Softver otvorenog koda

Žarko Zečević
Elektrotehnički fakultet
Univerzitet Crne Gore

Predavanje 4

SOK za kontrolu verzija

Ishodi učenja:

Nakon savladavanja gradiva sa ovog predavanja studenti će moći da:

- Razumiju potrebu za korišćenjem sistema za kontrolu verziju (VCS) i usvoje osnovnu VCS terminologiju
- Naprave razliku između centralizovanog i distribuiranog sistema za kontrolu verzija
- Koriste git – distribuirani sistem za kontrolu verzija, i GitLab – web okuženje otvorenog koda za čuvanje i upravljanje remote repozitorijuma

Šta je kontrola verzije softvera?

- Kada radimo na nekom projektu često želimo da sačuvamo i backup-ujemo ono što smo do nekog trenutka odradili, da bi se osigurali da to ne izgubimo.
- Nekad želimo da isprobamo neki novi pristup/tehnologiju u projektu, ali za svaki slučaj želimo da sačuvamo prethodnu verziju.
- Takođe nekad prosto želimo da napravimo backup projekta da bi se osigurali u slučaju da dođe do oštećenja diska.
- Na kraju, ako radimo u timu, potrebno je na neki način distribuirati promjene ostalim članovima tima.

Šta je kontrola verzije softvera?

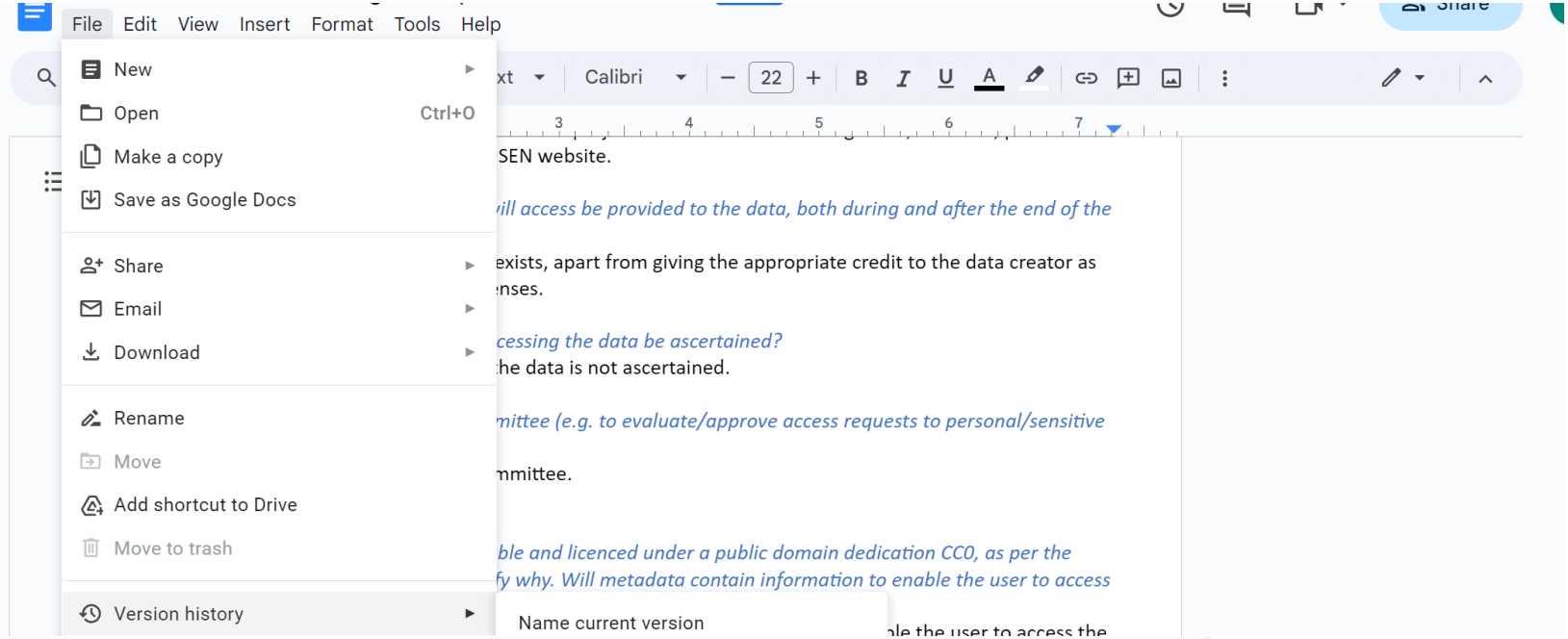
Sistem za kontrolu verzija (eng. VCS) je alat koji omogućava praćenje promjena u fajlovima. VCS treba da omogući uvid u:

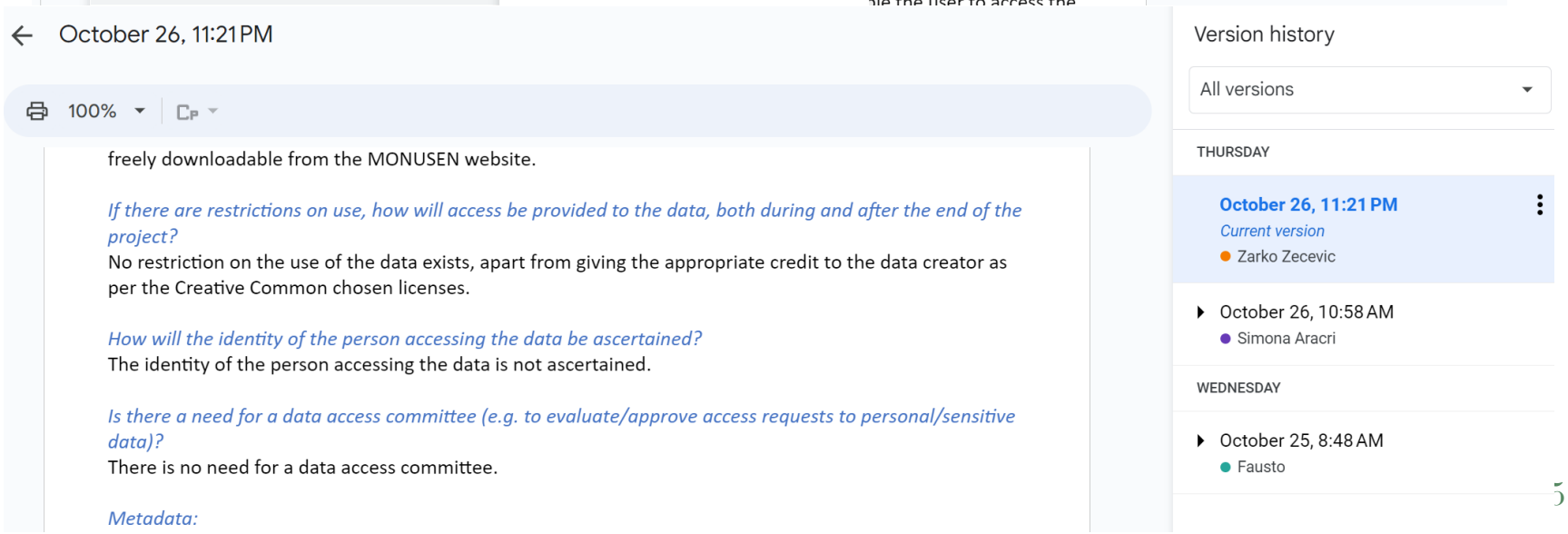
- Ko je napravio promjenu?
- Kada je promjena napravljena?
- Šta je tom prilikom promijenjeno i na koji način?

