Bosna i Hercegovina

Federacija Bosne i Hercegovine

Tuzlanski kanton

[Mješovita Srednja Elektro-mašinska Škola Lukavac](https://msemsl.wordpress.com/)

Maturski rad iz web programiranja

Tema: GITHUB

Mentor:

Emina Arapčić, prof

Ucenik:

Belma Mahmutović

Lukavac, maj 2019

Članovi komisije:

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Komentar:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ocjena:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sadržaj

[1. Uvod 4](#_Toc8858266)

[2. Šta je verzioniranje koda? 5](#_Toc8858267)

[2.1. Linearno verzioniranje koda 5](#_Toc8858268)

[2.2. Grafovi, grananje i spajanje grana 6](#_Toc8858269)

[3. Instalacija, konfiguracija i prvi projekat 9](#_Toc8858270)

[3.1. Instalacija 9](#_Toc8858271)

[3.2. Prvi git repozitorij 9](#_Toc8858272)

[3.3. Git naredbe 9](#_Toc8858273)

[3.4. Osnovna konfiguracija 10](#_Toc8858274)

[3.5. .gitignore 10](#_Toc8858275)

[4. Spremanje izmjena 11](#_Toc8858276)

[4.1. Indeks 11](#_Toc8858277)

[4.1.1. Spremanje u indeks 12](#_Toc8858278)

[4.1.2. Micanje iz indeksa 12](#_Toc8858279)

[4.2. Prvi commit 13](#_Toc8858280)

[4.3. Datoteke koje ne želimo u repozitoriju 13](#_Toc8858281)

[4.4. Historija projekta 14](#_Toc8858282)

[5. Grananje 15](#_Toc8858283)

[5.1. Popis grana projekta 15](#_Toc8858284)

[5.2. Nova grana 16](#_Toc8858285)

[*5.3.* Prebacivanje s grane na granu 16](#_Toc8858286)

[5.4. Brisanje grane 16](#_Toc8858287)

# Uvod

Git je alat koji je razvio Linus Torvalds da bi mu olaksao vodjenje jednog velikog i kompleksnog projekta – Linux kernela. U pocetku to nije bio program s danasnjom namjenom; Linus je zamislio da git bude osnova drugim sistemima za razvijanje koda. Drugi alati su se trebali razvijati na osnovu gita. Tako je, barem, bilo zamisljeno. Medutim, kao s mnogim drugim projektima otvorenog koda, ljudi su ga poceli koristiti takvog kakav jest, a on je rastao sa zahtjevima korisnika. Rezultat je program koji ima drugaciju terminologiju u odnosu na druge slicne, ali milioni programera sirom svijeta su ga prihvatili. Nastale su brojne platforme za hosting projekata, kao sto je Github1 , a vec postojeci su morali dodati git jednostavno zato sto su to njihovi korisnici trazili.

Nekoliko je razloga zasto je to tako:

* Postojeci sistemi za verzioniranje su zahtijevali da se tacno zna ko sudjeluje u projektu . To je demotivisali ljude koji bi mozda pokusali pomoci projektima kad bi imali priliku. S distribuiranim sistemima bilo ko moze raditi na njemu. Ukoliko misli da je napravio nesto korisno – vlasniku originalnog repozitorija bi predlozio da preuzme njegove izmjene. Broj ljudi koji se mogu ucestvovati u radu na nekom projektu je puno veci, a vlasnik i dalje zadrzava pravo odlucivanja cije ce izmjene uzeti, a cije nece.
* git je brz,
* vrlo je lako i brzo granati, isprobavati izmjene koje su radili drugi ljudi i preuzetiih u svoj kod,
* Linux kernel se razvijao na gitu, tako da je u svijetu otvorenog koda (open source) git stekao vaznost.

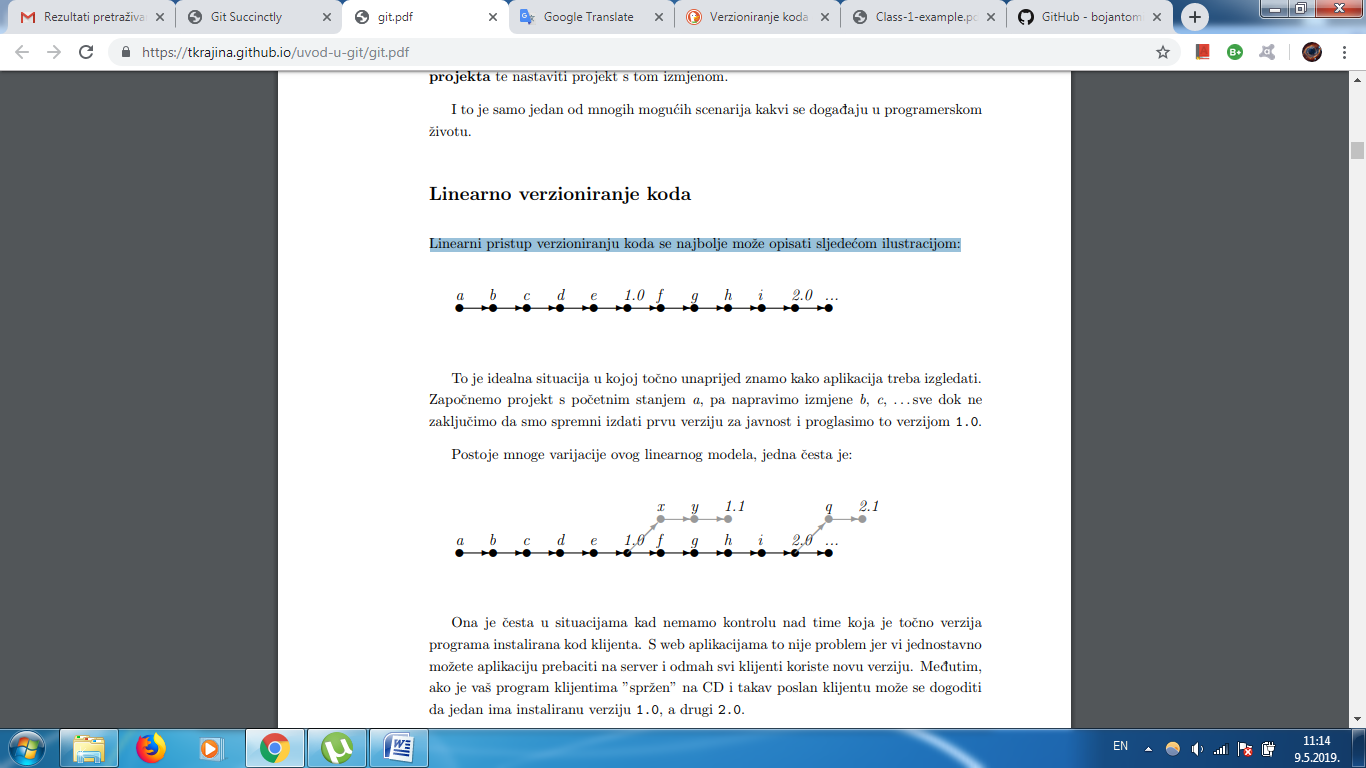
# Šta je verzioniranje koda?

S problemom verzioniranja koda sreli ste se kad ste prvi put napisali program koji rješava neki konkretan problem.

Svaka aplikacija koja ima stvarnog korisnika kojemu rješava neki stvarni problem ima i korisničke zahtjeve. Taj korisnik možemo biti mi sami, može biti neko hipotetsko tržištete (kojem planiramo prodati rješenje) ili može biti naručitelj. Korisničke zahtjeve ne možemo nikad tačno predvidjeti u trenutku kad krenemo pisati program. Možemo satima, danima i mjesecima sjediti s budućim korisnicima i planirati šta će sve naša aplikacija imati, ali kad korisnik sjedne pred prvu verziju aplikacije, čak i ako je pisana tačno prema njegovim specifikacijama, on će naći našto što valja. Radi li se o nekoj maloj izmjeni, možda ćemo je napraviti na licu mjesta. Možda ćemo trebati otići kući, potrošiti nekoliko dana i napraviti novu verziju.

