

UNIVERZITET DŽEMAL BIJEDIĆ U MOSTARU
FAKULTET INFORMACIJSKIH TEHNOLOGIJA MOSTAR

SEMINAR

Agilni software development, **GIT SCM**

Student: *Ernad Husremović, DL 2792*

Mentor: *mr. Adil Joldić*

ver: 0.9.5

Mostar, novembar 2012.

SADRŽAJ

1. Uvod	1
1.1. 'Source code management' (SCM) software	1
1.1.1. Centralizirani (mrežni) SCM	1
1.1.2. Distribuirani SCM (DSCM)	1
2. Gitlab servis i git klijent	3
2.1. Gitlab servis	3
2.2. git klijent	3
2.2.1. Gitlab ssh pristup	4
2.3. Git workflow	5
2.3.1. Novi projekat	5
2.4. Development	6
2.4.1. Prvi "commit"	7
2.4.2. Kreiranje novog "branch"-a	8
2.4.3. "Merge" promjena između dva "branch"-a	11
2.5. Public mirror "hello_ruby" projekta na Github-u	13
2.5.1. Github account	13
2.5.2. Github ssh postavke	14
2.5.3. Github kreiranje repozitorija	15
2.5.4. Kreiranje mirrora projekta na github-u	16
2.6. Gitlab "Diff" funkcija	16
2.7. Novi developer	17
2.7.1. Uključenje novog developera u projekat na gitlab-u	17
2.8. Email notifikacija	19
2.8.1. Kloniranje repozitorija	19
2.8.2. Novi developer, realizacija nove funkcije	20
2.8.3. Git log	20
2.9. Zajednički rad - gitlab "merge request"	21

2.9.1. Git pull	22
2.9.2. Brisanje privremenog "branch"-a	24
2.10. Git unutar IDE-a	24
2.10.1. Eclipse IDE	24
2.11. "Collective code ownership"	28
3. Zaključak	31
4. Literatura	32
A. Riječnik pojmova	33
B. Napomene	34
C. Software toolset	35
D. Software repozitoriji	36

Abstract

Dokument na bazi konkretnog primjera¹ objašnjava uobičajeni developerski 'workflow' pri korištenju GIT 'Source Code Management' (SCM) alata. U primjeru se koriste GIT web servisi 'Github'² i 'Gitlab'³. Prikazane su glavne operacije GIT klijenta koristeći navedene web servise.

Keywords: open source software, OSS, Source code management, SCM, DSCM, Version control, GIT

¹"HOWTO" stil

²<https://www.github.com>

³<https://gitlab.knowhow.out.ba>, <http://gitlabhq.com>

1. Uvod

1.1. ‘Source code management’ (SCM) software

‘Source code management’ (SCM) software obezbeđuje pohranu različitih verzija programskog koda u zajednički repozitorij. Iako je primarna namjena alata bilo čuvanje programskog koda, oni se mogu koristiti za pohranu (verzioranje) svih artifakta softverskog projekta uključujući dokumentaciju, dijagrame, kao i binarni kod. Ovaj software se često naziva i ‘Version control’ software. Sa stanovišta arhitekture, SCM sistemi se dijele na centralizovane i distribuirane (DSCM).

1.1.1. Centralizirani (mrežni) SCM

Najpoznatiji predstavnici su ‘CVS’ i ‘Subversion’. ‘CVS’ je među prvim SCM alatima-ima koji je postigao veliku popularnost. ‘Subversion’ je uveo značajna tehnička unapređenja, ali je arhitekturalno identičan svom predhodniku. Oba alata su ‘open source’ software (OSS).

Njihova glavna karakteristika je klasična ‘client-server’ arhitektura. Da bi se SCM operacije na klijentu izvršavale, neophodna je dostupnost SCM servera. Razlog je taj što se istorija promjena nalazi isključivo na serveru. Ova karakteristika danas se smatra ključnim nedostatkom centraliziranih SCM-ova

1.1.2. Distribuirani SCM (DSCM)

Distribuirani SCM-omi imaju potpuno različitu arhitekturu. Baza promjena nalazi se na svakom klijentu. Sami developeri određuju funkciju pojedinačnih repozitorija. Glavni repozitorij (‘upstream’) je dostupan udaljenim klijentima. Razmjena podataka (‘commit’-a) se vrši sistemom sinhronizacije repozitorija klijenta i servera (‘merge’ proces). Usljed nedostatka centralnog repozitorija, sasvim su normalne situacije u kojima se repozitoriji koji se usaglašavaju - sinhronizuju u stanju konflikta. U tom slučaju potrebno je izvršiti manuelni ”merge” proces kojim se promjene primljene iz različitih izvora usaglašavaju. Ovaj koncept otvara mogućnost da se pojave značajni problemi unutar većeg tima prilikom usaglašavanja repozitorija. Međutim, praksa je pokazala da se pravilnom primjenom obezbeđuje niz prednosti, prije svega *fleksibilnost* u radu.

Jedna od ključnih prednosti DSCM-ova je 'off-line' režim rada, kao posljedica činjenice da svaki korisnik ima sopstveni lokalni repozitorij promjena.

2. Gitlab servis i git klijent

2.1. Gitlab servis

"Gitlab"¹ je OSS projekat, po mnogo čemu "klon" popularnog "Github" servisa. Autor nudi komercijalni servis "gitlab.com" baziran na ovom serveru.

S obzirom da se radi OSS software-u, korisnici bez ograničenja mogu kreirati sopstvene "gitlab" server-e. U ovom materijalu se koristi server ["gitlab.knowhow.out.ba"](https://gitlab.knowhow.out.ba) instaliran upravo na taj način.

2.2. git cliënt

Tvorac git-a je kreator Linux-a, Linus Torvalds. Prve verzije git-a su se mogle koristiti samo "unix-like" sistemima². Popularnost i otvorenost projekta je brzo rezultirala i "native" git verzijom klijenta.

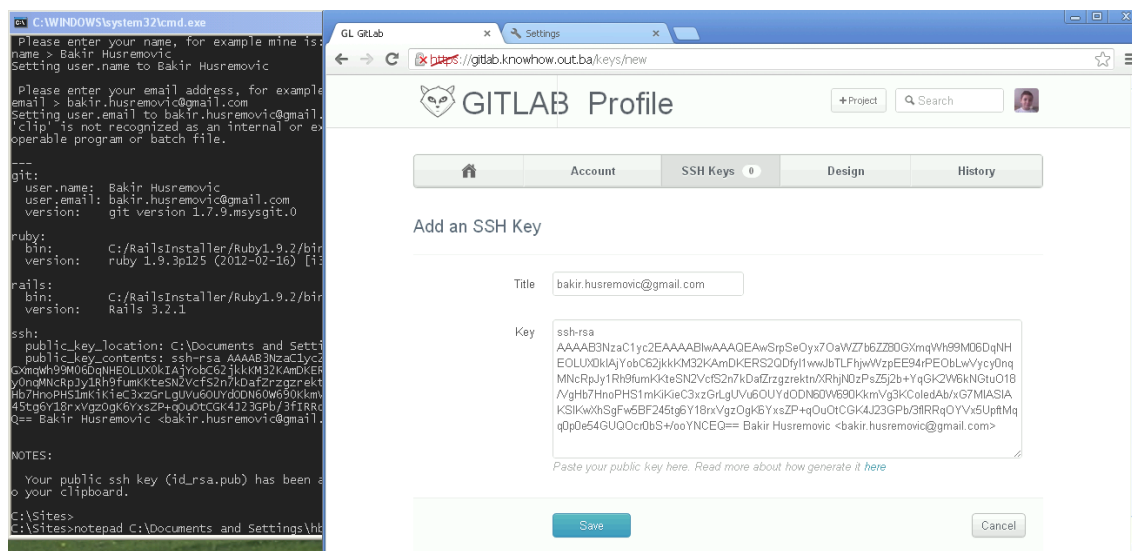
U našem primjeru je radi jednostavnosti korištenja na "Windows" klijentu korišten "Rails installer". "Rails installer" sadrži kompletno ruby/rails/git razvojno okruženje za "Windows" XP/W7 OS.

Nakon instalacije "Rails installer" traži podatke o korisniku, te kreira ssh ključ koji će se koristiti za povezivanje sa git serverom putem ssh protokola:

[illegible]

¹<http://gitlabhq.com>

²git se mogao koristiti na Windows-ima putem "cygwin" emulacijskog sloja, ali je rad sa iole većim repozitorijem bio veoma spor

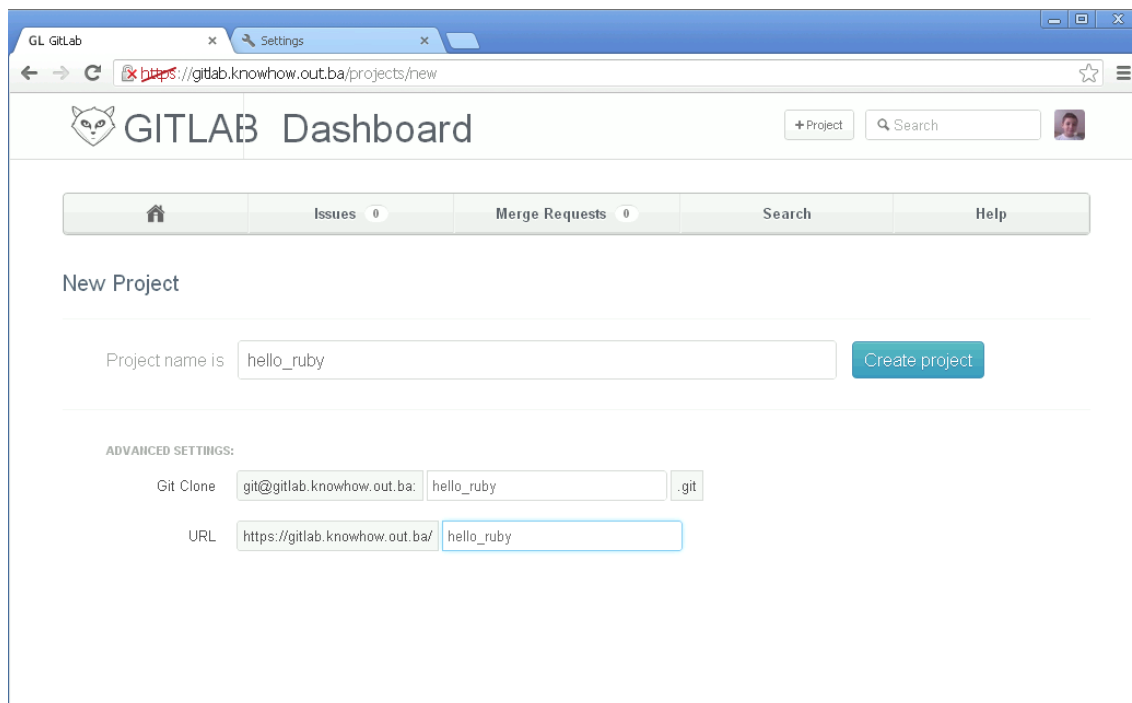


Slika 2.1: Gitlab ssh profil

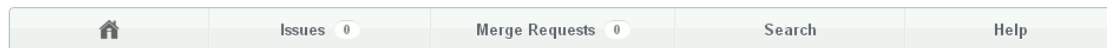
2.3. Git workflow

2.3.1. Novi projekat

Nakon podešenja ključ za udaljeni pristup, možemo kreirati naš prvi projekat "helo_ruby":



