	Ne
GNU Savannah	Da	Da	Da	Da	Da	Ne	Da	Ne	Ne
java.net	Ne	Da	Da	Da	Ne	Ne	Ne	Ne	Ne
Kallithea	Ne	Da	Da	Ne	Ne	Ne	Ne	Ne	Ne
Launchpad	Import	Da	Import	Import	Da	Ne	Ne	Ne	Nepoznato
OSDN	Da	Da	Da	Da	Da	Ne	Ne	Nepoznato	Nepoznato
Ourproject.org	Da	Ne	Ne	Da	Ne	Ne	Ne	Nepoznato	Nepoznato
OW2 Consortium	Da	Ne	Ne	Da	Ne	Ne	Ne	Nepoznato	Nepoznato
Helix TeamHub	Ne	Da	Da	Da	Ne	Ne	Ne	Da	Ne
Phabricator	Ne	Da	Da	Da	Ne	Ne	Ne	Ne	Ne
RhodeCode	Ne	Da	Da	Da	Ne	Ne	Ne	Ne	Ne
SEUL.org	Da	Ne	Ne	Da	Ne	Ne	Ne	Nepoznato	Nepoznato
SourceForge	Prestao	Da	Da	Da	Prestao	Ne	Ne	Nepoznato	Da
Sourcehut	Ne	Da	Da	Ne	Ne	Ne	Ne	Nepoznato	Nepoznato

Tablica 1: Prikaz poslužitelja sa popisom servisa za verzioniranjem koje podržavaju.

Izvor:https://en.wikipedia.org/wiki/Comparison_of_source-code-hosting_facilities#General_information

Od svih poslužitelja koji podržavaju “version control” najviše korisnika imaju:

1. “Github – 31,000,000 korisnika, 100,000,000 projekata
2. Bitbucket – 5,000,000 korisnika
3. Launchpad – 3,965,288 korisnika, 40,881 projekata
4. SourceForge – 3,700,000 korisnika, 500,000 projekata
5. GitLab – 100,000 korisnika, 546,000 projekata”

Izvor:https://en.wikipedia.org/wiki/Comparison_of_source-code-hosting_facilities#cite_note-github.com-60

3 Git

3.1 Povijest Git-a

Git je kao sustav nastao kao rezultat frustracija zajednice koja je radila sa postojećim Linux Kernel sustavima za version control. Razvoj Linux Kernela bio je zbilja neobičan za vrijeme u kojem je nastajao. Postojao je jako velik broj sudionika među kojima su bili i brojni stručnjaci koji su pridavali svoje napore projektu. Kao rezultat takve situacije na koji zajednica u to vrijeme nije navikla, stvorila se potreba za konkretan sustav version controle koji bi im omogućavao nesmetan rad i lakšu organizaciju. Iako u tom trenutku svi postojeći sustavi kontrole verzija nisu u potpunosti odgovarali potrebama programera, uglavnom su bili prisiljeni koristiti kombinaciju postojećih sustava za kontrolu verzija koji su tada bili Concurrent Revisions System i BitKeeper. BitKeeper je služio kao distribuirani sustav kontrole verzija, dok je Concurret Revision System služio kao centralizirani sustav kontrole verzija u formi poslužitelj – klijent, te je omogućavao spremanje kroz takozvane “check-out” i “check-in” funkcije i na taj način omogućavao upotrebu radnih kopija programskog koda Linux Kernela. Međutim promjene su stigle 2005. Godine kada BitKeeper prestao biti open source program, te je vlasnik autorskih prava toga programa Larry McVoy najavio oduzimanje licence koja je omogućavala besplatno korištenje BitKeepera. Larry McVoy je optužio australijskog programera Andrew-a Tridgella koji je bio jedan od glavnih stvaralaca BitKeeper-a, da je postupkom obrnutog inženjerstva prekršio licencu BitKeeper-a, u svom razvoju programa SourcePuller. Nakon toga većina programera koji su se zalagali za razvoja Linux Kernel sustava pomoću BitKeeper-a, u tom trenutku više nisu mogli pridonositi projektu.