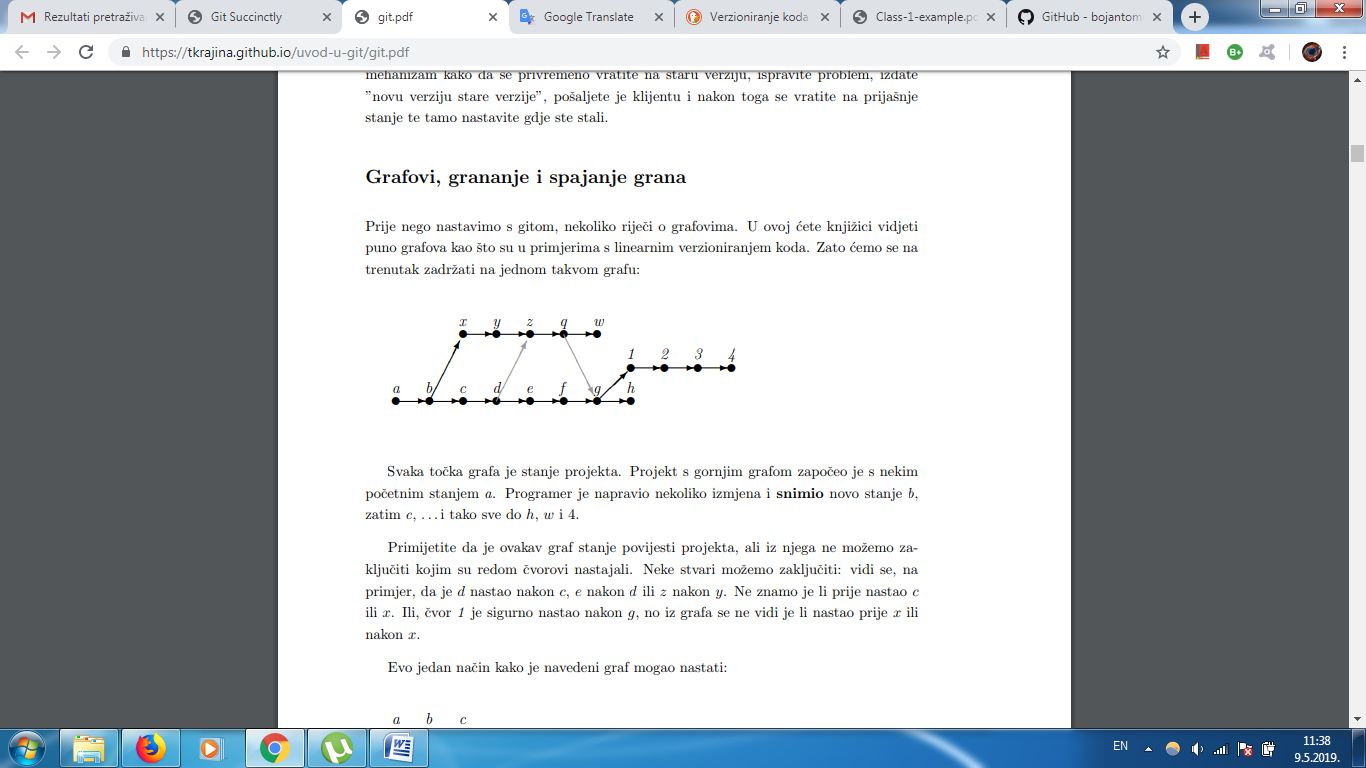
## Linearno verzioniranje koda

Linearni pristup verzioniranju koda se može se opisati sljedećom ilustracijom:



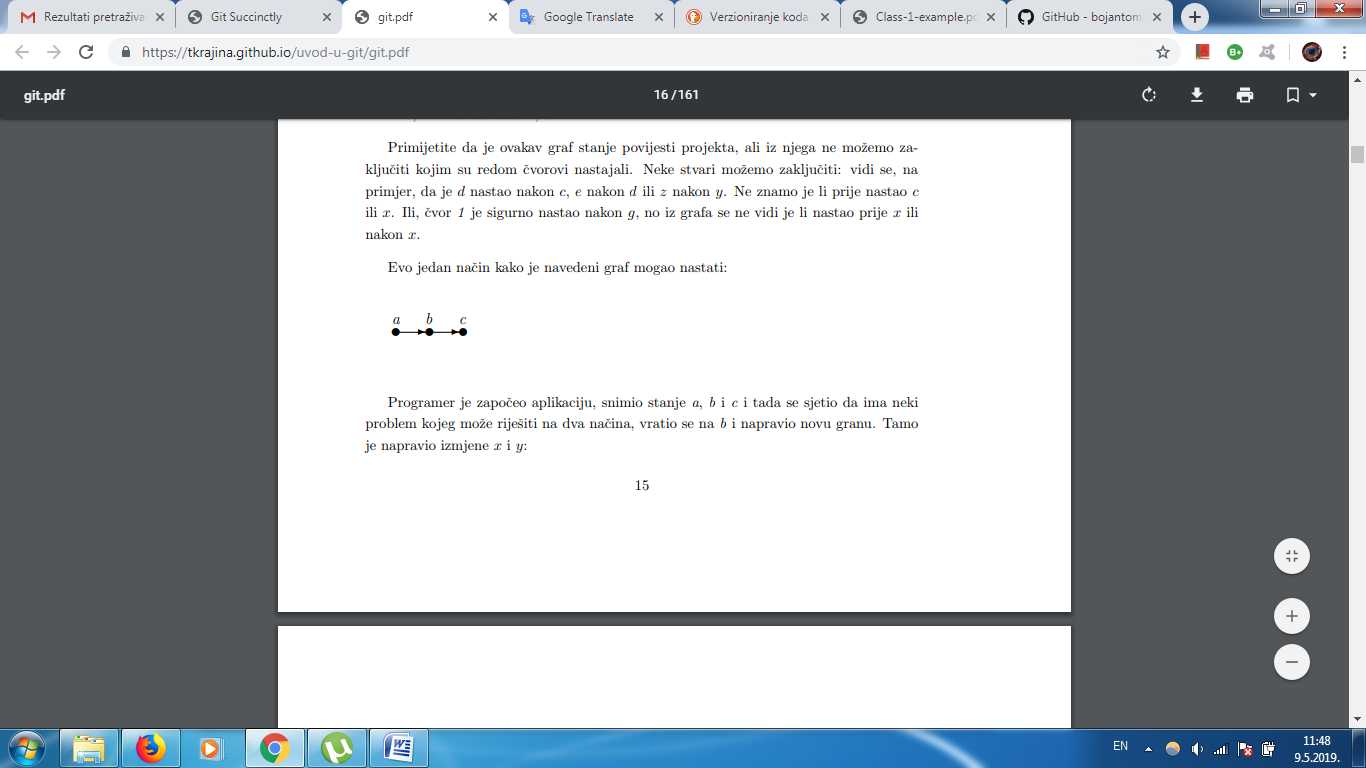
To je idealna situacija u kojoj tačno unaprijed znamo kako aplikacija treba izgledati. Započnemo projekat s početnim stanjem a, pa napravimo izmjene b, c, . . . sve dok ne zaključimo da smo spremni iynijeti svoju prvu verziju za javnost i proglasimo to verzijom 1.0.