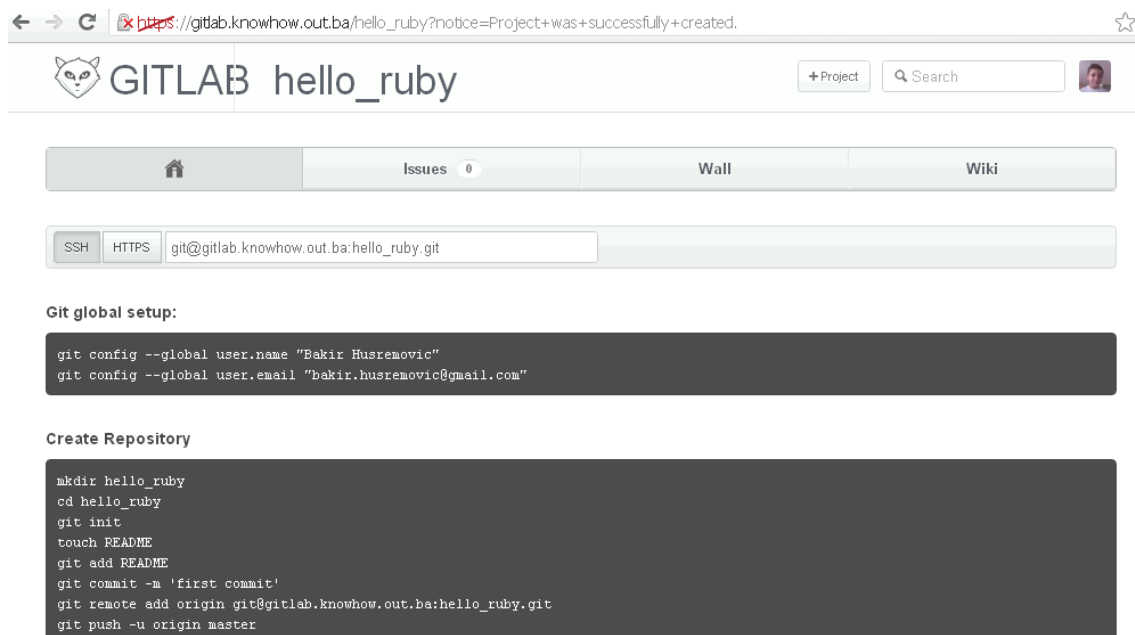
Slika 2.2: Gitlab: novi projekat



Creating project & repository. Please wait a few minutes

Slika 2.3: Gitlab: novi projekat - kreiranje git repozitorija

Nakon što je git repozitorij kreiran, dobijamo informacije o operacijama koje se obavljaju na remote klijentu za kreiranje i pristup git repozitoriju:



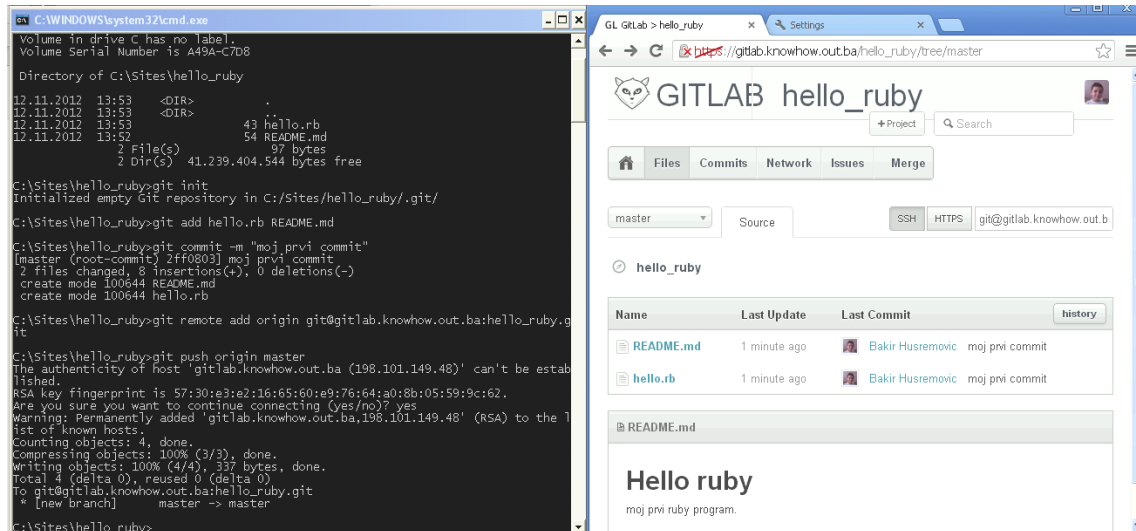
Slika 2.4: Gitlab: informacije o pristupu git repozitoriju

2.4. Development

Na "gitlab" serveru je sve spremno za rad na našem prvom projektu. Napomenimo da u ovom trenutku na projektu radi samo jedan developer - Bakir. Kao kreator, on je ujedno i vlasnik projekta.

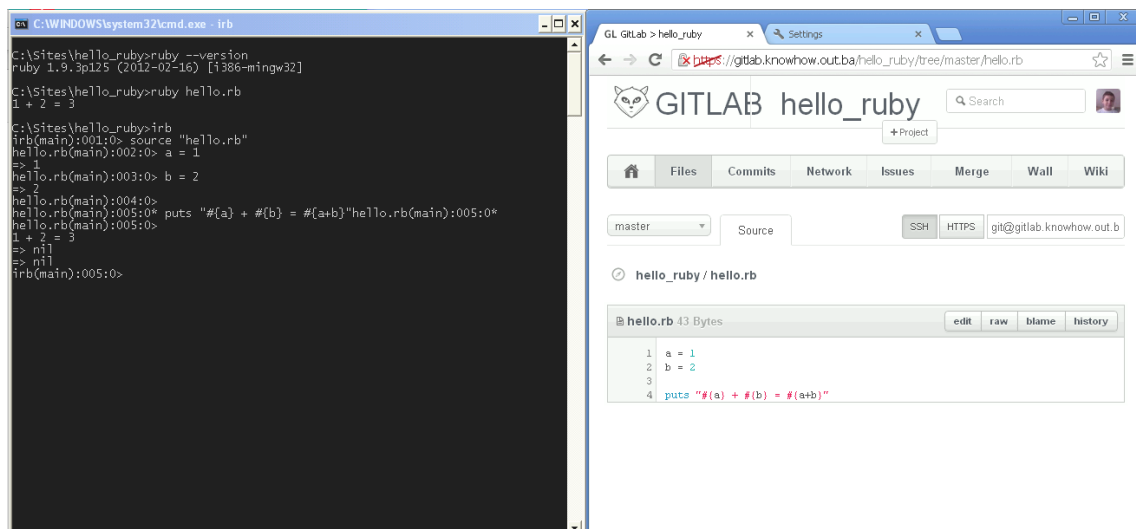
2.4.1. Prvi "commit"

Slijedi rad na developerskoj radnoj stanici. Bakir će na svom "Windows" desktopu napraviti git repozitorij, te u njega smjestiti prvu verziju aplikacije.

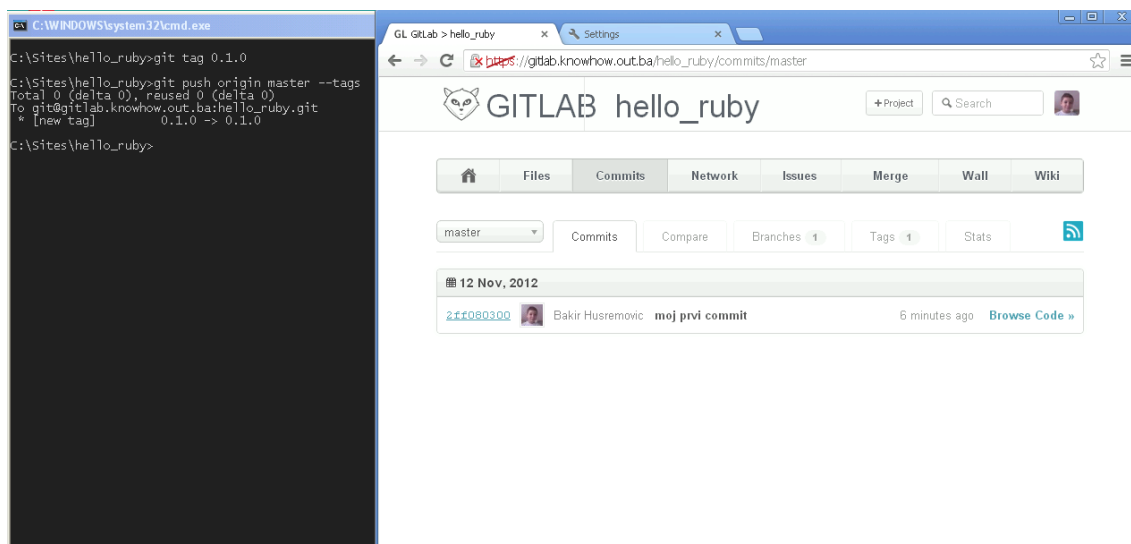


Slika 2.5: Prve operacije na git repozitoriju: init, add, commit, push

Na konzoli su napravljeni prvi testovi hello.rb programa. Takođe, na gitlab web konzoli je nakon "push" operacije programski kod "stigao" i na server:



Slika 2.6: Izvorni kod programa na gitlab serveru



Slika 2.7: "Označavamo" prvu verziju našeg software-a: release 0.1.0

Napravimo pregled do sada izvršenih operacija:

1. `git init` - kreiramo `.git` direktorij, lokalni git repozitorij
2. `git remote add origin` - navodimo lokaciju našeg udaljenog - serverskog git repozitorija
3. `git add <ime fajla>` - dodajemo fajlove u lokalni repozitorij
4. `git commit` - sve što unutar projekta dodali/promijenili smještamo u lokalni repozitorij
5. `git tag` - "tagiramo" (označavamo) verziju 0.1.0
6. `git push origin master` - šaljemo master⁴ "branch" na server

2.4.2. Kreriranje novog "branch"-a

Nakon što je realizovao prvu verziju, Bakir kreće u izradu varijante aplikacije koja će moći preuzeti parametre sa komandne linije.

Pretpostavimo da se o zahtjevnom programskom zahvatu (rad tokom više radnih dana). Iz tog razloga ćemo otvoriti poseban branch "parametri". Na taj način će tokom rada na "parametar" branch-u "master" branch ostati funkcionalan.

Kreiramo novi branch:

```
C:/Sites/hello_ruby$ git branch parametri
```

⁴novokreirani git repozitorij uvijek ima "master" branch - to je podrazumjevani (eng. default) branch

```
origin/HEAD -> origin/master
origin/master
origin/parametri
```

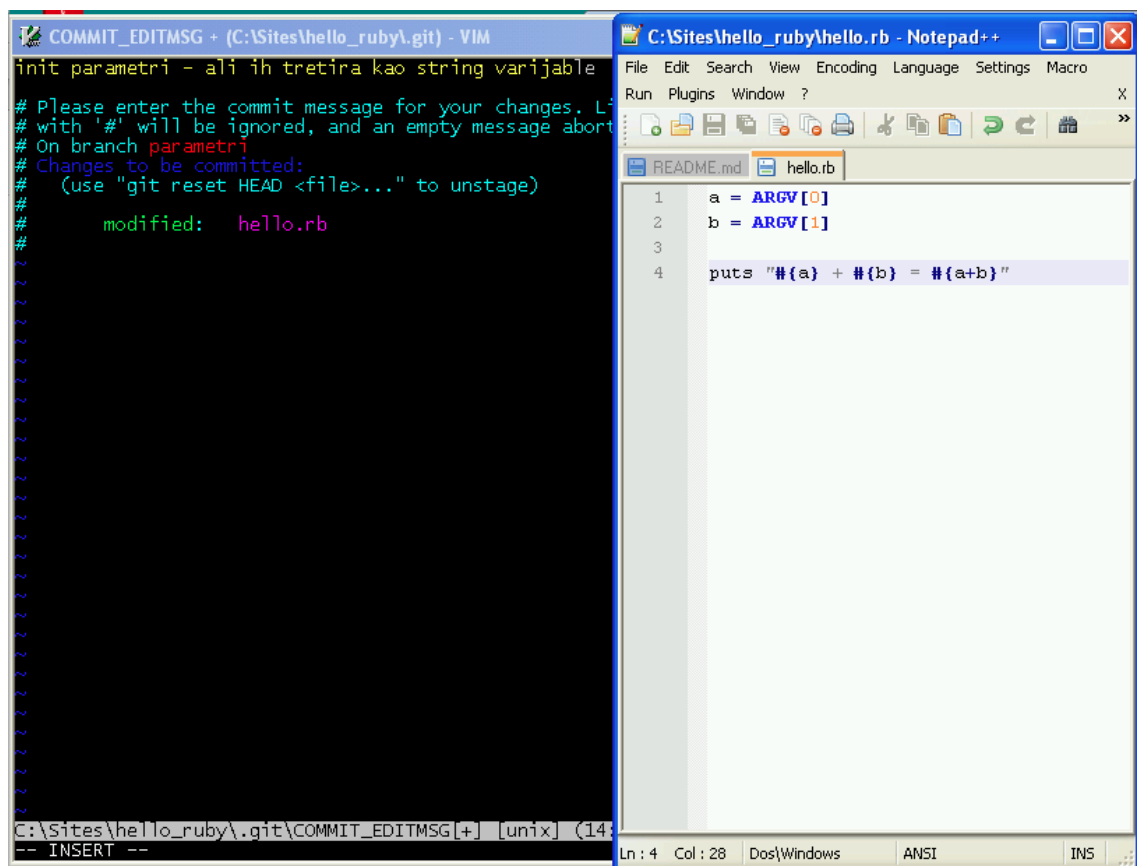
Prebacujemo se na rad sa "parametri" branch-om.

```
C:/Sites/hello_ruby$ git checkout parametri
```

```
Branch parametri set up to track remote branch parametri from origin.
Switched to a new branch 'parametri'
```