VCS može biti:

- Manuelan (ručno pravimo kopije projekta i dajemo odgovarajuće nazive verzijama, a zatim ih šerujemo članovima tima)
- Automatski (softver je zadužen za praćenje promjena, kreiranje verzija i distribuciju fajlova članovima tima)

Google Docs


File Edit View Insert Format Tools Help
New
Open Ctrl+O
Make a copy
Save as Google Docs
Share
Email
Download
Rename
Move
Add shortcut to Drive
Move to trash
Version history
Name current version
Calibri
22
SEN website.
ill access be provided to the data, both during and after the end of the
exists, apart from giving the appropriate credit to the data creator as
nses.
cessing the data be ascertained?
he data is not ascertained.
mittee (e.g. to evaluate/approve access requests to personal/sensitive
mmittee.
ble and licenced under a public domain dedication CC0, as per the
fy why. Will metadata contain information to enable the user to access


October 26, 11:21 PM
100%
freely downloadable from the MONUSEN website.
If there are restrictions on use, how will access be provided to the data, both during and after the end of the project?
No restriction on the use of the data exists, apart from giving the appropriate credit to the data creator as per the Creative Common chosen licenses.
How will the identity of the person accessing the data be ascertained?
The identity of the person accessing the data is not ascertained.
Is there a need for a data access committee (e.g. to evaluate/approve access requests to personal/sensitive data)?
There is no need for a data access committee.
Metadata:
Version history
All versions
THURSDAY
October 26, 11:21 PM
Current version
Zarko Zecevic
October 26, 10:58 AM
Simona Aracri
WEDNESDAY
October 25, 8:48 AM
Fausto

Google Docs

Sisteme za kontrolu verzija svakodnevno koristimo. Na primjer, Google Docs automatski prati promjene u dokumentima korišćenjem elementarnog VCS

Prednosti:

- Odljičan za *rich* tekst
- Omogućava kolaboraciju u realnom vremenu
- Čuva se na cloud-u automatski

Nedostaci:

- Nije dobar za *plain* tekst
- Zahtijeva internet konekciju
- Podržava samo jednu „trenutnu“ verziju jednog fajla

Benefiti korišćenja VCS

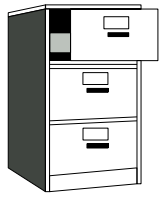
- Prednosti za one koji samostalno rade
  - Backup-ovi sa mogućnošću praćenja promjena
  - Tagovanje – označavanje određene verzije u vremenu
  - Grananje – nezavisno razvijanje više verzija
  - Praćanje promjena
  - Vraćanje promjena (undo)
- Prednosti za timove
  - Rad na istom kodu u timu
  - Spajanje konkurentnih verzija
  - Podrška za rješavanje konflikta kada se isti fajl simulatno mijenja od strane više developera
  - Poznato je koji autor je napravio promjene i kada

Terminologija – VCS vokabular

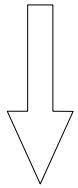
- Repozitorijum – baza u kojoj se čuvaju fajlovi
- Local copy – radna, lokalna verzija korisnika
- Trunk – Primarna lokacija za fajlove u projektu
- Branch – sekundarna lokacija za fajlove
- Revision – verzija fajla ili repozitorijuma
- Commit – čuvanje promjena u repozitorijumu
- Revert – vraćanje promjena
- Merge – spajanje promjena iz jedne grane u drugu
- Conflict – nastaje kada se mergovanje ne može automatski odraditi

Checkout – kreiranje lokalne kopije

- Pojam checkout se odnosi na kreiranje lokalne kopije fajlova iz repozitorijuma
- Direktorijum u kom se kreira lokalna kopija obično sadrži dodatne fajlove sa informacijama o verziji fajla, revizijama i samom repozitorijumu (primer .svn i .git)
- Modifikovanjem fajlova zapravo modifikujemo lokalnu kopiju partikularne verzije fajlova