Linux zajednica se nadala kako će lako pronaći sustav koji će ponuditi alternativu BitKeeper-a, ali sličnog programa koji bi odgovarao potrebama programera u tom trenutku nije postojao. Linus Torvalds, glavni programer zadužen za projekt razvoja Linux Kernela, počeo je razmišljati o razvijanju novog sustava za kontrolu verzioniranjem, ubrzo nakon toga Linus Torvalds,

počeo je raditi na novom sustavu za kontrolu verzioniranjem iz kojega se naposlijetku razvio današnji Git.
U jednom od e-mailova Linus Torvalds se zahvalio i prenio BitKeeperu kako je sretan zbog njegovog pridonošenja razvoja Linux Kernel sustava, te kako je BitKeeper pomogao razvoju Linux Kernela pomoću njihovih funkcija koje su omogućile praćenje promjena te detaljniji prikaz povijesti spremljenih promjena. Nastanak Git-a nije bio jedini projekt potaknut povlačenjem licence BitKeeper-a, iz njega se izrodio i Git-ov konkurent Mercurial koji je drugi najpopularniji alat za version control.

Prve faze razvoja Git-a:

Linus Torvalds je nakon uskraćivanja besplatne licence BitKeeper-a uvidjeo je potrebu za konkretnim sustavom koji će preuzeti funkcije sustava za kontrolu verzija BitKeeper. Tada je istaknuo određene dizajnerske kriterije za novi sustav kontrole verzioniranja. Linus Torvalds taja je naglašavao tri ključne karakteristike novoga sustava:

1. Sustav zaštitnih mjera kojima će se paziti na korupcije datoteka i dokumenata.
2. Omogućavanje Distribuiranog toka razvijanja.
3. Visoke performanse

Također je naglašavao kako za neke od karakteristika kao što je naprimjer zakrpavanje podataka nebi trebao trajati više od tri sekunde, a da ostali sustavi za navođenje i upravljanje bi trebali svoj posao pokretanja zakrpa bi se trebao izvršavati unutar trideset sekundi, u koju se uključivala i operacija ažuriranja meta(opisnih) podataka. Tako zamišljeni sustav u tom trenutku nije mogao dostići svih 250 programera koji su u tom trenutku radili na razvoju Linux Kernel-a. Zbog ranoga utjecaja BitKeeper-a na razvoj Git-a, Git je tada omogućavao više lokalnih i distribuiranih tijekova nego što je to tada mogao BitKeeper. Sudionici projekta Linux Kernel-a su tada mogli raditi na svojim repozitorijima bez internetske mreže te inkrementalno raditi commit-eve, te su mogli birati koje promjene žele dijeliti, kada su njihove promjene spremne za

push, te su mogli pohranjivati njihov rad, njihove promjene na različite grane(branch-eve). Nakon kratkog vremena Linus Torvalds je ipak uspio razviti Git visokih performansi i on je napisan u potpunosti u programskom jeziku C, koji je i omogućio visoke performanse. Od 2005. Godine od kako je i inicijalno razvijen Git od strane Linusa Torvaldsa, Git kao takav se održavao uz konstantna poboljšanja. Za održavanje Git-a zadužen je Junio Hamano japanski programer koji trenutno radi za Google. Junio Hamano je dobio ovu priliku zbog Linusovog velikog povjerenja u njega, a tu priliku je dobio samo dva mjeseca nakon što je izašla prva verzija Git-a.

Današnja verzija Git-a je besplatna i distribuirana je pod “GNU General Public Licence” licencom. Linus Torvalds je svom sustavu dao ime Git(što u Britankom slengu predstavlja neugodnu osobu) te sarkastično izjavio: “Ja sam egoistični gad, a sve svoje projekte imenujem po sebi. Prvo 'Linux', sada 'git'.” A prvi opis Git-a je bio “glupi pratitelj sadržaja”. Iako je prvobitno osmišljen i napravljen samo za Linux, Git je danas dostupan na većini većih operacijskih sustava(Windows,MacOS i sl.)

Na službenoj dokumentaciji Junio Hamano je istaknuo kako se njegovo vrijeme na održavanju Git-a raspodjeljuje:

1. Komunikacija 45% vremena: rasprave, dijagnosticiranje i pregledavanje prijavljenih bugova, komentiranje, prijedlozi alternativa, i odbacivanje zakrpa
2. Integracija 50% vremena: prihvaćanje zakrpa od sudionika pri kojem se obavljaju manje ispravke i korekcije, testiranje bracheva, pushanje rezultata, te objavljivanja
3. Vlastiti development 5% vremena