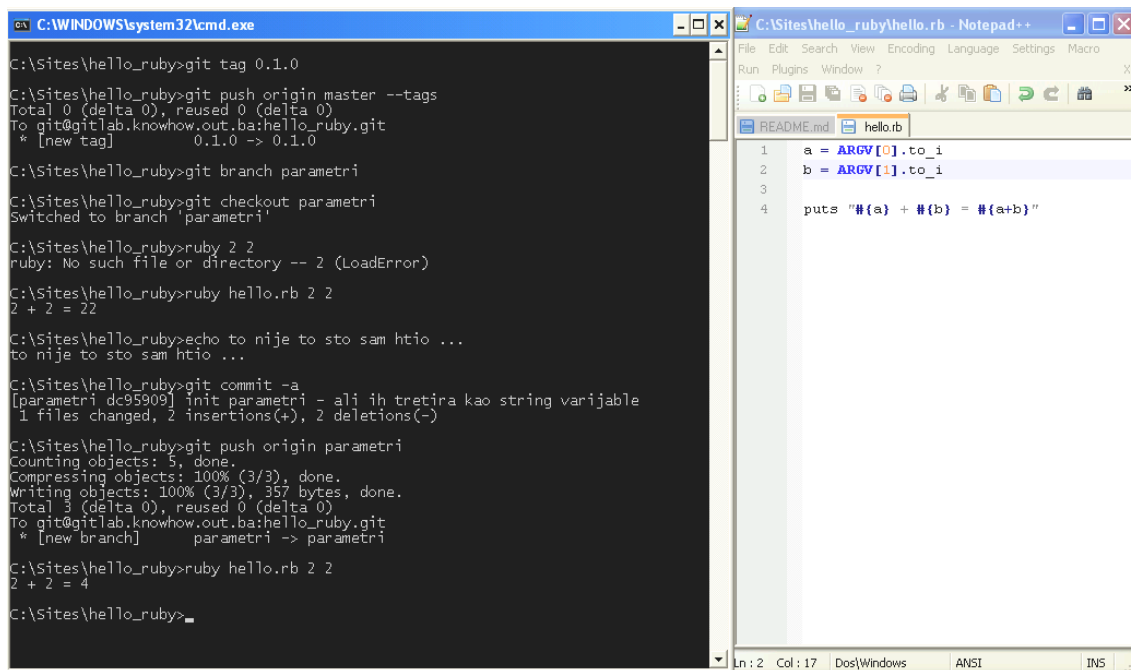
Pišemo novu verziju aplikacije, pa je testiramo.

Tokom pisanja nove verzije aplikacije, vršimo "commit" promjena u lokalni repozitorij:



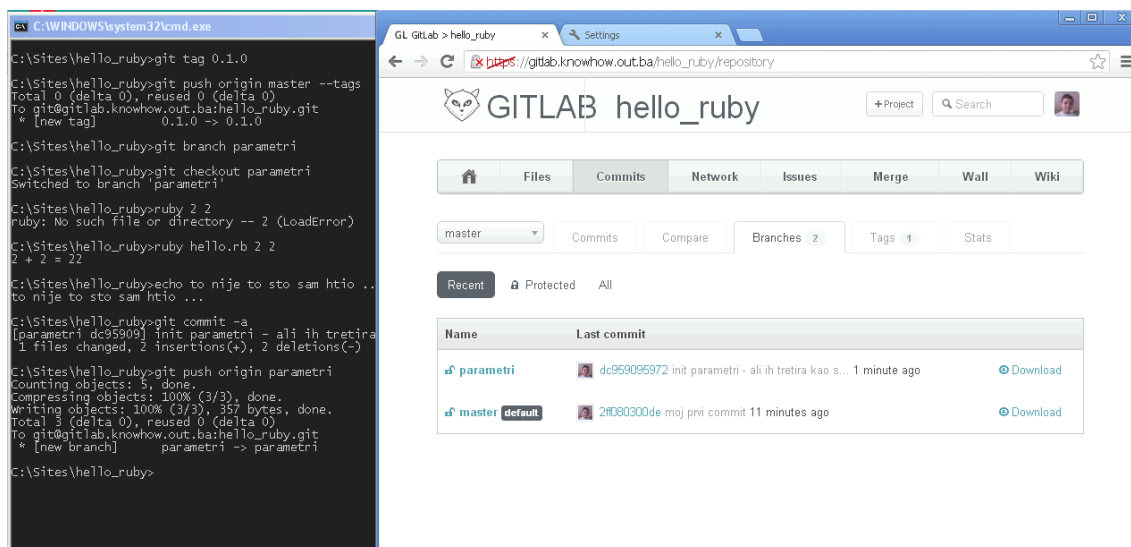
Slika 2.8: git commit, unos "commit" poruke sa "vim" editorom

"Commit" se vrši nakon svakog značajnijeg seta promjena na projektu. Između dva "commit"-a ne treba praviti velike pauze. Poželjno je da vrijeme između dva commit-a ne bude duže od par sahata. Treba izbjegavati velike "commit"-e. Dugački commiti su teški za praćenje i otklanjanje grešaka.



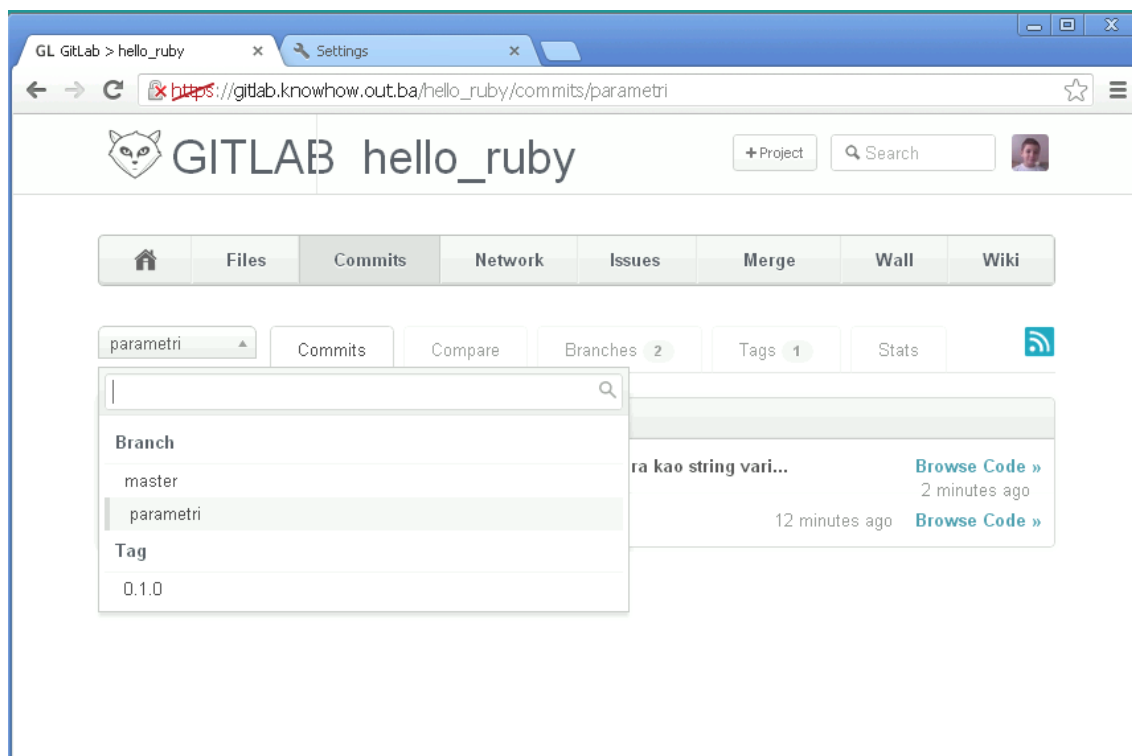
Slika 2.9: Testiranje aplikativnog koda "parametri" branch-a

Kada smo završili sa pisanjem nove verzije aplikacije, vršimo "push" na server:



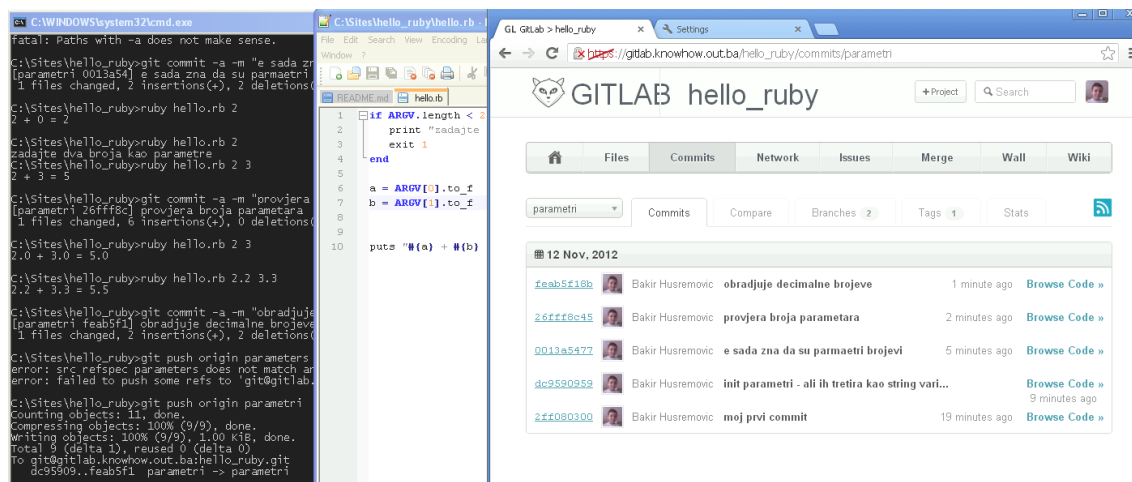
Slika 2.10: Novi branch "parametri" na gitlab-u

Na web konzoli vidimo sve brach-ove i tag-ove našeg projekta:



Slika 2.11: Gitlab - pregled "branch"-ova i "tag"-ova

Kada se pozicioniramo na branch, vidimo sve "commit"-e izvršene na tom branch-u:



Slika 2.12: Gitlab commit-i na parametar branch-u

2.4.3. "Merge" promjena između dva "branch"-a

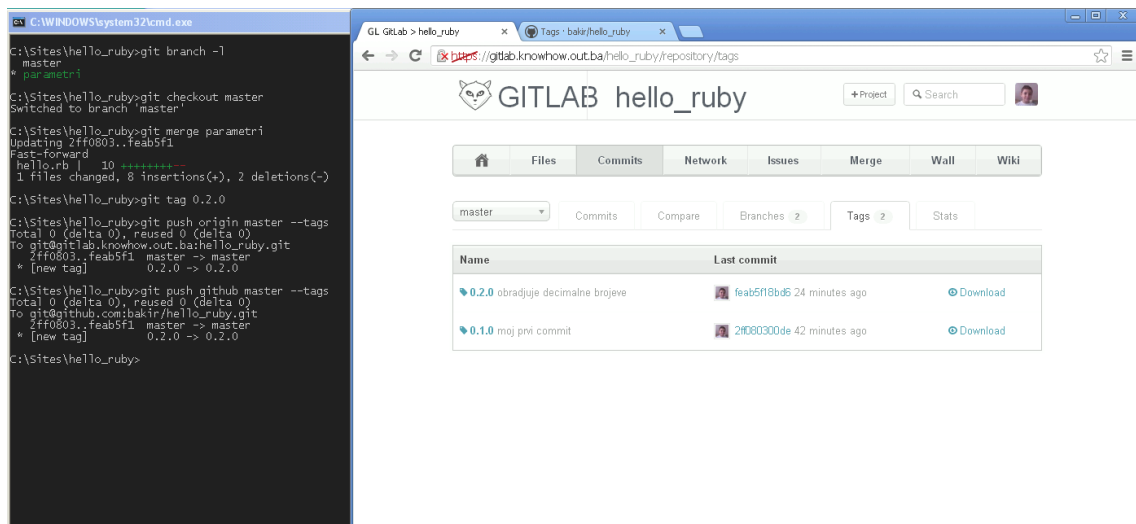
Nakon što su planirane operacije unutar "parametar" branch-a završene, promjene integrišemo u glavni - "master" branch.

To vršimo na sljedeći način:

1. `git checkout master` - prebacujemo se na "master" brach

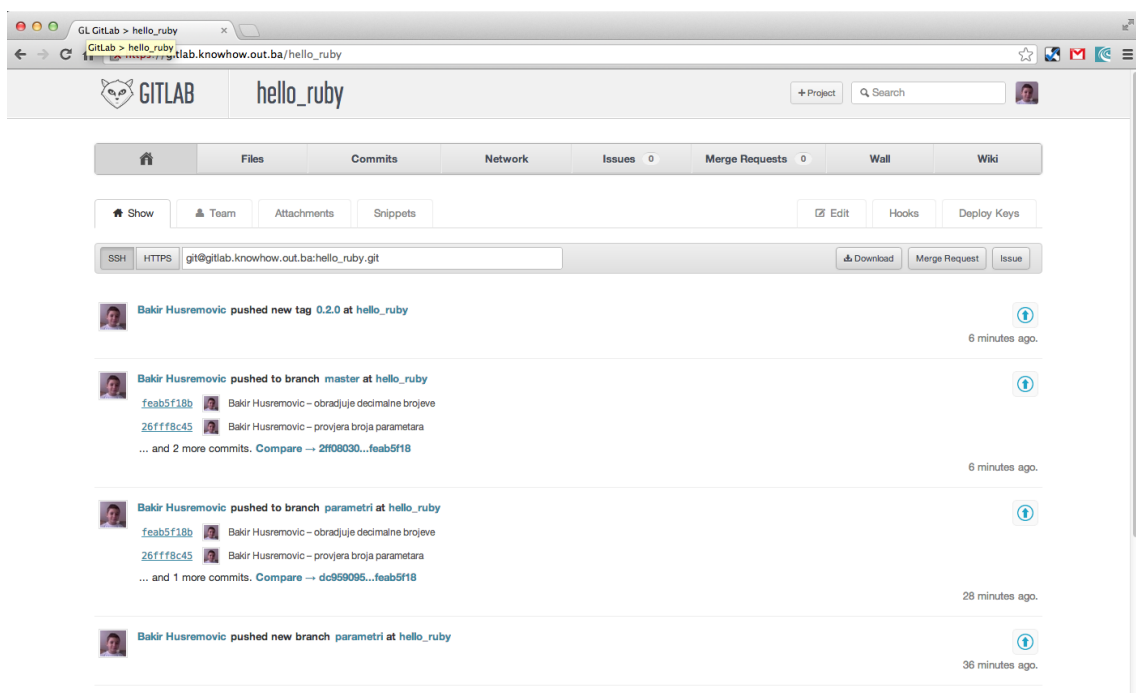
2. `git merge parametri` - vršimo vršimo "merge" (spajanje, integraciju) promjena iz "parametri" u master branch
3. `git tag 0.2.0` - označavamo verziju 0.2.0
4. `git push master -tags` - pushiramo master branch, sa tagom na server

Na gitlab serveru provjeravamo da li su promjene "došle" na server:



Slika 2.13: Gitlab pregled tagova

Pregled svih operacija koje su do sada izvršene na projektu nalazi se na [URL-u projekta](#):



Slika 2.14: Glavne aktivnosti na projektu

2.5. Public mirror "hello_ruby" projekta na Github-u

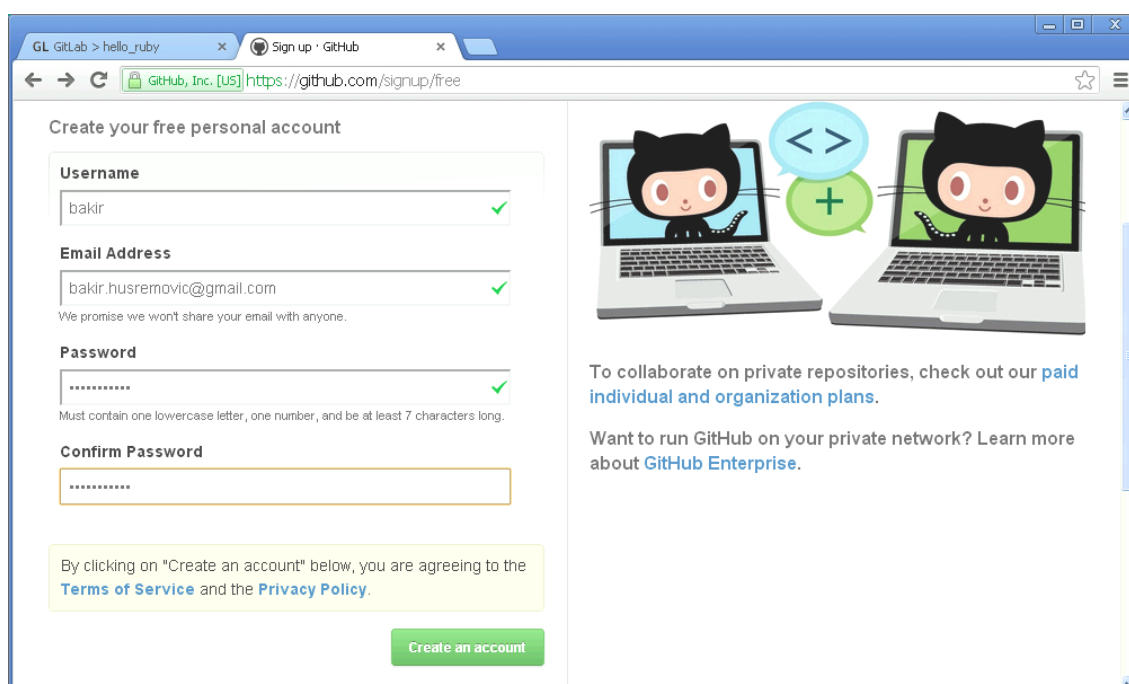
Glavni repozitorij projekta se nalazi na gitlab serveru. Međutim, "gitlab" ne omogućava anonimni pristup repozitorijima.

Ništa nas ne sprječava da na drugom git serveru kreiramo mirror našeg repozitorija.

To ćemo i učiniti. Otvorićemo account na github-u i na njemu kreirati hello_ruby projekat.

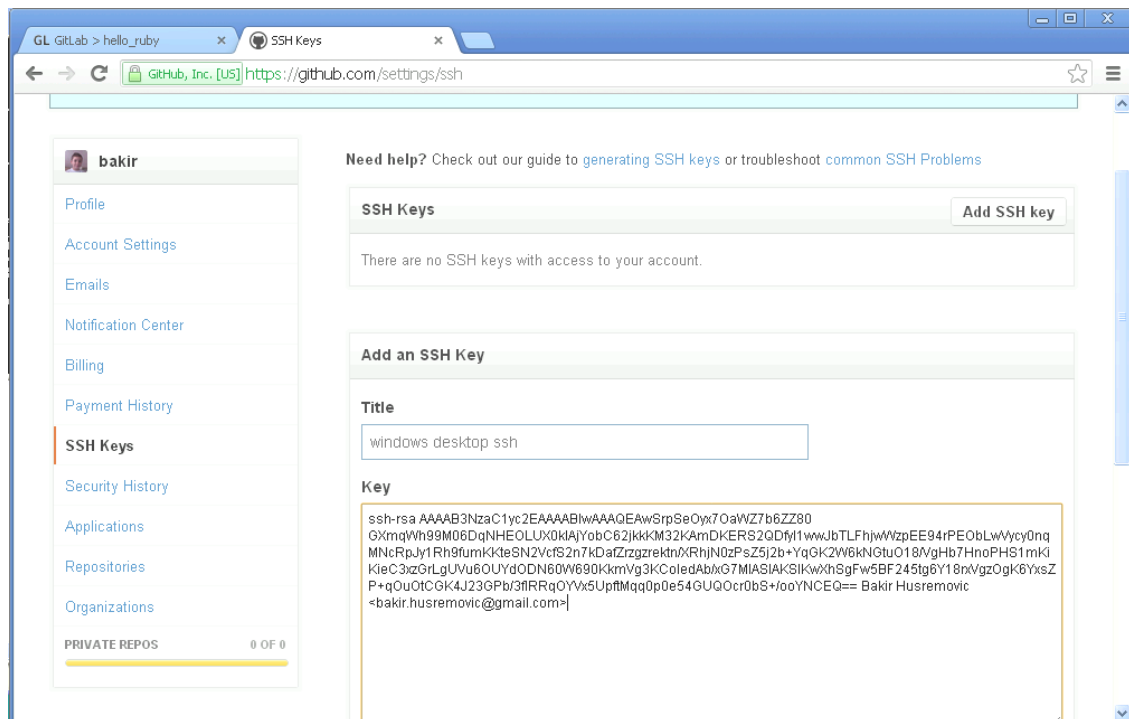
2.5.1. Github account

Kreiranje novog korisničkog računa na github-u:

The image is a screenshot of a web browser displaying the GitHub sign-up page. The browser's address bar shows the URL 'https://github.com/signup/free'. The page title is 'Sign up · GitHub'. The main heading is 'Create your free personal account'. There are four input fields: 'Username' with the value 'bakir', 'Email Address' with 'bakir.husremovic@gmail.com', 'Password' (masked with dots), and 'Confirm Password' (also masked). Each of the first three fields has a green checkmark to its right. Below the password fields, a note states: 'Must contain one lowercase letter, one number, and be at least 7 characters long.' At the bottom of the form, there is a line of text: 'By clicking on "Create an account" below, you are agreeing to the [Terms of Service](#) and the [Privacy Policy](#).' Below this is a green button labeled 'Create an account'. To the right of the form, there is an illustration of two laptops with the GitHub Octocat logo on their screens, connected by speech bubbles containing code symbols (<>) and a plus sign (+). Below the illustration, there is text: 'To collaborate on private repositories, check out our [paid individual and organization plans](#).' and 'Want to run GitHub on your private network? Learn more about [GitHub Enterprise](#).'