Repozitorijum

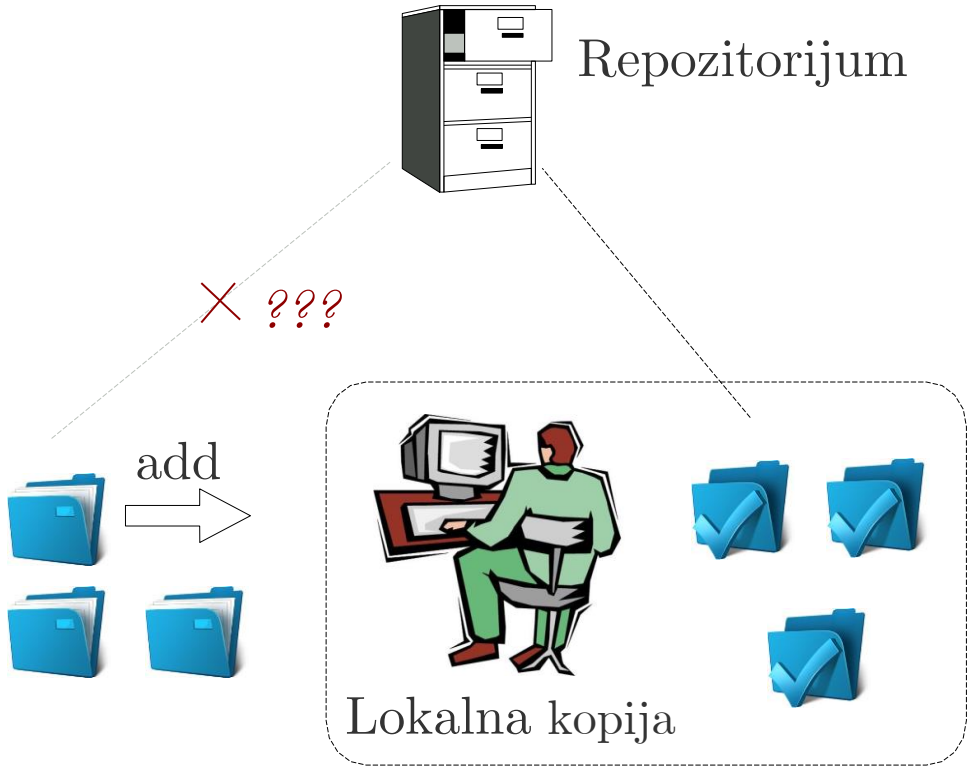


checkout

Lokalna kopija

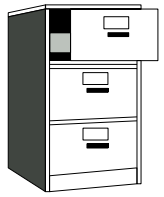
Add – dodavanje novih fajlova

- Potrebno je informisati VSN o dodavanju novih datoteka kako bi on pratio promjene u njima
- Bez ekplicitnog dodavanja datoteke VSN ne bi znao koje fajlove da prati, a koje ne

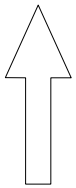


Commit – čuvanje promjena u repo

- Šalje se zahtjev da se lokalna modifikacija fajlova prihvati kao nova verzija u repozitorijumu.
- VSN kreira novu verziju i dodjeljuje joj broj revizije za jedan veći u odnosu na najnoviju verziju
- U slučaju da u repozitorijumu nema novije revizije (u odnosu na lokalno izmijenjenu kopiju), promjene se čuvaju u repo-u. U suprotnom:
  - Potrebno je ažurirati lokalnu verziju
  - Ako se spajanje fajlova ne izvrši automatski, potrebno je ručno riješiti konflikte



Repozitorijum

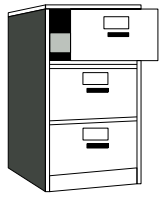


commit

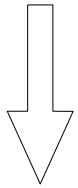
Lokalna kopija

Update – ažuiranje lokalne kopije

- Pojam update se odnosi na ažuriranje lokalne kopija repo-a na najnoviju ili specificiranu verziju iz repo-a
- Ako su promjene kompatabilne sa lokalnim modifikacijama onda se vrši automatsko mergovanje
- U suprotnom potrebno je ručno riješiti konflikte i prihvatiti odgovarajuću verziju