Službena izdanja verzija Git-a

Version	Original release date	Latest (patch) version	Release date (of patch)
0.99	2005-07-11	0.99.9n	2005-12-15
1.0	2005-12-21	1.0.13	2006-01-27
1.1	2006-01-08	1.1.6	2006-01-30
1.2	2006-02-12	1.2.6	2006-04-08
1.3	2006-04-18	1.3.3	2006-05-16
1.4	2006-06-10	1.4.4.5	2008-07-16
1.5	2007-02-14	1.5.6.6	2008-12-17
1.6	2008-08-17	1.6.6.3	2010-12-15
1.7	2010-02-13	1.7.12.4	2012-10-17
1.8	2012-10-21	1.8.5.6	2014-12-17
1.9	2014-02-14	1.9.5	2014-12-17
2.0	2014-05-28	2.0.5	2014-12-17
2.1	2014-08-16	2.1.4	2014-12-17
2.2	2014-11-26	2.2.3	2015-09-04
2.3	2015-02-05	2.3.10	2015-09-29
2.4	2015-04-30	2.4.12	2017-05-05
2.5	2015-07-27	2.5.6	2017-05-05
2.6	2015-09-28	2.6.7	2017-05-05
2.7	2015-10-04	2.7.6	2017-07-30
2.8	2016-03-28	2.8.6	2017-07-30
2.9	2016-06-13	2.9.5	2017-07-30
2.10	2016-09-02	2.10.5	2017-09-22
2.11	2016-11-29	2.11.4	2017-09-22
2.12	2017-02-24	2.12.5	2017-09-22
2.13	2017-05-10	2.13.7	2018-05-22
2.14	2017-08-04	2.14.5	2018-09-27
2.15	2017-10-30	2.15.3	2018-09-27

2.16	2018-01-17	2.16.5	2018-09-27
2.17	2018-04-02	2.17.2	2018-09-27
2.18	2018-06-21	2.18.1	2018-09-27
2.19	2018-09-10	2.19.2	2018-11-21
2.20	2018-12-09	2.20.1	2018-12-15
2.21	2019-02-24	2.21.0	2019-02-24
2.22	2019-06-07	2.22.1	2019-08-10
2.23	2019-08-16	2.23.0	2019-08-16

Tablica 2: Verzije Git-a kroz njegovu povijest verzija. Dokaz da i sam version control sustav, koristi version control sustav

Izvor: https://en.wikipedia.org/wiki/Git#cite_note-linusGoogleTalk-11

3.2 **Git kao version control**

U današnje vrijeme Git se doista razlikuje od njegovog inicijalog izdanja. Kroz sve ove godine razvijanja sustav se poboljšavao, ali bitno je napomenuti da Git ima zbilju veliku bazu aktivnih korisnika koja je uvelike pridonjela njegovom razvitku.

Prednosti Git-a:

1. Brzina:

   Brzina je jedna od većih prednosti Git-a. Git je odlično optimiziran za procese koji provodi i uz to ne treba stalno biti spojen na internetsku mrežu kako bi izveo operacije. Osim toga kao što je već navedeno napisan je u programskom jeziku C, što znači da ima prednosti nad version control sustavima koji su napisani u programskim jezicima viših razina (eng. High level programming languages)
2. Pouzdanost

   Osim u rijetkim slučajevima rada bez pristupa internetskoj vezi i korupcijama podataka, sve promjene su spremljene kod svakoga od sudionika u njihovim lokalnim repozitorijima, što znači da se izgubljeni dokumenti lako mogu vratiti na zadnje stanje.

3. Podrška za nelinearno razvijanje

   Koristeći neograničeni broj grananja i ponovnih spajanja Git omogućuje nelinearni razvoj projekata. Grananje se koristi vrlo lako te daje nove lokalne kopije sa cijelom povijesti promjena koje su napravljene. Na taj način Master branch to jest glavna linija razvoja uvijek je najkvalitetnija, te nema rizika od narušavanja.
4. Sigurnost

   Korištenje algoritma za hashiranje SHA1, pri identifikaciji objekata u svojem spremištu. Sve datoteke su pomoću hash funkcije provjerene kada se izvrši check-out.
5. Skalabilnost

   Git je vrlo podložan povećanju sudionika zbog načina na koji funkcionira, iako broj repozitorija sa više korisnika raste Git koristi odličan sustav kompresije, što ga čini vrlo skalabilnim
6. Besplatan je:

   Git je izdan pod “GNU General Public License version 2” što znači da je potpuno besplatan.