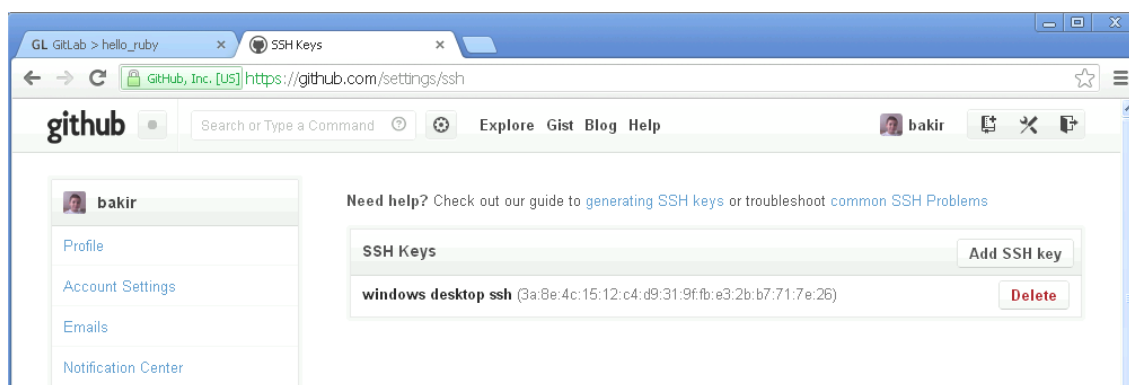
Slika 2.15: Github prijava

2.5.2. Github ssh postavke



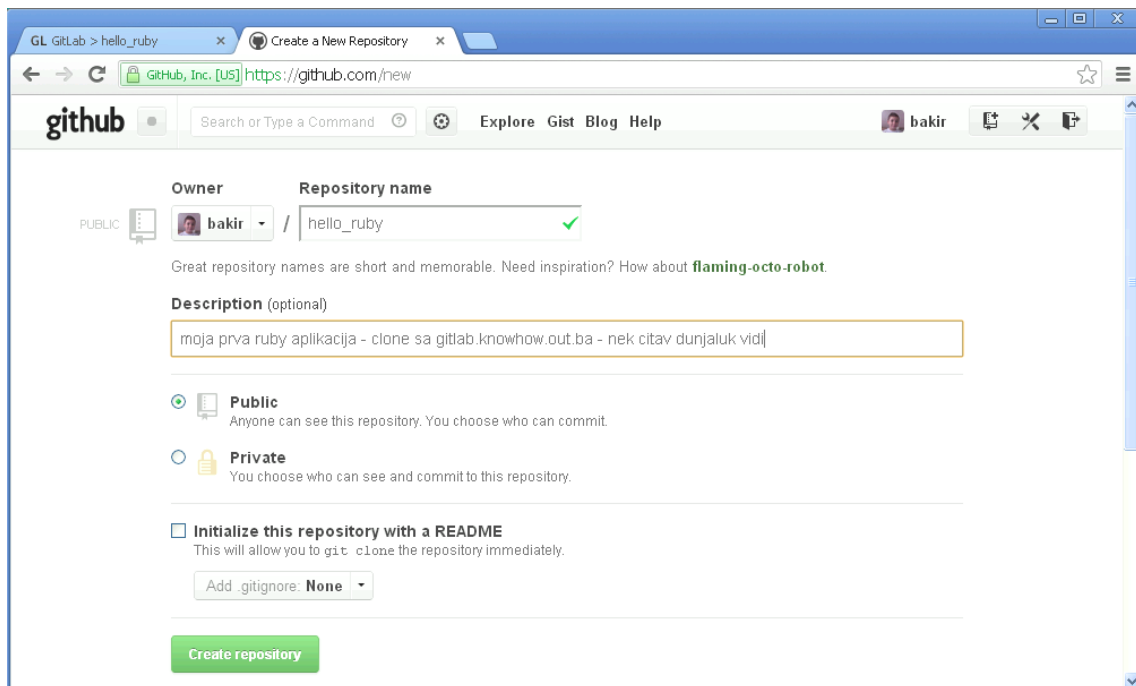
Slika 2.16: Github ssh profil

Ako razvoj obavlja na više radnih stanica, developer za svaku od njih dodaje poseban ssh ključ.



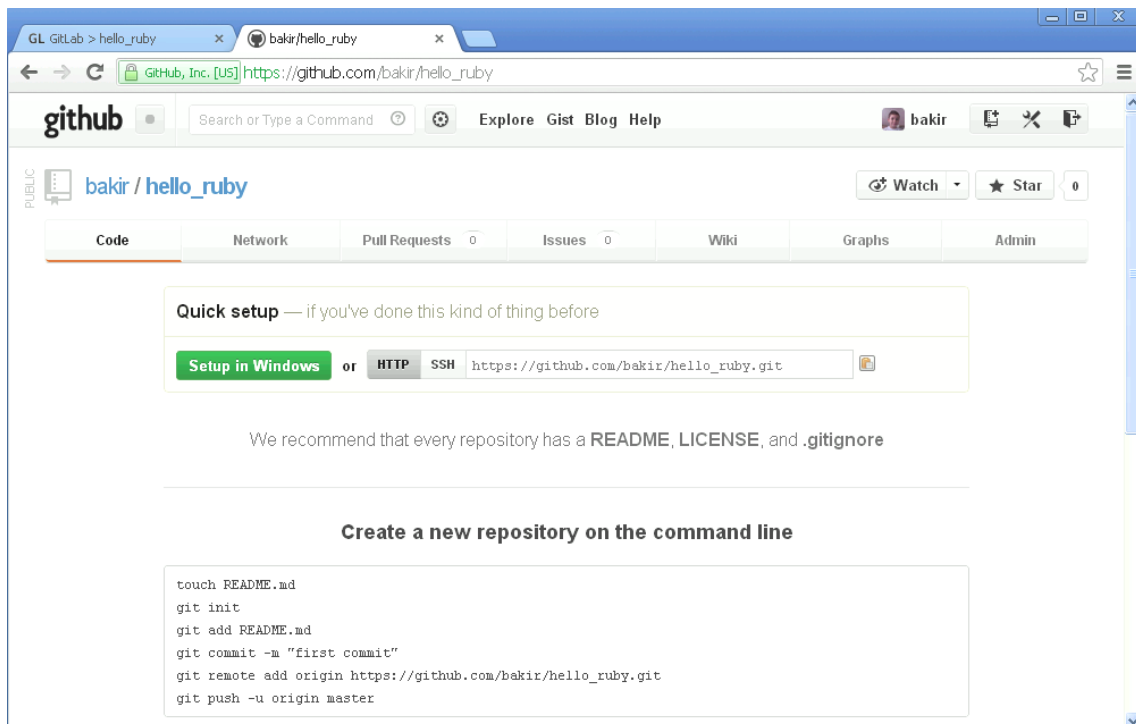
Slika 2.17: Github pregled aktivnih ssh ključeva

2.5.3. Github kreiranje repozitorija



Slika 2.18: Github novi repozitorij

Nakon kreiranja servis nam prikazuje git operacije za uspostavljanje git repozitorija na radnoj stanici developera:

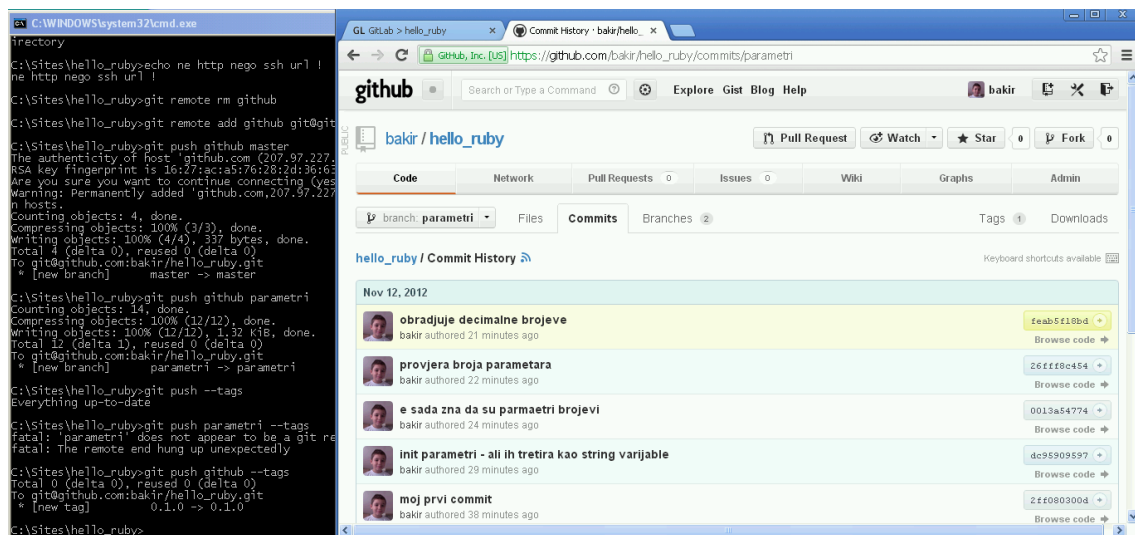


Slika 2.19: Github novi repozitorij - informacije o korištenju

2.5.4. Kreiranje mirrora projekta na github-u

S obzirom da mi već imamo remote "origin" koji ukazuje na gitlab server, radimo modifikaciju linija koje nam je predložio github:

1. `git remote add github gitlab@github.com:bakir/hello_ruby.git`
- podesi github remote repozitorij
2. `git push github master -tags` - pošalji master branch na github, zajedno sa tagovima
3. `git push github parametri` - pošalji parametri branch na github



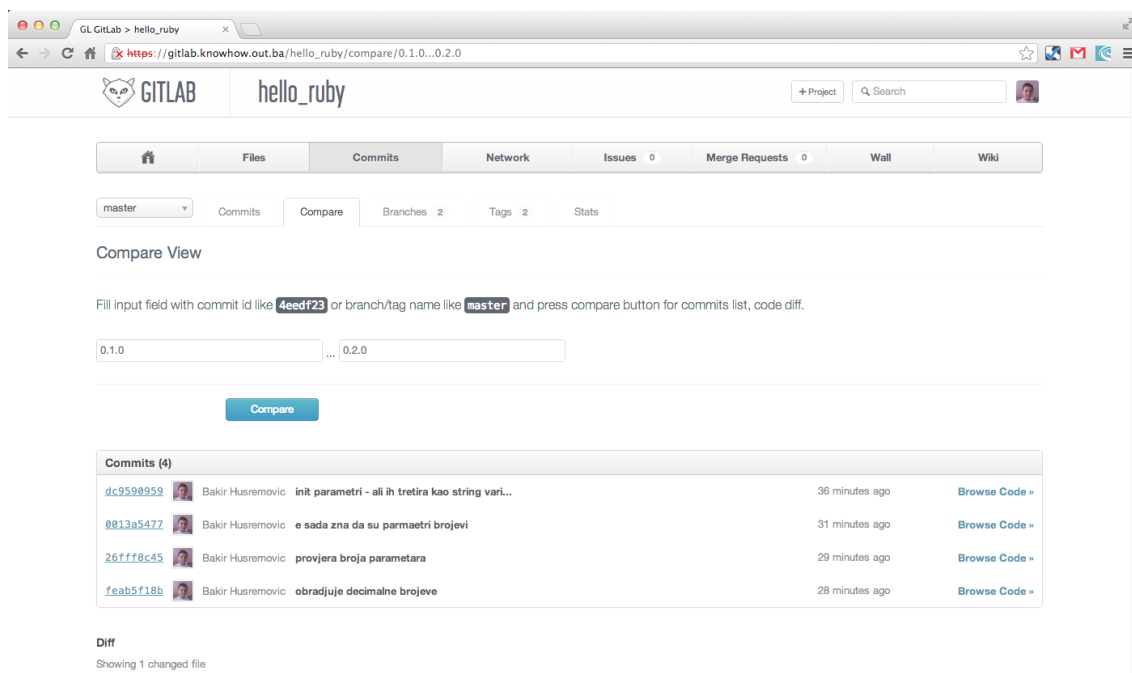
Slika 2.20: Repozitorij na github-u nakon "push" operacija

Na kraju, imamo github mirror našeg projekta https://github.com/bakir/hello_ruby

2.6. Gitlab "Diff" funkcija

Gitlab nam obezbeđuje jednostavan pregled diff-ova (razlika) između različitih tačaka unutar git repozitorija.