Repozitorijum



update

Lokalna kopija

Rješavanje konflikata

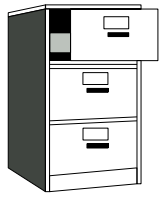
- VSN ne sprječava nastajanje konflikata, ali obezbjeđuje alate kojim se konflikti rješavaju
- Konflikti obično nastaju kada više autora simultano mijenja jedan isti fajl
- Konflikti se mogu izbjeći odabirom odgovarajuće strukture izvornih fajlova, korištenjem modula i dobrom organizacijom projektnih fajlova i timskim radom
- Konflikti se mogu dalje izbjeći dodjeljivanjem prava pristupa određenim datotekama, odnosno programerima (autorizacija)

Primjer konflikta

```
<hmtl>
    <body>
        <h1> HTML stranica <h1>
        <h1> Primjer konflikta <h1>
<<<<<<< HEAD
             Dodate dvije linije koda
             u lokalnom repo-u
=======
            Dodate dvije linije koda
            na serveru
>>>>>>> 0083ec67f406bf238fedfa1846310925ceea91f2
    </body>
</hmtl>
```

Tagovanje – označavanje verzije

- VCS čuva informacije o istoriji promjena fajlova
- Možemo označiti paritukularnu verziju u repozitoriju korišćenjem taga, npr. Release 1.0
- Tag – simboličko ime za određenu verziju (stanje) u repozitorijumu
- HEAD tag se obično koristi za označavanje tekuće verzije u repozitorijumu