3.2.1 Git naredbe

Neke naredbe u Git-u se koriste puno češće od drugih, to je jednostavno tako zbog njegovoga načina rada. U prvim primjerima naredbi Git-a navest ćemo one naredbe koje se češće koriste od drugih.

Generalno u Git Bash-u naredbe su složene u formatu:

“Git ‘naredba’ ‘opcija1’ ‘opcija2’...”

U nastavku su prikazane neke od git naredbi koje se češće koriste pri radu.

Za pregled svih naredbi i opcija koje se mogu koristiti postoji naredba “git help”.

Dobavljanje i kreiranje projekata:

Naredba	Opis
git init	Inicijalizacija lokalnog Git repozitorija
git clone	Kreiranje lokalne kopije

Tablica 3: Naredbe za inicijalizaciju i kloniranje

Izvor: https://github.com/joshnh/Git-Commands

Provjere i mijenja stanja:

Naredba	Opis
git status	Provjera stanja
git add [file-name.txt]	Dodavanje datoteke u “staging”
git add -A	Dodavanje svih novih i promjenjenih datoteka u “staging”
git commit -m "[commit message]"	Commit-anje promjena
git rm -r [file-name.txt]	Brisanje datoteke ili mape

Tablica 4: Naredbe za izmjene i promjene stanja

Izvor: https://github.com/joshnh/Git-Commands

Grananje i spajanje (eng. Branching and Merging):

Naredba	**Opis**
git branch	Izlistavanje grana
git branch -a	Izlistavanje grana (lokalnih i udaljenih)
git branch [branch name]	Kreiranje nove grane
git branch -d [branch name]	Brisanje grane
git push origin --delete [branch name]	Brisanje udaljene grane
git checkout -b [branch name]	Kreiranje grane i prebacivanje rada na nju
git checkout -b [branch name] origin/[branch name]	Kreiranje udaljene grane i prebacivanje rada na nju
git branch -m [old branch name] [new branch name]	Preimenovanje lokalne grane
git checkout [branch name]	Prebacivanje na granu
git checkout -	Prebacivanje na zadnju granu
git checkout -- [file-name.txt]	Odbacivanje promjena na datoteci
git merge [branch name]	Spajanje grane na aktivnu
git merge [source branch] [target branch]	Spajanje grane na ciljanu granu

Tablica 5: Naredbe za grananje i spajanje

Izvor: https://github.com/joshnh/Git-Commands

Dijeljenje i ažuriranje projekata:

Naredba	**Opis**
git push origin [branch name]	Postavljanje grane na udaljeni repozitorij
git push -u origin [branch name]	Postavljanje grane na udaljeni repozitorij i zapamti granu
git push	Postavljanje promjena na udaljeni repozitorij(na zapamćenu granu)
git push origin --delete [branch name]	Brisanje udaljene grane
git pull	Ažuriranje lokalnog repozitorija na zadnje promjene
git pull origin [branch name]	Ažuriranje promjena iz udaljenog repozitorija
git remote add origin	Dodavanje udaljenog repozitorija

Tablica 6: Naredbe za Dijeljenje i Ažuriranje

Izvor: https://github.com/joshnh/Git-Commands

Usporedbe i istraživanje

Naredba	**Opis**
git log	Pregled promjena
git log –summary	Pregled promjena sa detaljima
git diff [source branch] [target branch]	Pregled promjena između dvije grane

Tablica 7: Naredbe za usporedbe

Izvor: https://github.com/joshnh/Git-Commands

3.2.2 Nejasnoće između sličnih Git naredbi

Git Pull i Git Fetch:

Git Pull i Git Fetch naredbe su u naravi jako slične, ali ipak među njima postoji ključna razlika. Git Pull se koristi kako bi se „povukle“ promjene sa same glave master brancha, što znači da preuzete promjene se odmah integriraju u radnu kopiju dok Git Fetch naredba samo preuzima nove promjene sa udaljenog repozitorija, ali ne integrira nove promjene u radnu kopiju.

Git Push i Git Commit:

Git commit koristimo kada želimo dodati promjene u svoj lokalni repozitorij dok se naredba Git push koristi za prebacivanje svih commitanih promjena na udaljeni repozitorij.

Git Merge i Git Rebase:

Kada se koristi Git merge glava master branch-a će se commit-ati te na taj način će sačuvati obije povijesti commit-ova, dok je rezultat naredbe Git rebase brisanje commit povijesti na način da se ona zapisuje direktno na master branch.