Te tačke mogu biti stanje branch-a ili tag-a. Pogledajmo "diff" između verzija 0.1.0 i 0.2.0:



Slika 2.21: Gitlab uporedi dvije verzije (dva tag-a)

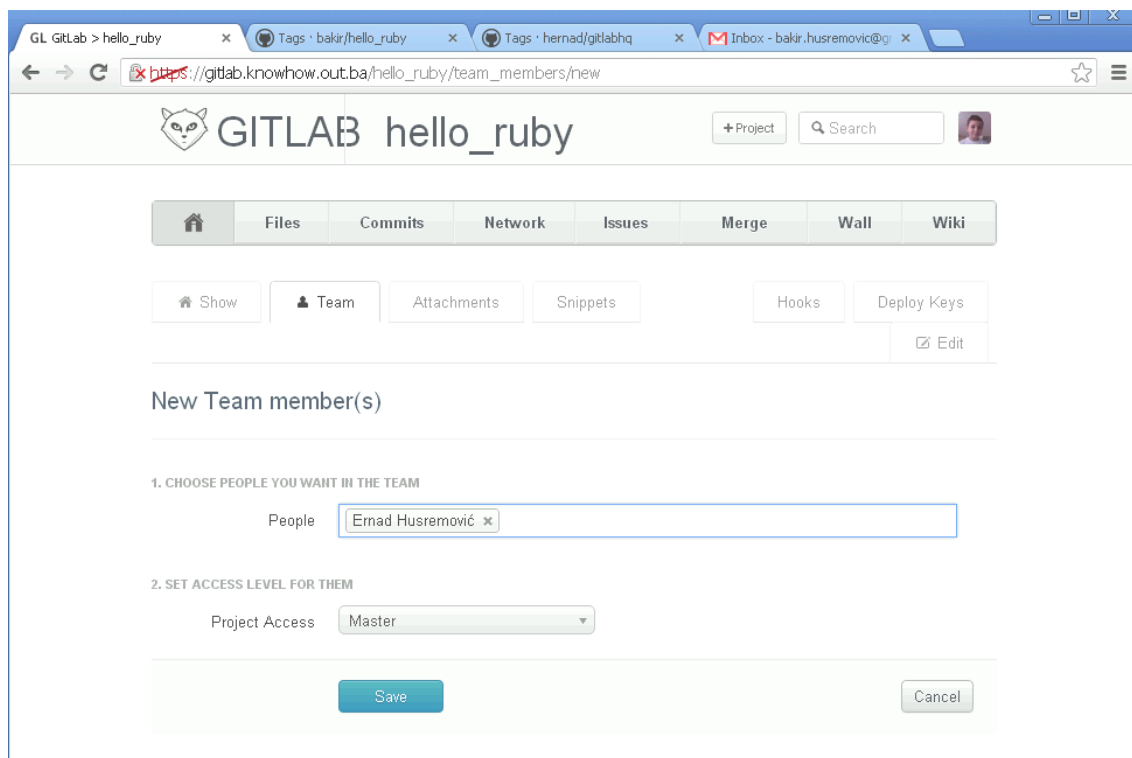


Slika 2.22: Gitlab uporedi dvije verzije - diff

2.7. Novi developer

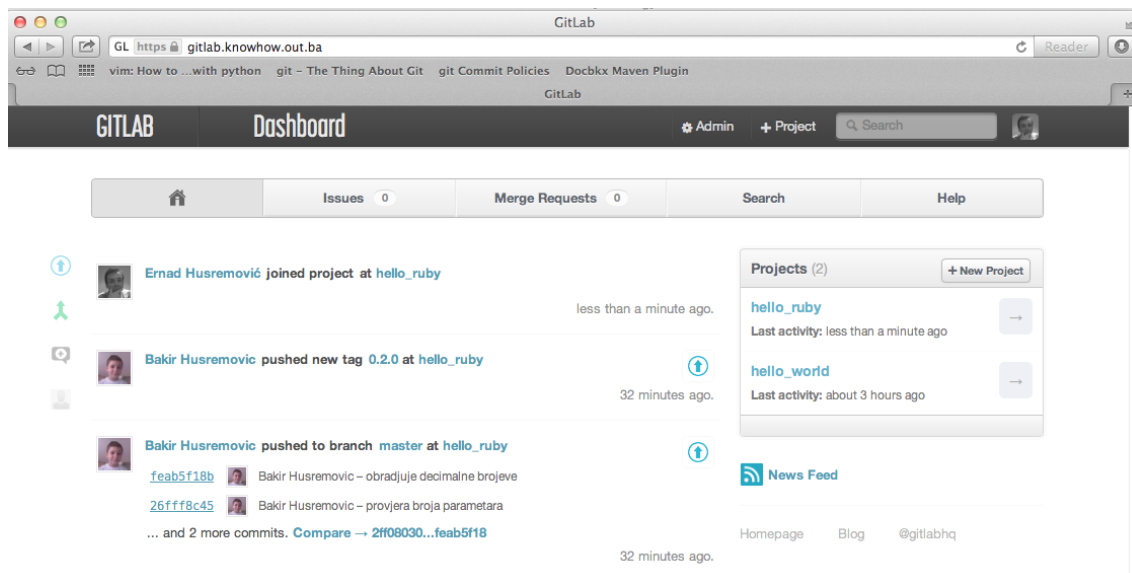
2.7.1. Uključenje novog developera u projekat na gitlab-u

Vlasnik projekta "hello_ruby" Bakir uključuje novog developera Ernad u projekat:



Slika 2.23: Gitlab: dodaj novog člana u projekat

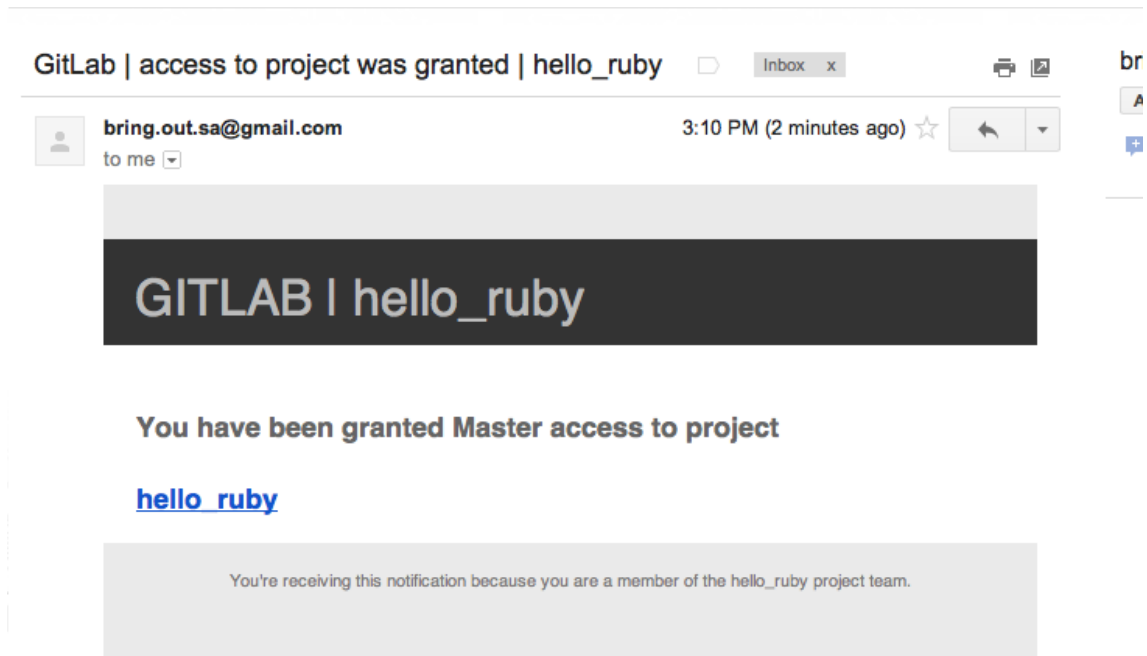
Pri tome mu daje maksimalna prava (Master uloga).



Slika 2.24: Gitlab dodaj novog člana u projekat 2

2.8. Email notifikacija

Developer primaju email notifikacije o svim bitnim događajima na projektu. Tako Ernad prima informaciju o uključenju u "hello_ruby":



Slika 2.25: Gitlab email notifikacija

2.8.1. Kloniranje repozitorija

Prvi korak novog developera je kloniranje repozitorija projekta na svoj desktop.

Novi developer koristi Ubuntu linux desktop.

```
hernad@agile_dev_env$
```

```
git clone git@gitlab.knowhow.out.ba:hello_ruby.git
```

```
Cloning into 'hello_ruby'...
remote: Counting objects: 16, done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 16 (delta 1), reused 0 (delta 0)
Receiving objects: 100% (16/16), done.
Resolving deltas: 100% (1/1), done.
```

Pregled branchova na udaljenom repozitoriju:

```
hernad@hello_ruby$ git branch -r
```

```
origin/HEAD -> origin/master
origin/master
origin/parametri
```

2.8.2. Novi developer, realizacija nove funkcije

Ernad je senior developer. Uočava da njegov mladi kolega Bakir nije obradio bitne stavke kod obrade ulaznih parametara putem konzole.

Otvora novi branch "babo", da ne bi remetio stanje u glavnom branchu-u.

```
hernad@hello_ruby$ git branch babo
```

```
hernad@hello_ruby$ git checkout babo
```

2.8.3. Git log

Vrši niz promjena. Pregled tih promjena se može pogledati usa git log operacijom:

Pregled svog "commit" log-a:

```
hernad@hello_ruby$ git log
```

```
commit 585688af924873852d72a0b9ccad16952c95fef0
Author: Ernad Husremovic <hernad@bring.out.ba>
Date: Mon Nov 12 15:42:26 2012 +0100

    uputstvo

commit 71f0f17df4b4d0dcbca50034d638bc6bfae50424
Author: Ernad Husremovic <hernad@bring.out.ba>
Date: Mon Nov 12 15:38:46 2012 +0100

    zaokruzenje pravi probleme

    ruby hello.rb
    treba zadati dva broja kao parametre aplikacije
    Unesite varijablu a: 1.1

    Unesite varijablu b: 2.2
    1.1 + 2.2 = 3.30000000000000003

commit 8e1c0bd154dd6a57d67266aee5f1f005c4aea603
Author: Ernad Husremovic <hernad@bring.out.ba>
Date: Mon Nov 12 15:35:18 2012 +0100

    ako ne navedes parametre, procitaj ih sa konzole

commit feab5f18bd616ea7b4278024579e0a7aa728f448
Author: Bakir Husremovic <bakir.husremovic@gmail.com>
Date: Mon Nov 12 14:15:45 2012 +0100
```


obradjuje decimalne brojeve

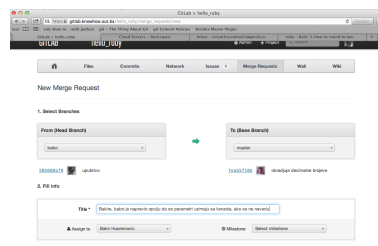
Kada završi promjene, Ernad vrši "push" svog brancha na server:

```
hernad@hello_ruby$ git push origin babo
```

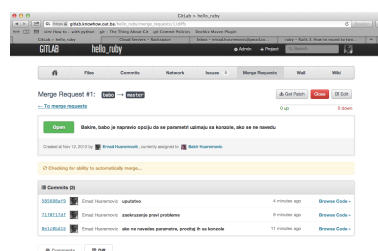
```
Counting objects: 12, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 1.42 KiB, done.
Total 9 (delta 1), reused 0 (delta 0)
To git@gitlab.knowhow.out.ba:hello_ruby.git
* [new branch] babo -> babo
```

2.9. Zajednički rad - gitlab "merge request"

Ernad informiše bakira putem gitlab "Merge request" opcije od svom radu:

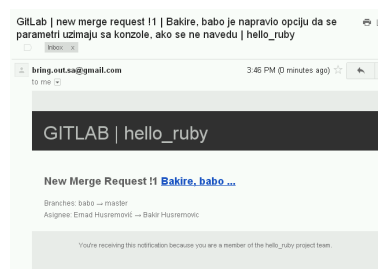


Slika 2.26: Merge request



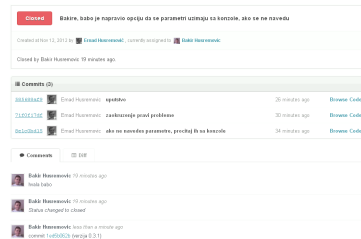
Slika 2.27: Merge request

Bakir prima informaciju:



Slika 2.28: Bakir je o "merge request"-u informisan putem email-a

Te odgovara pozitivno tako što vrši merge "babo" branch-a u "master" branch.