Repozitorijum

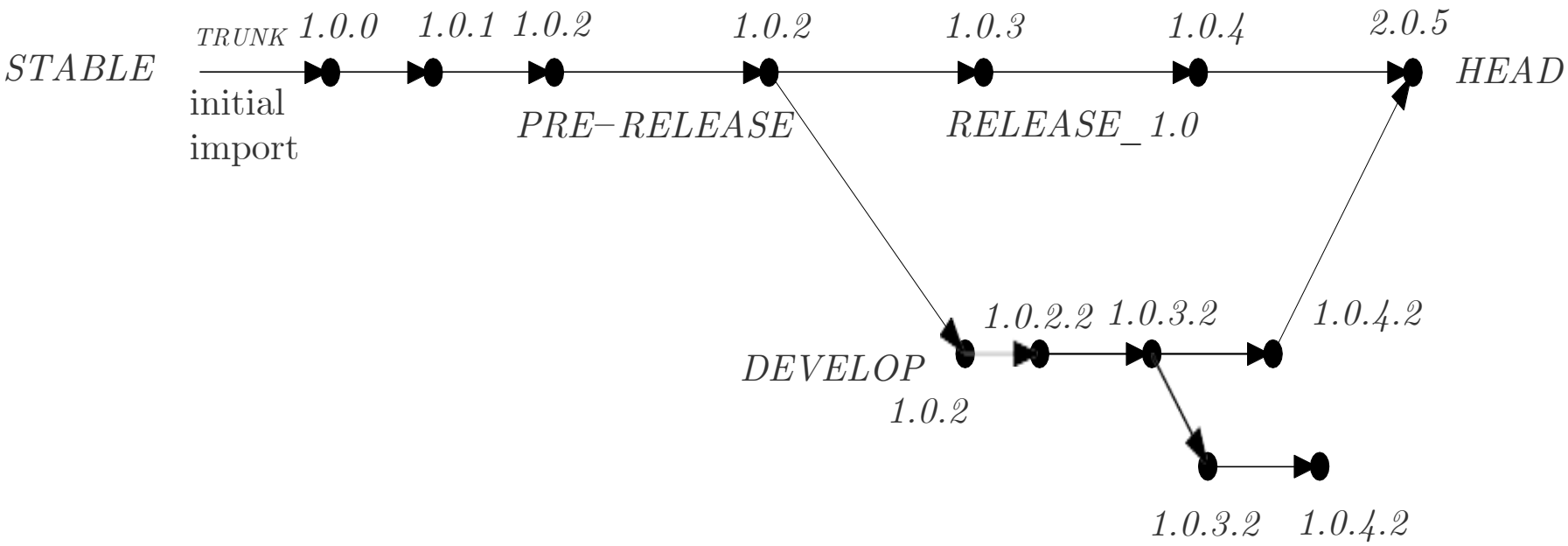
update

Lokalna kopija

Grananje i nazivi grana

- Omogućava paralelni rad na različitim idejama / tokovima / implementacijama. Napr. kada želimo:
  - Postepeno da ažuriramo tehnologije koje se koriste u projektu
  - Da testiramo ili evaluiramo nove dodatke prije nego što ih uključimo u glavni kod
- Postoje standardni nazivi za grane:
  - CURRENT, TRUNK – glavna razvojna grana
  - STABLE – stabilna verzija
- Korišćenje mnogo grana u razvoju, te spajanje grana. predstavljaju najbitnije karakteristike VSN-a

Primjer grana



Centralizovani i distribuirani VCS

Centralilizovani

- Jedan centralni repozitorijum. Klijenti imaju samo tekuću verziju
- Precizno definisan source repozitorijuma (jednostavna autorizacija, jedna tačka otkaza)
- Različite verzije se obično označavaju sekvencijalnim brojevima (jednostavno za pamćenje)
- Ako se respozitorijum nalazi na serveru, onda može biti zahtjevana mrežna konekcija za rad
- Generalno manje koriste grananje za eksperimentisanje

Distribuirani

- Svaki korisnik ima kopiju repozitorijuma (komplikovanije za autorizaciju, više redudantnih kopija, može zahtijevati više prostora za smještanje velikih repozitorijuma)
- Moguće je raditi offline (kreira se lokalna kopija repozitorijuma)
- Identifikatori verzija su Globalni ID-ovi
- Više grana i šerovanja

Primjeri VCS-a

- Nekad se zovu i Source Code Manager-i (SCM)
- Postoji veliki broj VCS sistema koji su open source i vlasnički softver:

https://en.wikipedia.org/wiki/List_of_version_control_software

Lokalni: SCCS (1972), RCS (1982), PVCS1 (1985), QVCS1 (1991)

Klijent-sever: CVS (1986), ClearCase1 (1992), Perforce1 (1995), Subversion (2000), Surround SCM1 (2002), Visual Studio Team Services1 (2014)

Distribuirani: BitKeeper, Darcs, SVK, Bazaar, Mercurial, Git, Plastic SCM1, Visual Studio Team Services

- Free/open-source – Subversion, Git
- Vlasnički – Surround SCM, Plastic SCM

Dobro je znati koji sve sistemi postoje, koje su prednosti jednih u odnosu na druge kako bi odabrali koji vama najviše odgovara:

https://en.wikipedia.org/wiki/Comparison_of_version_control_software

Subversion vs Git

Subversion

- Centralni repozitorijum – glavni repozitorijum je jedini source i jedino u njemu se čuva kompletna istorija fajla
- Korisnici mogu da povuku jedino kopiju tekuće verzije
- Autorizacija se može vršiti na nivou pojedinih direktorijuma
- ID revizije se odnosi na čitav repozitorijum
- Tagovi i grane su zapravo direktorijumi

GIT

- Distribuirani repozitorijum – svaka lokalna kopija predstavlja kompletan repozitorijum sa kompletnom istorijom promjena
- Veća redudantnost i veća brzina
- Grananje i mergovanje se češće koristi
- Grane i tagovi su zapravo markeri za „podskupove“ repo-a