4. Zaključak

Možda se u početku version control sustavi ne čine kao nešto toliko značajno pogotovo na linearnim primjerima, ali možemo zaključiti da je njihov utjecaj na informatičku industriju, a osobito na računalne programere već je sada ostavio velike tragove. Većina aplikacija i sustava koji imaju određeni broj funkcionalnosti i modula koristi neki od version control sustava. Iako se na prvu koncept version control sustava se možda ne čini toliko kompleksan u praksi se za početnike pokazalo drugačije, ali jedno je sigurno: version control sustave svakako se isplati poznavati i koristiti. Zbog svojih karakteristika i prednosti version control sustavi pružili su informatičkoj industriji zbilja puno. U današnje vrijeme zahvaljujući njima moguće je da tisuću stručnjaka sa različitih dijelova svijeta istovremeno rade na jednom projektu na nekoliko razina. Timovi za razvijanje programskog koda, timovi koji osiguravaju kvalitetu provjeravaju napravljene funkcionalnosti rade na istom projektu, a u isto vrijeme poduzeće na tom istom sustavu u produkcijskom okruženju izvršava svakodnevne zadatke, i to sve u jednom velikom transparentnom sustavu. Uz ovakve mogućnosti nije čudno da su version control sustavi toliko zaživjeli. Velik broj sustava nastao je kao rezultat korištenja nekih od version control sustava, a razvijanje velikih sustava uveliko je olakšano i ubrzano ovakvim načinom rada. Iako imaju svojih nedostataka, version control sustavi zbilja su umanjili mogućnost pogreški na minimum, pogotovo uz dobru organizaciju poslova i dobroj komunikaciji između samih sudionika takvih projekata. U svijetu version control sustava daleko vodeći je sustav Git, iako postoji broj velikih sustava razvijen uz pomoć version control sustava Mercurial, u čije ime govore Mozilla i Facebook. Iako se Git koristi u većini velikih projekata širom svijeta, itekako se preporuča početnicima da unaprjeđuju svoje znanje vezano za version control sustave, pogotovo ako se bave ili planiraju baviti nekima od djelatnosti vezane uz razvoj softvera. Otkako je Git stekao popularnost, on ne prestaje rasti. Više je razloga zašto je to tako. Git kao version control sustav ima najveću zajednicu, koja aktivno koristi i aktivno radi na konstantnom poboljšanju Git-a kao version control sustava, teško da će ga u skorije vrijeme neki drugi sustav zamijeniti. Git je zasigurno jedan od najmoćnijih alata u svojoj branši, stoga ne čudi činjenica da je početnicima potrebno neko vrijeme dok se ne priviknu na rad sa Git version control sustavom. Zahvaljujući vizijama Linusa Torvaldsa i zahvaljujući

povlačenju besplatne licence BitKeeper version control sustava, Git je postao to što je danas, te se u manjoj mjeri smatra standardom za version control sustave. Uz veliku zajednicu aktivnih korisnika Git ima najveći broj pružatelja usluga hostinga, ali velika zajednica je ujedno i razlog zašto najveći broj sustava podržava Git.

LITERATURA

Knjige:

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

Scott Chacon, Ben Straub, Pro Git 2nd Edition, Apress, 2014

Internet:

https://mastersofmedia.hum.uva.nl/blog/2009/11/01/git-virtue-github-and-commons-based-peer-production/

https://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/

https://www.atlassian.com/blog/software-teams/version-control-centralized-dvcs

https://atlassian.com/agile/software-development/git

https://www.edureka.co/blog/git-commands-with-example/

https://about.gitlab.com/2017/05/17/learning-curve-is-the-biggest-challenge-developers-face-with-git/

https://www.techopedia.com/definition/30953/distributed-version-control-system-dvcs

https://en.wikipedia.org/wiki/Comparison_of_source-code-hosting_facilities#General_information

https://en.wikipedia.org/wiki/Git#cite_note-linusGoogleTalk-11

https://github.com/joshnh/Git-Commands

https://paulhammant.com/2014/09/29/shades-of-trunk-based-development/

https://blog.codinghorror.com/software-branching-and-parallel-universes/

https://www.edureka.co/blog/what-is-git/

https://rhodecode.com/insights/version-control-systems-2016

POPIS TABLICA

POPIS ILUSTRACIJA