Slika 2.29: Merge request, Bakirov odgovor

2.9.1. Git pull

Git "pull" operacija uzima promjene sa servera. U slučaju novog branch-a, git klijent informiše o tome da se pojavio novi "branch", te da developer treba poduzeti odgovarajuće operacije da bi taj branch integrirao u svoj repozitorij:

```
C:/Sites/hello_ruby> git pull
```

```
remote: Counting objects: 12, done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 9 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (9/9), done.
From gitlab.knowhow.out.ba:hello_ruby
 * [new branch] babo -> origin/babo
You asked me to pull without telling me which branch you
want to merge with, and 'branch.master.merge' in
your configuration file does not tell me, either. Please
specify which branch you want to use on the command line and
try again (e.g. 'git pull <repository> <refspec>').
See git-pull(1) for details.
```

If you often merge with the same branch, you may want to use something like the following in your configuration file:

```
[branch "master"]
  remote = <nickname>
  merge = <remote-ref>

[remote "<nickname>"]
  url = <url>
  fetch = <refspec>
```

See git-config(1) **for** details.

Pregled svih udaljenih branchova:

```
C:/Sites/hello_ruby> git branch -r
```

```
github/master
github/parametri
origin/babo
origin/master
origin/parametri
```

Bakir se operacijom "checkout" brancha "babo" povezuje sa udaljenim brahno-om istog imena:

```
C:/Sites/hello_ruby> git checkout babo
```

```
Branch babo set up to track remote branch babo from origin.
Switched to a new branch 'babo'
```

Njegov radni branch je sada "babo":

```
C:/Sites/hello_ruby> git branch -l
```

```
* babo
  master
  parametri
```

Međutim, on ne želi raditi na tom branchu, nego ga integrira u master branch operacijom git merge. Prvo se prebacuje na "master":

```
C:/Sites/hello_ruby> git checkout master
```

```
Switched to branch 'master'
```

pa onda vrši merge babo branch-a:

```
C:/Sites/hello_ruby> git merge babo
```

```
Updating feab5f1..585688a
Fast-forward
 README.md | 36 ++++++
 hello.rb  | 38 ++++++
 2 files changed, 67 insertions(+), 7 deletions(-)
```

Bakir postavlja novu verziju aplikacije 0.3.0:

```
C:/Sites/hello_ruby> git tag 0.3.0
```

Sve promjene koje je napravio (merge u master branch, novi tag 0.3.0) šalje na server:

```
C:/Sites/hello_ruby> git push github -tags
```

```
Counting objects: 21, done.
Compressing objects: 100% (18/18), done.
Writing objects: 100% (18/18), 2.37 KiB, done.
Total 18 (delta 4), reused 0 (delta 0)
To git@github.com:bakir/hello_ruby.git
 * [new tag] 0.3.0 -> 0.3.0
 * [new tag] 0.3.1 -> 0.3.1
```

2.9.2. Brisanje privremenog "branch"-a

Nakon merđžiranja branch-a koji je bio privremene prirode, radi preglednosti vršimo njegovo brisanje.

Brišemo branch na lokalnom repozitoriju:

```
C:/Sites/hello_ruby> git branch babo -d
```

```
Deleted branch babo (was 585688a).
```

Pogledajmo pregled remote branchova-a:

```
C:/Sites/hello_ruby> git branch -r
```

```
github/master  
github/parametri  
origin/babo  
origin/master  
origin/parametri
```

Brišemo i remote branch babo:

```
C:/Sites/hello_ruby> git branch -rd origin/babo
```

```
Deleted remote branch origin/babo (was 585688a).
```

Šaljemo promjene o brisanju na server. Naglašavamo operaciju "delete":

```
C:/Sites/hello_ruby> git push origin -delete babo
```

```
To git@gitlab.knowhow.out.ba:hello_ruby.git  
- [deleted] babo
```

2.10. Git unutar IDE-a

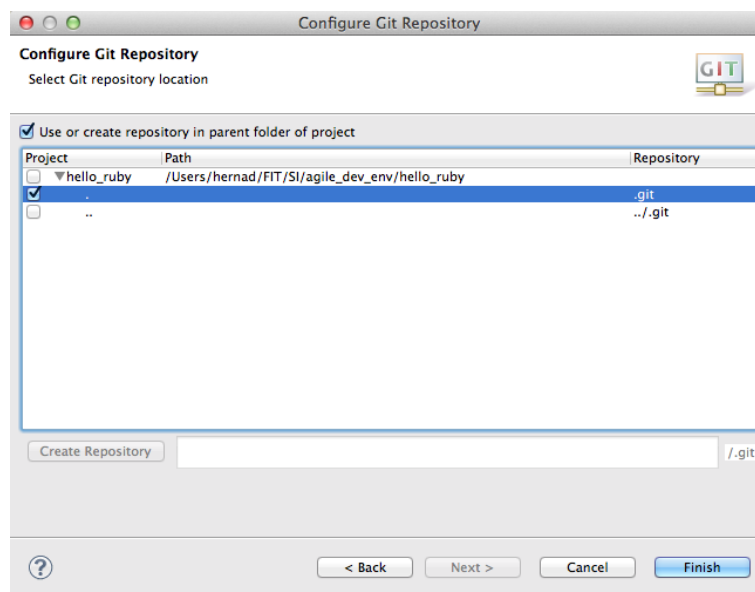
Git ima podršku za sve poznate IDE-alate:

- Microsoft visual studio "GIT Source Control Provider" - <http://gitssc.codeplex.com>
- JetBrains Idea - <http://www.jetbrains.com/idea/webhelp/using-git-integration.html>
- Netbeans - <http://netbeans.org/kb/docs/ide/git.html>
- Eclipse - <http://www.eclipse.org/egit>

2.10.1. Eclipse IDE

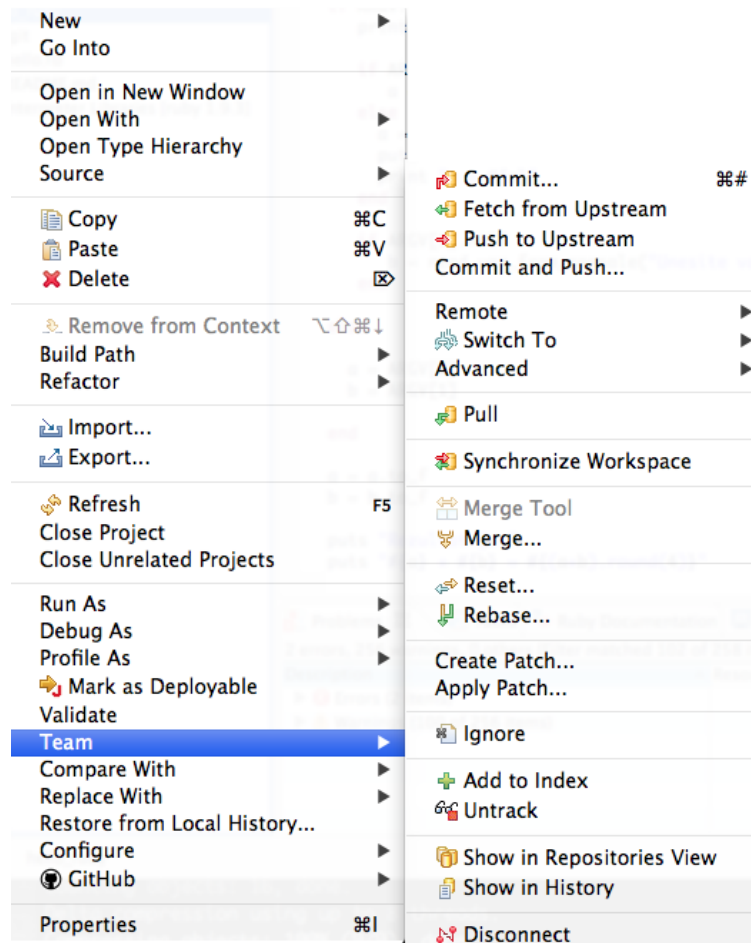
Instalacijom [EGit plugin-a](#) možemo unutar našeg IDE-a vršiti operacije nad git repozitorijom.

Untar našeg projekta podešavamo, git repozitorij (s obzirom da on već postoji):



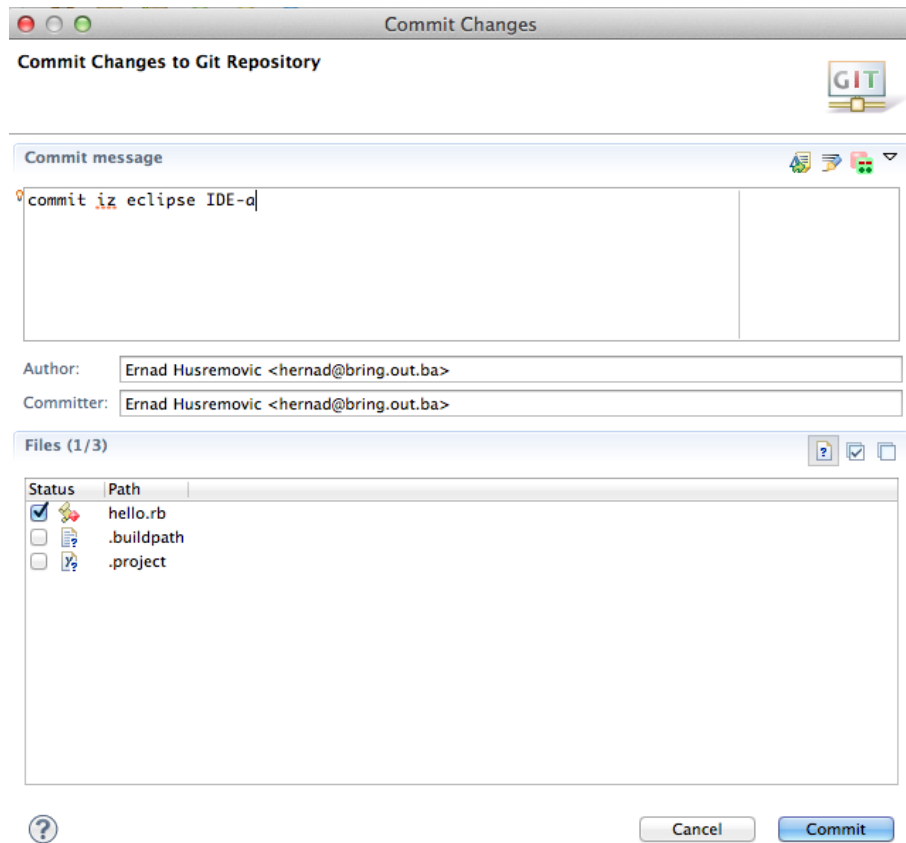
Slika 2.30: Eclipse IDE git - setup postojećeg repozitorija u IDE-u

Nakon toga su nam dostupne SCM operacije nad repozitorijom:



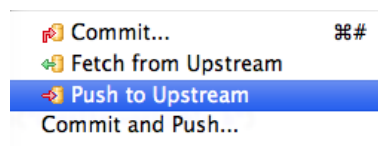
Slika 2.31: Eclipse IDE GIT - Project team support

Nakon izvršenih promjena unutar IDE-a, pokrećemo commit unutar IDE-a:

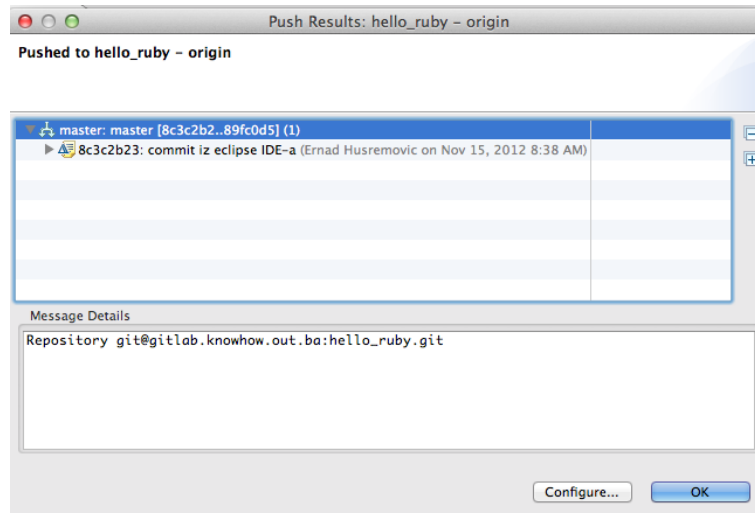


Slika 2.32: Eclipse IDE commit dijalog

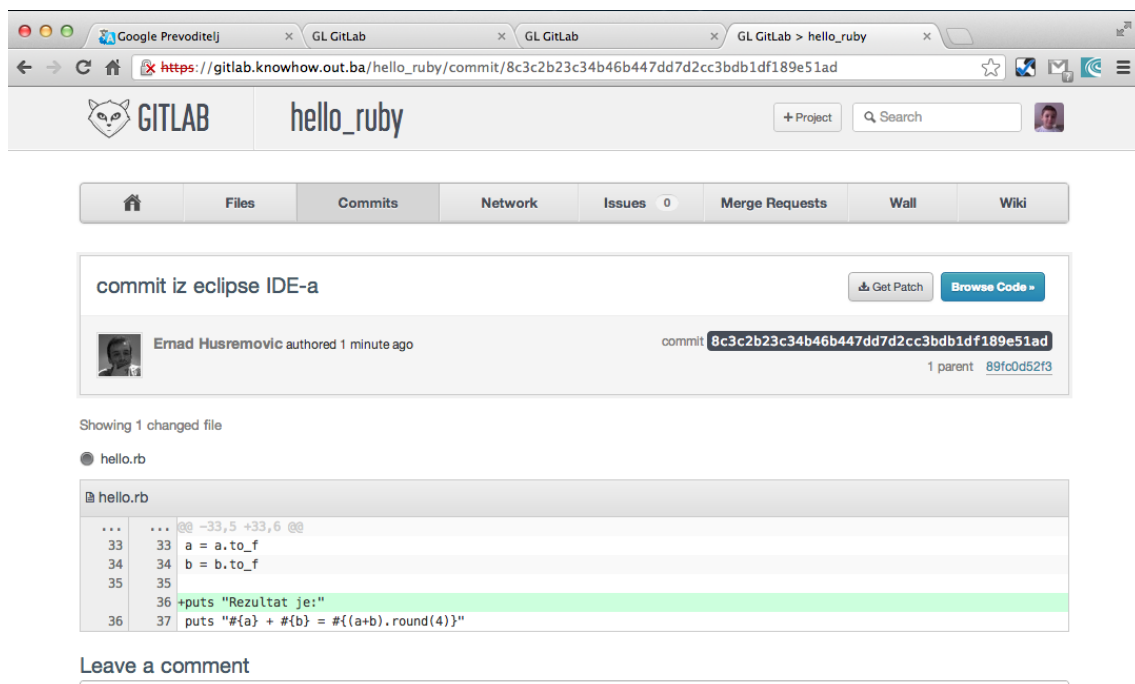
Push na remote repozitorij:



Slika 2.33: Eclipse push dijalog



Slika 2.34: Eclipse IDE, informacija o izvršenoj push operaciji



Slika 2.35: Naša promjena nalazi se na gitlab serveru

2.11. "Collective code ownership"

We are all responsible for high-quality code.

"Collective code ownership" (J. Shore i S. Warden, 2008, str.) je agilni princip po kome svako odgovoran za visoki kvalitet koda. Ako se tokom čitanja uputstva uoče

nedostaci, ako je programski kod nepregledan, pristupa se popravci. Pri tome se ne gleda ko je originalni pisac uputstva ili koda. Kod je u vlasništvu tima.

Ernad je nakon Bakirovih intervencija na uputstvu uočio da postoji još par stavki koje bi trebalo ispraviti.

Za ovu intervenciju nema nikakvog razloga praviti novi branch - ispravke će se vršiti direktno u "master"-u.

Ernad vrši update master-a na svom desktopu:

```
hernad@hello$ git checkout master
```

```
Switched to branch 'master'
Your branch is behind 'origin/master' by 6 commits, and can be ←
fast-forwarded.
```

Ernad preuzima (pull-uje) Bakir-ove promjene sa servera:

```
hernad@hello$ git pull
```

```
Updating feab5f1..1ed5b06
Fast-forward
 README.md | 34 ++++++
 hello.rb  | 38 ++++++
 2 files changed, 65 insertions(+), 7 deletions(-)
```

Vršimo ispravku uputstva u našem editoru: `hernad@hello$ vi README.md`

Commit promjena:

```
hernad@hello$ git commit -a -m "uputstvo update"
```

```
[master 74fb346] uputstvo update
1 file changed, 1 insertion(+), 1 deletion(-)
```

Označava novu verziju 0.3.2:

```
hernad@hello$ git tag 0.3.2
```

```
hernad@hello$ git push origin master -tags
```

```
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 333 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@gitlab.knowhow.out.ba:hello_ruby.git
 1ed5b06..74fb346 master -> master
* [new tag] 0.3.2 -> 0.3.2
```

Treba uočiti da je ovaj proces brz i jednostavan. Ne traži nikakve posebne dogovore između developera. Jednostavno, svako ko uoči potrebu da napravi sitnu korekciju - to i čini.

3. Zaključak

Git nam omogućava da se jedan od ključnih principa agilnog software dvelopmenta, "Collective code ownership", realizira u praksi.

Git omogućava developeru da sve svoje ideje sa minimalnim "overhead"-om pohrani u centralni repozitorij (branching).

Web servisi kao što su Github i Gitlab omogućavaju jednostavnu i efikasnu komunikaciju tima. Centralna tačka komunikacije je izvorni kod¹. Kako je izvorni kod projekta je glavni proizvod programskog tima, to je i prirodno mjesto komunikacije.

Pored mogućnosti da se direktno ispravlja programski koda, svaki član tima ima mogućnost komentara pojedinih "commit"-a.

Ta opcija omogućava iskusnijim članovima tima (senior developer, product manager, agile couch) da junior programerima brzo i jednostavno daju upute za korekcije i nadogradnju njihovog koda.

¹Ovdje se misli na izvorni kod u širem smislu. Misli se na sve digitalne artefakte projekta. Dokumentacija je sastavni dio izvornog koda

4. Literatura

J.Shore i S.Warden. *The Art of Agile Development*. O'Reilly, 2008.

Dodatak A

Riječnik pojmova

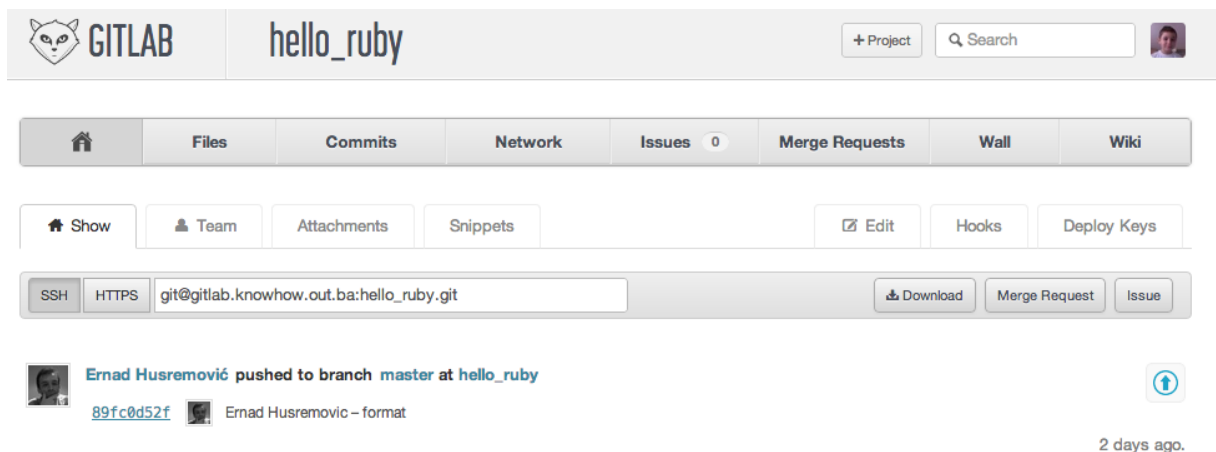
- `tag` - tag je oznaka unutar git repozitorija koja najčešće pokazuje na određenu verziju (release) ili podverziju - iteraciju (iteration)
- `branch` - branch označava poseban ogranak projekat u odnosu na glavni tok razvoja. Primjera radi, ako želimo uvesti neke eksperimentalne funkcije, otvaramo poseban "experimental" branch unutar koga implementiramo naše eksperimentalne funkcije. Te promjene se ne vide na glavnom - "master" branch-u.
- `merge request` - Proces merđžiranja (eng. merge - spojiti) je proces u kome se vrši spajanje promjena iz različiti branch-ova ili različitih repozitorija. Vezano za primjer branch-a (vidi: branch), ako su se promjene u "experimental" branch-u pokazale dobrim, "merge" operacijom ih prebacujemo - integriramo u "master" branch. Nakon uspješnog merđžiranja "experimental" branch možemo obrisati.
- `repository` - baza u promjena. Lokalni repozitorij se nalazi u .git repozitoriju našeg projekta na lokalnom disku. 'Remote' repozitorij se uobičajeno nalazi na udaljenom serveru kome pristupamo putsem ssh ili http protokola.
- `diff` - Skraćenica od "difference". Koristi se za poređenje dva različita stanja repozitorija. Tačka poređenja mogu biti "tag", "branch" ili jednostavno određeni "commit".

Dodatak B

Napomene

Materijal je prepun anglizama, što odstupa od uobičajenih akademskih standarda pisanja. Međutim, s obzirom na ciljni auditorij, mišljenja smo da bi pokušaj prevođenja na maternji jezik izazvao više konfuzije nego što bi pomogao u razumjevanju sadržaja.

Uzmimo za primjer **"commit"** operaciju. Uzevši u obzir kontekst, "commit" operacijom se promjene šalju - predaju u lokalni repozitorij. Stoga bi prevod najadekvatnije bilo koristiti riječ "predaj" na bosanskom jeziku. Koliko bi prevoda ovih termina bio neprimjeren, zorno pokazuje pokušaj prevoda opcija na sljedećem ekranu:



Slika B.1: Prevod na bosanski - nemoguća misija

Stoga se termini "hardware", "software", "commit", "push", "merge", "branch" u materijalu koriste isključivo u izvornoj formi.

Dodatak C

Software toolset

1. Mac OS X 10.8.2
2. mvim, vim tekst editor ver 7.3
3. MacTex (TeX Live 2012)

Dodatak D

Software repozitoriji

- Agilni developerski environment https://github.com/hernad/agile_dev_env