## Grafovi, grananje i spajanje grana

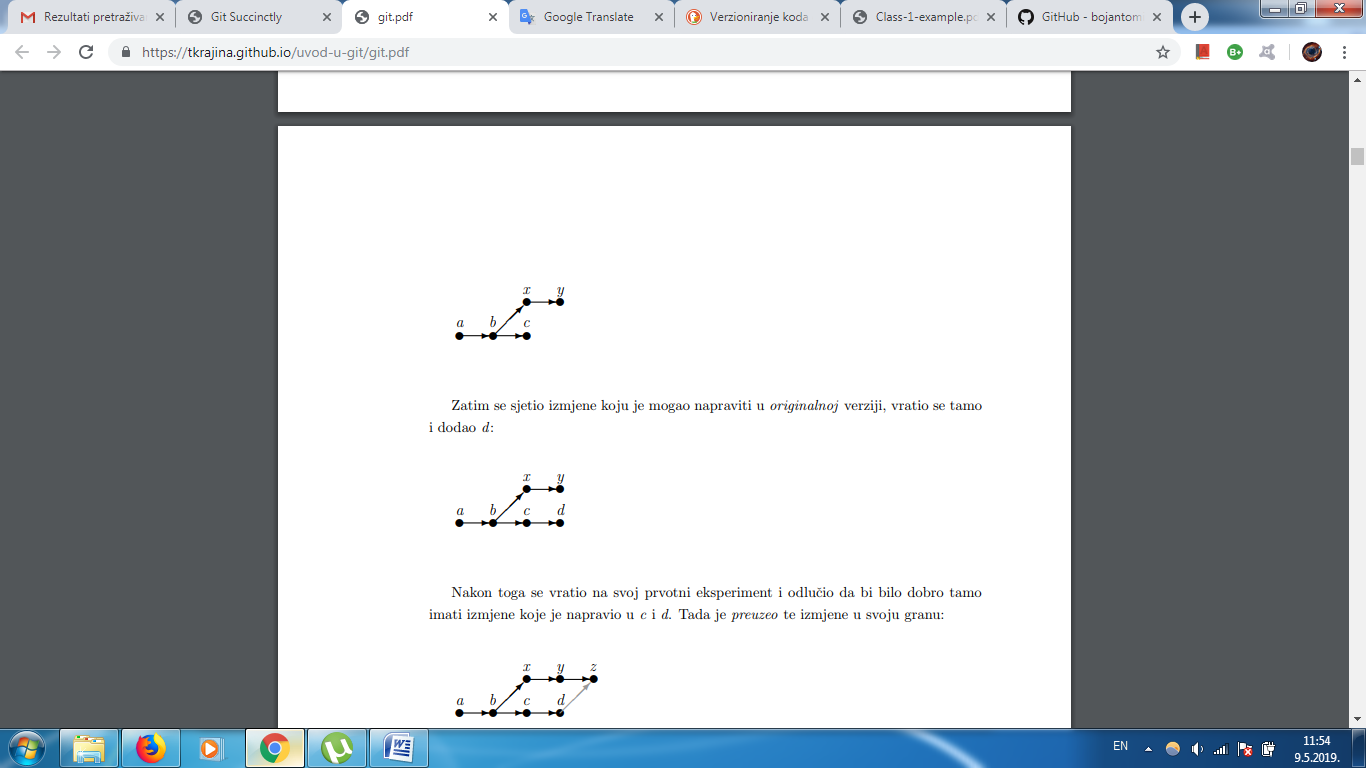


Svaka tačka grafa je stanje projekta. Projekat sa gornjim grafom započeo je sa početnim stanjem *a*. Programer je napravio nekoliko izmjena i snimio novo stanje *b*, zatim *c*, . . . i tako sve do *h, w* i *4*. Ovakav grafprestavlja stanje historije projekta, ali iz njega ne možemo zaključiti kojim su redom čvorovi nastajali. Neke stvari možemo zaključiti: vidi se da je d nastao nakon *c*, *e* nakon *d* ili *z* nakon *y*. Ne znamo je li prije nastao *c* ili *x*. Čvor 1 je sigurno nastao nakon *g*, ali iz grafa se ne vidi je li nastao prije x ili nakon x.

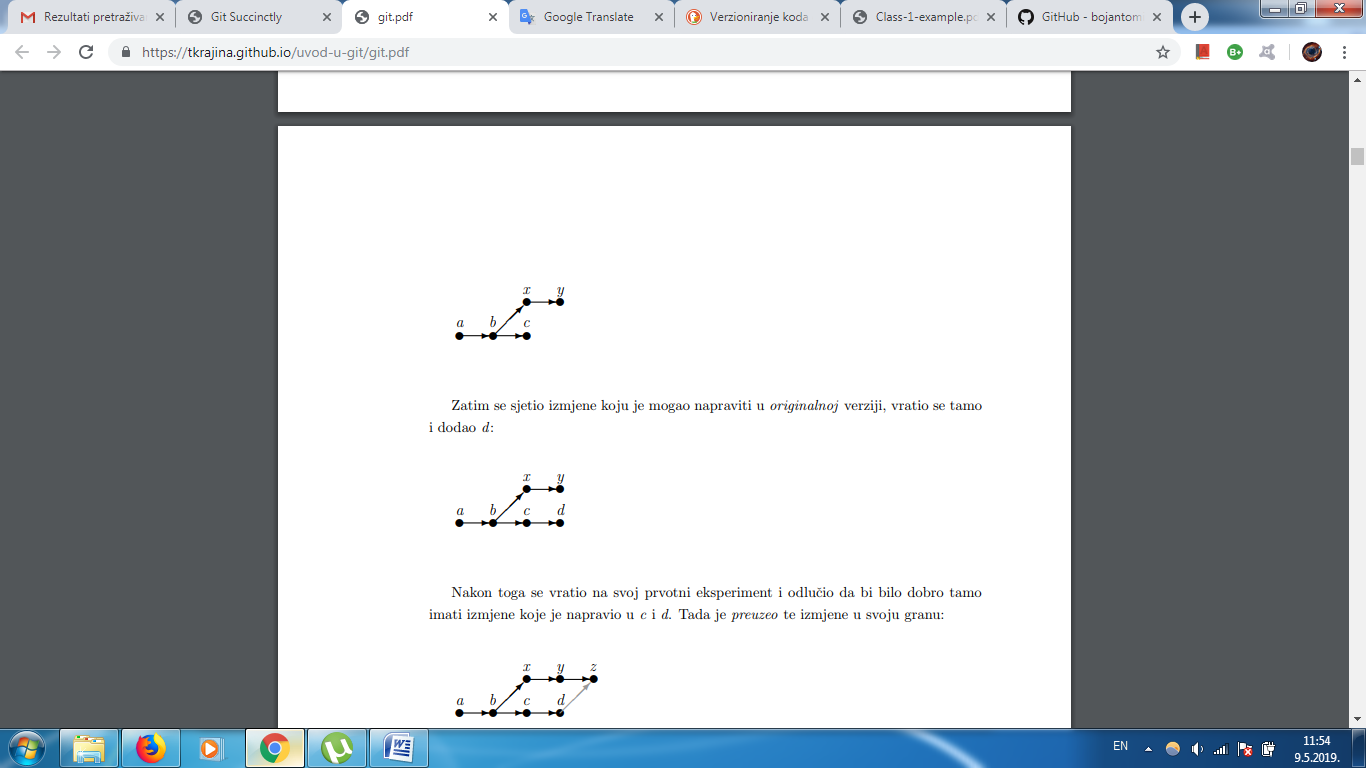
Način kako je navedeni graf mogao nastatii:



Programer je započeo aplikaciju, snimio stanje *a, b* i *c* i tada se sjetio da ima neki problem koji može riješiti na dva načina, vratio se na b i napravio novu granu. Tamo je napravio izmjene *x* i *y*:

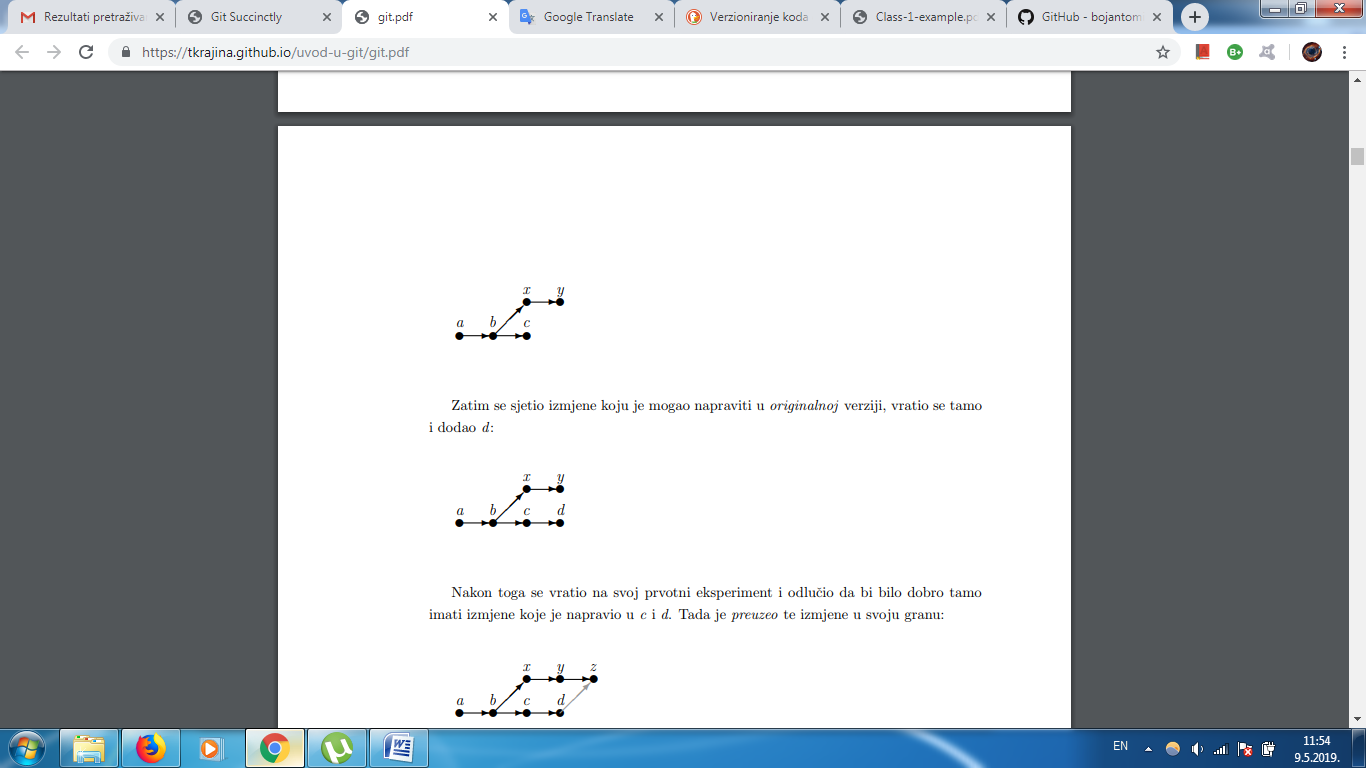


Sjetio se izmjene koju je mogao napraviti u originalnoj verziji, vratio se tamo i dodao *d*:

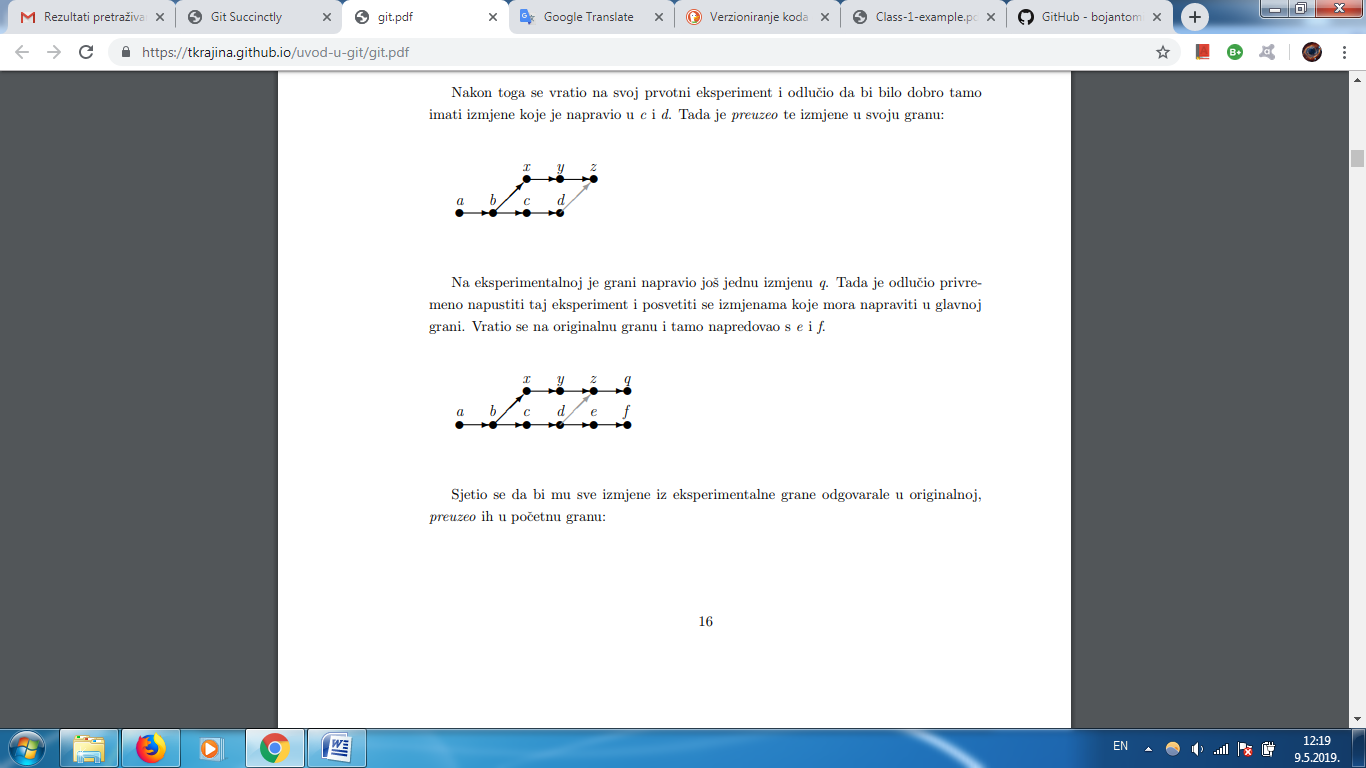


Vratio se na prvobitnu verziju i odlučio da bi bilo dobro tamo napraviti izmjene koje je napravio u *c* i *d*.

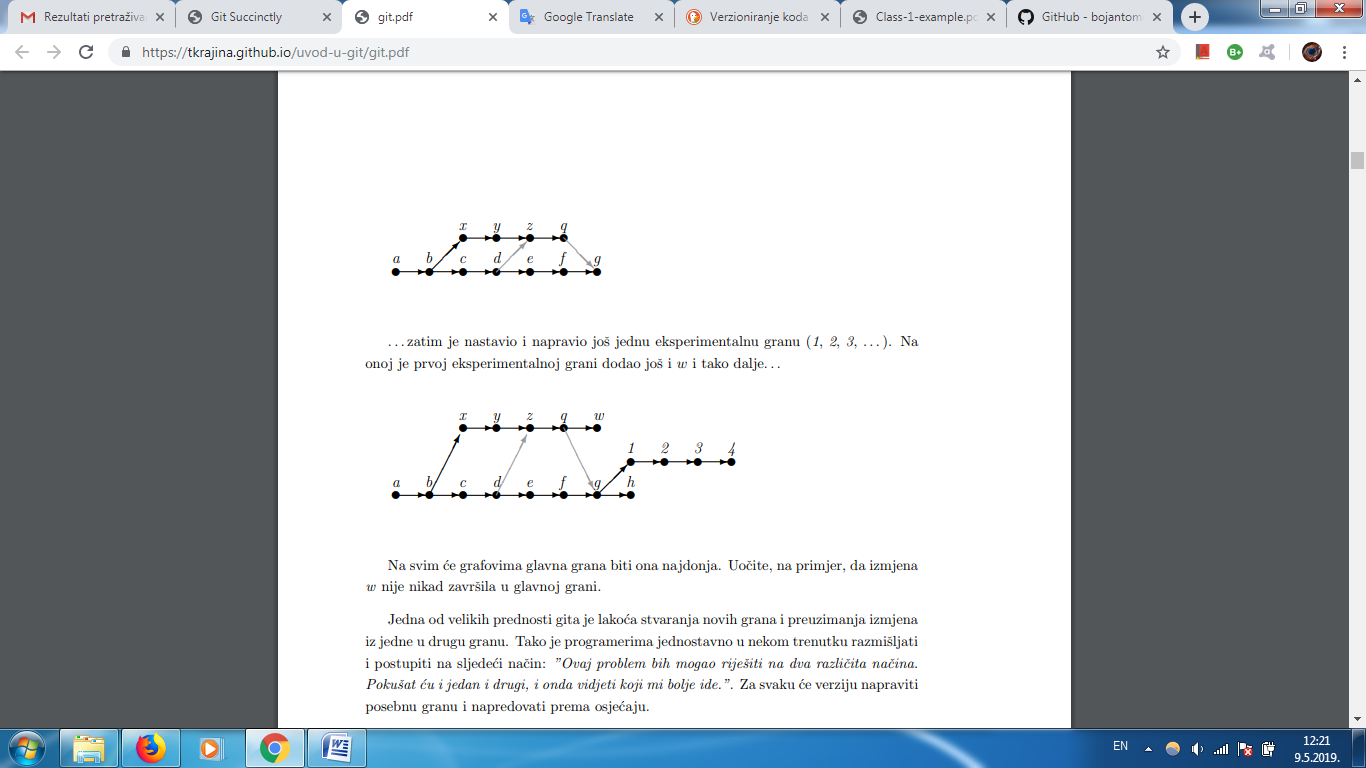
Tada je preuzeo te izmjene i dodao u svoju granu:



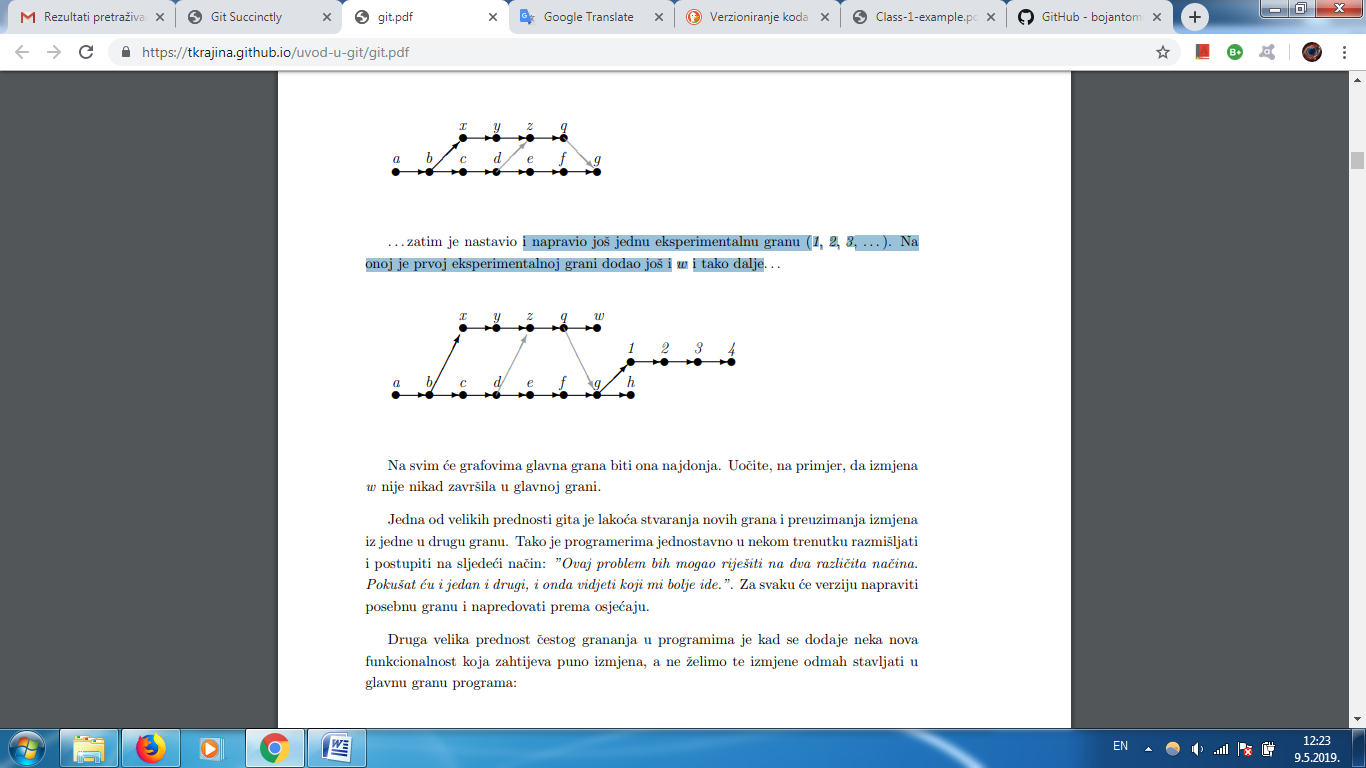
Na eksperimentalnoj grani napravljena je još jedna izmjena *q*. Tada se vraća izmjenama koje koje se moraju napraviti u glavnoj grani. Na originalnoj grani dodadte su izmjene *e* i *f*.



Sve izmjene iz eksperimentalne grane odgovarale su u originalnoj, pa su preuzete u glavnoj grani:



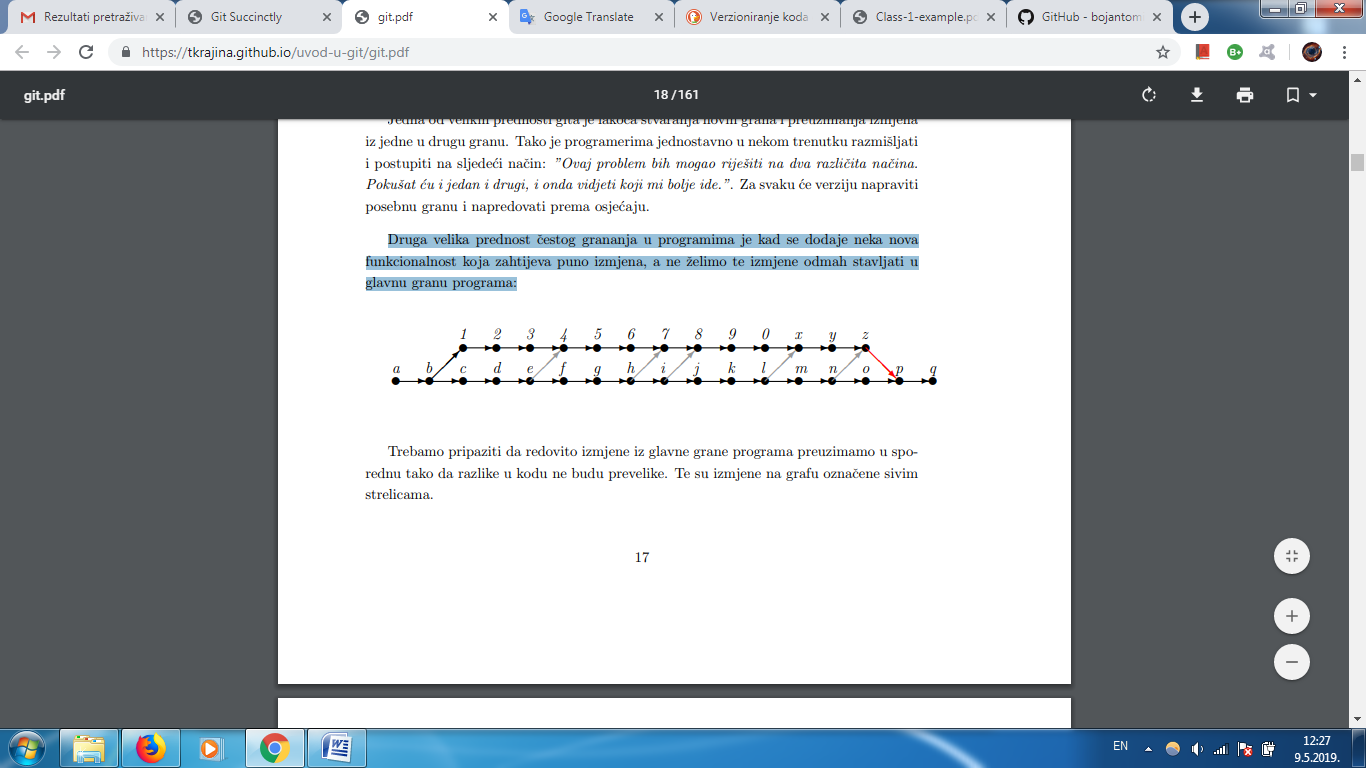
Napravljena je još jedna ekspreimentalna grana (1, 2, 3, . . . ). Na prvoj eksperimentalnoj grani dodadno je *w* .



Glavna grana u grafovima će uvijek biti ona koja je najdonja.

Jedna od velikih prednosti gita je lakoća stvaranja novih grana i preuzimanja izmjena iz jedne u drugu.

Druga velika prednost čestog grananja u programima je kad se dodaje neka nova funkcionalnost koja zahtijeva puno izmjena, a ne želimo te izmjene odmah stavljati u glavnu granu programa:



Trebamo pripaziti da redovno izmjene iz glavne grane programa preuzimamo u sporednu tako da razlike u kodu ne budu velike.

Kad završimo novu funkcionalnost, u glavnu granu treba preuzeti sve izmjene iz sporedne. Na taj ćemo način često imati ne samo dvije grane (glavnu i sporednu) nego nekoliko njih. Imati ćemo posebne grane za različite nove funkcionalnosti, posebne grane za eksperimente, posebne grane u kojima ćemo isprobavati izmjene koje su napravili drugi programeri, posebne grane za ispravljanje pojedinih *bugova*,...

# Instalacija, konfiguracija i prvi projekat

## Instalacija

## Prvi git repozitorij

Apsolutno svaki direktorij može postati git repozitorij. Ne mora postojati udaljeni server i neki centralni repozitorij kojeg koriste i ostali koji rade na projektu. Ako već postoji udaljeni repozitorij s kojeg preuzimate izmjene od drugih programera on ne mora biti jedan jedini. Može postojati više takvih udaljenih repozitorija, sami ćete odlučiti na koje ćete ”slati” svoje izmjene i sa kojih preuzimati izmjene. I vlasnici tih udaljenih repozitorija imaju istu slobodu kao i vi, mogu sami odlučiti čije će izmjene preuzimati kod sebe i kome slati svoje izmjene. Stvari, uglavnom, funkcionišu bez većih problema.

Prvi i najjednostavniji korak: napravit ćemo novi direktorij moj-prvi-projekat i stvoriti novi repozitorij u njemu:

$ mkdir moj-prvi-projekt

$ cd moj-prvi-projekt

$ git init

Initialized empty Git repository in /home/user/moj-prvi-projekt/.git/

$

Sa *git init* stvoren je direktorij *.git*. U principu cijela historija, sve grane, čvorov, i komentari, apsolutno sve vezano uz repozitorij čuva se u tom direktoriju.

## Git naredbe

U prethodnom primjeru smo u našem direktoriju inicijalizirali git repozitorij naredbom *git init*. Git naredbe uvijek imaju sljedeći oblik:

git <naredba> <opcija1> <opcija2> ...

## Osnovna konfiguracija

Sva git konfiguracija se postavlja pomoću naredbe *git config.* Postavke mogu biti lokalne (odnosno vezane uz jedan jedini projekt) ili globalne (vezane uz korisnika na računaru).

Globalne postavke se postavljaju s:

git config --global <naziv> <vrijednost>

. . . i one se spremaju u datoteku *.gitconfig* u vašem *home* direktoriju. Lokalne postavke se spremaju u *.git* direktorij u direktoriju koji sadrži vaš repozitorij, a tada je format naredbe *git config*:

git config <naziv> <vrijednost>

Za normalan rad na nekom projektu, drugi korisnici trebaju znati ko je tačno radio koje izmjene na kodu (commitove). Zato trebamo postaviti ime i email adresu koja će u historiji projekta biti ”zapamćena” uz svaku našu spremljenu izmjenu:

$ git config --global user.name "Belma Mahmutovic"  
$ git config --global user.email [belma.m@privatna.domena.com](mailto:belma.m@privatna.domena.com)

## .gitignore

Prije ili kasnije desit će se situacija da u direktoriju sa repozitorijem imamo datoteke koje ne želimo spremati u historiju projekta. To mogu biti konfiguracijske datoteke za različite editore ili datoteke koje nastaju kompajliranjem (.*class* za javu, .*pyc* za python, .*o* za C, i sl.). U tom slučaju trebamo nekako gitu dati do znanja da takve datoteke ne treba nikad snimati. Otvorite novu datoteku naziva .*gitignore* u glavnom direktoriju projekta (ne nekom od poddirektorija) i jednostavno unesite sve ono što ne treba biti dio historije projekta.

Ako ne želimo .*class*, .*swo*, .*swp* datoteke i sve ono što se nalazi u direktoriju *target*/ , naša će .*gitignore* datoteka izgledati ovako:

# Vim privremene datoteke:

\*.swp

\*.swo

# Java kompajlirane klase:

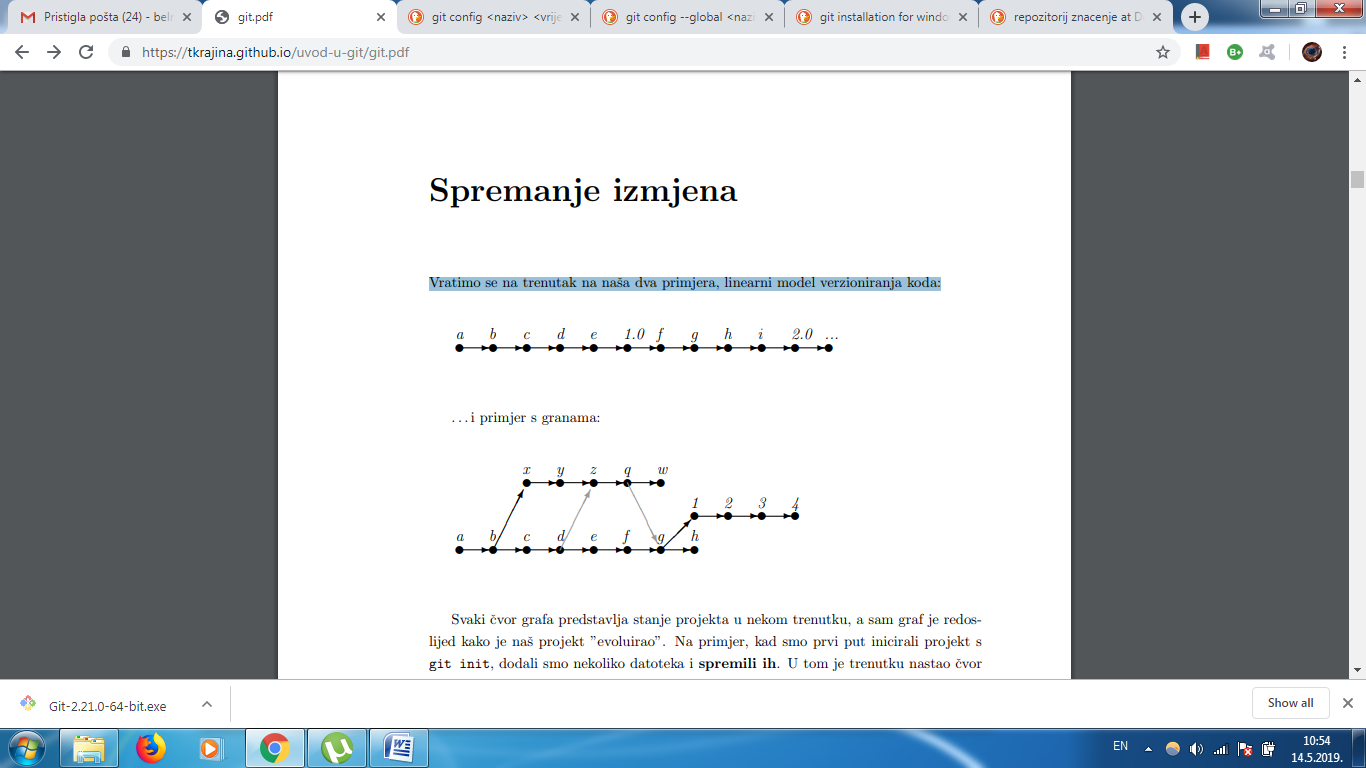
\*.class # Output direktorij s rezultatima kompajliranja i builda:

target/\*

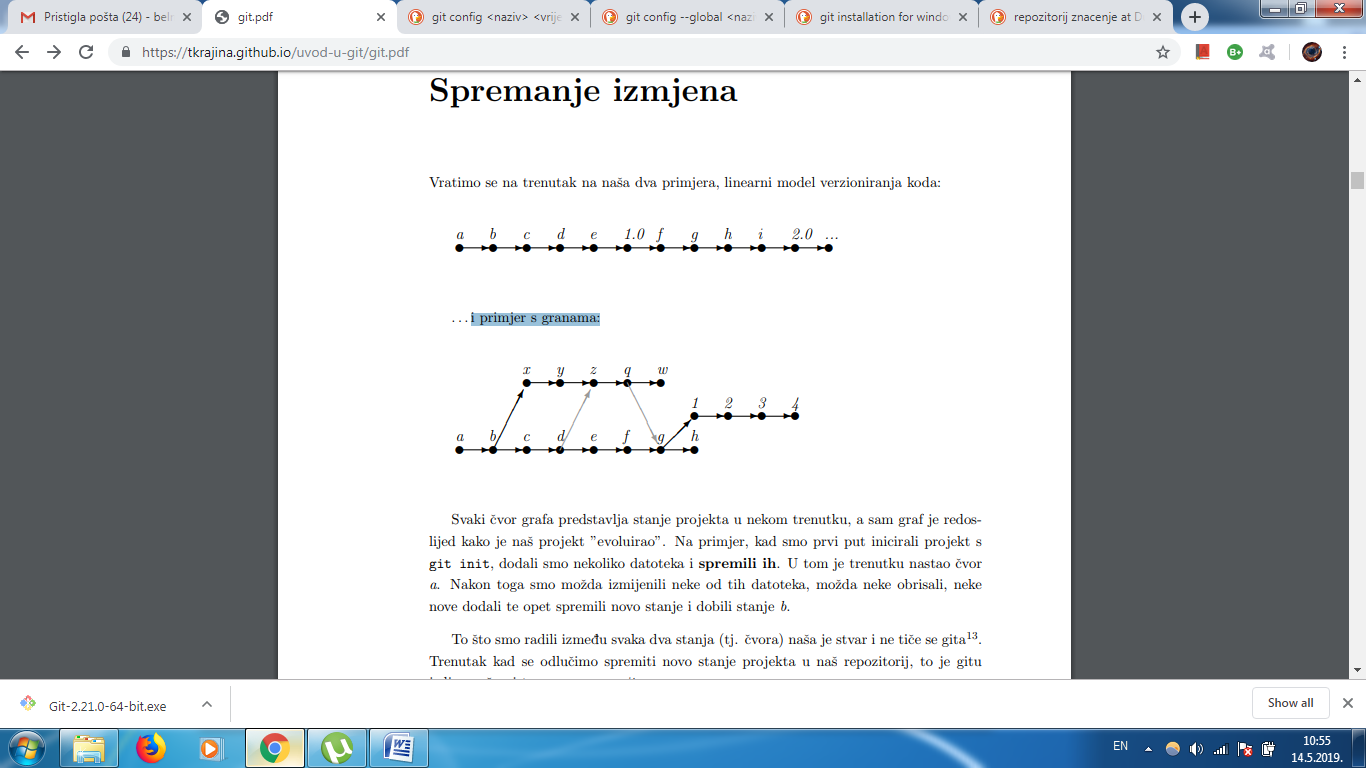
Linije koje započinju znakom *#* su komentari i git će se ponašati kao da ne postoje.

# Spremanje izmjena

Vratimo se na naš primjer, linearni model verzioniranja koda:



I primjer s granama:



Svaki čvor grafa predstavlja stanje projekta u nekom trenutku, a sam graf je redoslijed kako je naš projekat ”evoluirao”. Kada smo prvi put inicirali projekat sa *git init*, dodali smo nekoliko datoteka i spremili ih. U tom je trenutku nastao čvor *a*. Nakon toga smo možda izmijenili neke od tih datoteka, možda neke obrisali, neke nove dodali te opet spremili novo stanje i dobili stanje *b.*

Kada se odlučimo spremiti novo stanje projekta u naš repozitorij, to je gitu jedino važno i to se zove *commit.*

## Indeks

Govorimo o tome kako ćemo ”commitovati datoteku” ili ”staviti datoteku u indeks” . . . , treba imati na umu da git ne čuva ”datoteke” nego stanja, odnosno verzije datoteka. Za jednu te istu datoteku git čuva njena različita stanja kako se mijenjala kroz historiju. Mi datoteke u našem projektu mijenjamo, a sami odlučujemo u kojem su trenutku one takve da bismo ih snimili.

U gitu postoji poseban ”međuprostor” u koji se ”stavljaju” datoteke koje ćemo spremiti (commitati). Sada imamo tri različita ”mjesta” u kojima se čuvaju datoteke odnosno konkretna stanja pojedinih datoteka:

1. Git repozitorij čuva različita stanja iste datoteke (historiju datoteke).
2. Radna verzija repozitorija je stanje datoteka u našem direktoriju. Ono može biti isto ili različito u odnosu na stanje datoteka u repozitoriju.
3. Poseban ”meduprostor” za commit gdje privremeno spremamo trenutno stanje datoteka prije nego što ih commitamo.

Taj ”meduprostor za commit” zove se index (indeks).

### Spremanje u indeks

Recimo da smo promijenili datoteku *uvod.tex.* Nju možemo staviti u indeks:

git add uvod.tex

. . . i sad je status:

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD ..." to unstage)

#

# modified: uvod.tex

#

Dio u kojem piše: *Changes to be committed.* E to je popis datoteka koje smo stavili u indeks.

Nakon što smo datoteku spremili u indeks, spremni smo za commit ili možemo nastaviti dodavati druge datoteke sa *git* *add* sve dok se ne odlučimo za snimanje.

Čak i ako smo datoteku obrisali, moramo je dodati u indeks naredbom *git add*, zato što u indeks ne stavljamo u stvari datoteku nego neko njeno (izmijenjeno) stanje. Kad smo datoteku obrisali, u indeks treba spremiti novo stanje te datoteke, ”izbrisano stanje”.

*git add* ne moramo koristiti s jednom datotekom. Ako spremamo cijeli direktorij datoteka, možemo ga dodati sa:

git add naziv direktorija/\*

Ili ako želimo dodati apsolutno sve što se nalazi u našoj radnoj verziji:

git add .

### Micanje iz indeksa

Datoteku smo stavili u indeks i kasnije se predomislili, lako je iz indeksa maknemo naredbom:

git reset HEAD -- <datoteka1> <datoteka2>...

Događat će nam se situacija da smo promijenili neku datoteku, ali kasnije zaključimo da ta izmjena nije bila potrebna. I sad je ne želimo spremiti, nego vratiti u stanje kakvo je trenutno spremljeno u repozitoriju. To se može ovako:

git checkout HEAD -- <datoteka1> <datoteka2>...

## Prvi commit

Izmjene možemo spremiti sa:

git commit -m "Nova verzija"

U stringu nakon -*m* moramo unijeti komentar uz svaku promjenu koju spremamo u repozitorij. Git ne dopušta spremanje izmjena bez komentara .

Sad je status projekta opet:

$ git status

# On branch master

nothing to commit (working directory clean)

## Datoteke koje ne želimo u repozitoriju

Ovo je situacija koja se često dešava. Greškom smo u repozitorij spremili datoteku koja tamo ne treba biti. Međutim, tu datoteku ne želimo obrisati sa našeg diska nego je samo ne želimo njenu historiju imati u repozitoriju.

To se dešava kad nam editor spremi konfiguracijske datoteke koje su njemu važne, ali nisu bitne za projekat. Eclipse tako zna snimiti *.project*, a Vim sprema radne datoteke s ekstenzijama .*swp* ili .*swo*. Ako smo takvu datoteku jednom dodali u repozitorij, a naknadno smo ustanovili a nam nije potrebna, onda je prvo trebamo dodati u .*gitignore*. Nakon toga git zna da ubuduće neće biti potrebno snimati izmjene na njoj.

Ali ona je i dalje u repozitoriju. Ne želimo je obrisati s diska a ni u povijesti historiji projekta (od sad pa na dalje). Neka je to, na primjer*, test.pyc*. Postupak je:

git rm --cached test.pyc

To će nam u indeks dodati stanje kao da je datoteka obrisana iako je ostavljena netaknuta na disku. Drugim riječima *git rm --cached* sprema ”obrisano stanje” datoteke u indeks. Sad tu izmjenu treba *commit*ovati da bi git znao da od ovog trenutka nadalje datoteku može obrisati iz svoje historije.

Budući da smo datoteku prethodno dodali u .*gitignore*, git nam ubuduće neće nuditi da je *commit*ujemo. Odnosno, šta god radili s tom datotekom, git status će se ponašati kao da ne postoji.

## Historija projekta

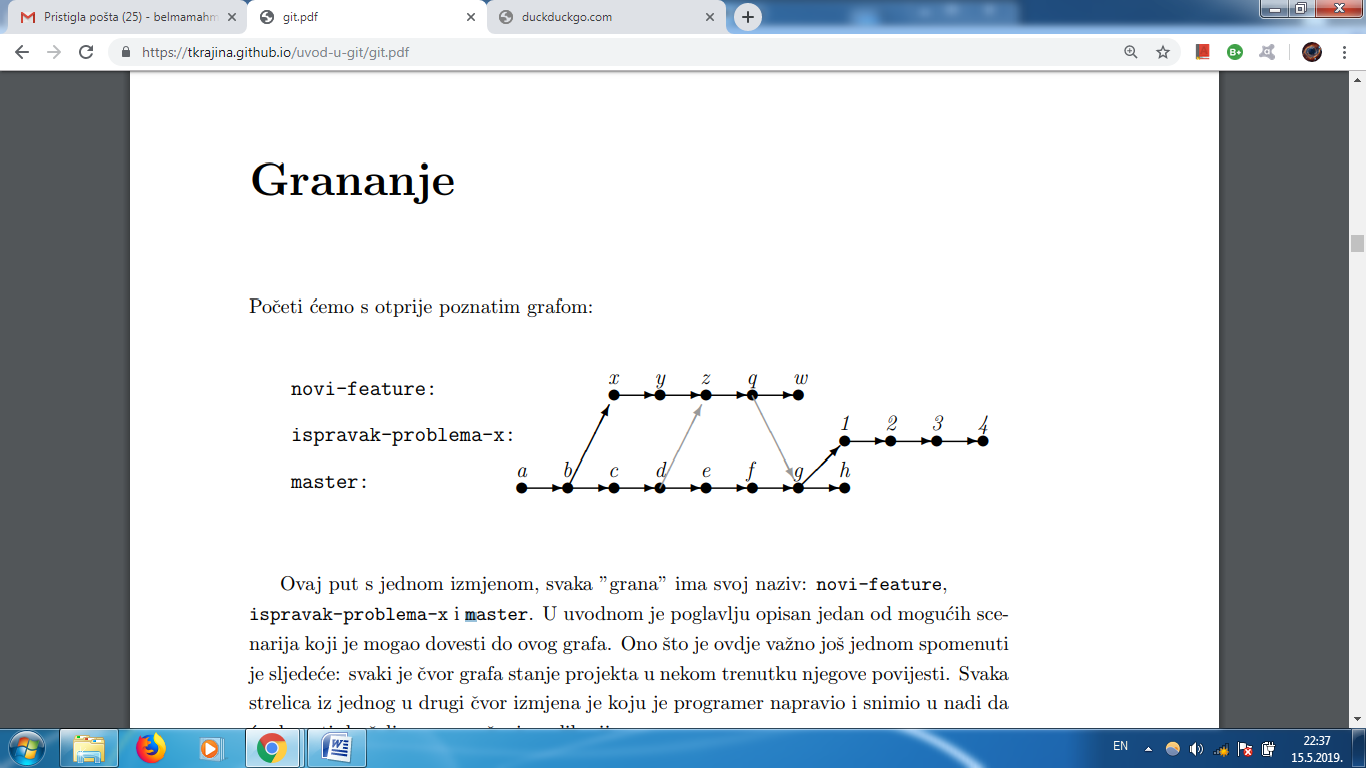
Sve prethodne *commit*ove možemo pogledati sa *git log*:

$ git log

commit bf4fc495fc926050fb10260a6a9ae66c96aaf908

U gitu svaki *commit* ima jedinstveni string koji ga identifikuje. Taj string 34 ima 40 alfanumeričkih znakova, a primjere takvih stringova možemo vidjeti sa naredbom *git log*. Na primjer, *bf4fc495fc926050fb10260a6a9ae66c96aaf908* je jedan takav.

# Grananje

Počet ćemo sa otprije poznatim grafom:

*novi-feature:*

*ispravak-problema-x:*

*master:*

Svaka ”grana” ima svoj naziv: *novi-feature, ispravak-problema-x* i *master*. svaki čvor grafa je stanje projekta u nekom trenutku njegove historije. Svaka strelica iz jednog u drugi čvor izmjena je koju je programer napravio i snimio da bi dočao do željenog ponašanja aplikacije.

## Popis grana projekta

Jedna od velikih prednosti gita je što omogućava jednostavan i brz rad s višestrukim granama. Ako želimo vidjeti koje tačno grane naš projekat trenutno ima, naredba je *git branch.* Rezultat te naredbe:

$ git branch

\* master

Znači da naš projekat trenutno ima samo jednu granu. Svaki git repozitorij u početku ima jednu jedinu granu i ona se uvijek zove *master*.

Ako smo naslijedili projekat koji je neko prethodno već granao, dobit ćemo:

$ git branch

api

development

editable-text-pages

less-compile

\* master

Ili na primjer ovako:

$ git branch

api

\* development

editable-text-pages

less-compile master

Svaki redak predstavlja jednu granu, a redak koji počinje zvjezdicom (\*) grana je u kojoj se trenutno nalazimo. U toj grani tada možemo raditi sve što i na master (commitati, gledati njenu historiju, . . .).

## Nova grana

Novu granu možemo stvoriti naredbom *git branch* *<naziv\_grane>.* Na primjer:

git branch eksperimentalna-grana

Imamo granu *eksperimentalna-grana*, ali u njoj nemamo još ni jednan čvora (commit). U toj grani sada možemo mijenjati (dodavati, brisati) datoteke, spremati ih u indeks sa *git add* i commitovati sa *git commt.*

## Prebacivanje s grane na granu

*Git branch* stvorit će napraviti samo novu granu. Prebacivanje s jedne grane na drugu granu radi se naredbom *git checkout* :

$ git checkout eksperimentalna-grana

Switched to branch ’eksperimentalna-grana’

Na glavnu se granu vraćamo sa *git checkout master*. Sada kad smo se prebacili na novu granu, možemo tu uredno *commit*ovati svoje izmjene. Sve što tu radimo neće biti vidljivo u master grani.

## Brisanje grane

Grananjem možemo doći na situacije kad ćemo se naći s previše grana. Možda smo neke grane napravili da bismo isprobali nešto novo, a to se na kraju pokazalo kao loša ideja pa smo granu napustili. Ili smo je započeli da bismo riješili neki problem, ali taj problem je prije nas riješio neko drugi. U tom slučaju granu možemo obrisati *sa git branch <naziv\_grane> -D* .

Ako je stanje grana na našem projektu sljedeće:

$ git branch

eksperimentalna-grana

\* master

...nakon:

$ git branch -D eksperimentalna-grana

Deleted branch eksperimentalna-grana (was 1658442)

. . . . novo stanje će biti:

$ git branch

\* master

Primijetimo da ne možemo obrisati master:

$ git branch -D master

error: Cannot delete the branch ’master’ which you are currently on.

Ne možemo obrisati granu na kojoj se trenutno nalazimo. Brisanjem grane ne brišemo njene commitove. Oni ostaju dio historije.