Karakterisike GIT-a

- Dozvoljeno je kreiranje verzija na lokalnom nivou, bez internet konekcije
- Centralni repozitorijum „vodi“ odgovorni programer
- Commit dodaje promjene samo u lokalni repozitorijum, a naknadno je potrebno te promjene uploadovati na server
- Efikasan za veće projekte
- Jednostavan korisnički interfejs
- Tagovi i grane predstavljaju markere za stanja u repozitorijumu
- Pogodnost implementacije Git-a zavisi od projekta i modela razvoja

Konfiguracija korisničkih podataka

- Prije nego što se počne rad sa gitom potrebno je unijeti osnovne informacije o autoru:

```
git config --global user.name "John Smith"
git config --global user.email jsmith@seas.upenn.edu
```

- Ovo je potrebno odraditi samo jednom, jer se informacije o svakom korisniku čuvaju u fajlu `~/.gitconfig`
- Ukoliko za neki određeni projekat želite da koristite druge korisničke podatke, tada se prvo treba pozicionirati u direktorijum u kom se nalazi projekat, a zatim treba koristiti gornje komande, ali bez opcije **global**.

Kreiranje repozitorijuma

- Repozitorijum se kreira tako što se prvo pozicionirate u direktorijum u kojem će biti smješten projekat, a zatim se kuca komanda:

```
git init
```

- Ovom komandom se krerira direktorijum .git u kom se čuvaju sve informacije o istoriji promjena fajlova
- Svaki fajl se mora dodati u repozitorijum kako bi sistem vršio njegovo praćenje:

```
git add .   # (dodaje se svi fajlovi iz dir .)
```

- Svaka promjena fajla se čuva u „radnom prostoru“, a promjene se moraju naknando sačuvati u repozitorijum:

```
git commit –m "Poruka o napravljenim izmjenama"
```

Rad sa verzijama

- Stanje radnog direktorijuma možete provjeriti komandom:

  ```
  git status
  ```

  Ova komanda pruža informacije o tome da li postoje nepotvrđene promjene fajlova.
- Svakom commit-u, odnosno update-u lokalnog repozitorijuma se dodjeljuje jedinistveni hash string. Spisak commit-a i hash stringova se može dobiti komandom:

  ```
  git log
  ```
- Ukoliko želite da poništite promjene radne verzije i u radni prostor učitate zadnju sačuvanu (commit-ovanu) verziju, tada koristite komandu:

  ```
  git checkout naziv_fajla
  ```

Rad sa verzijama

- Ukoliko pak želite da učitate neku stariju veziju fajla, tada se koristi komanda

```
git checkout hash_string naziv_fajla
```

- Commit-i se ne moraju raditi nakon svake izmjene, već se najčešće rade izmjene više fajlova, nakon čega se izmjene čuvaju. Ukoliko želite da učitate staro stanje svih fajlova, tada se koristi gornja komanda, ali se ne navodi ime fajla

```
git checkout hash_string
```

- Konačno, ukoliko projekat ima više razvojnih grana, prelazak iz jednu u drugu granu se takođe vrši pomoću komande checkout:

```
git checkout naziv_grane
```

Git primjer

```
mkdir ~/public_html/git
git init # incijilazacija repo-a
echo Prva linija > index.html
git status # provjera statusa
git add index.html # dodavanje fajla
git status
git commit -m "dodata prva linija“ # komitovanje
git status
echo Druga linija >> index.html
git status
git commit -am "dodata druga linija“
# mora se koristiti opcija a da bi se poromjene mogle sacuvati
# u suprotnom prije commit-a treba odraditi git add index.html
echo Treca linija >> index.html
git commit -am "dodata treca linija“
git log
git checkout 65afc301712968e795e52de240178f72dc074b7a
```

Ostale GIT komande

`git commit -am "poruka"` – čuvanje promjene u repo

Svaka izmjena fajla, čak iako fajl postoji u repo-u, se mora registrovati u repozitorijum pomoću komande git add. Dodavanjem opcije –a ovaj korak se automatizuje, tj. nije potrebno korisitiiti git add komandu prije git commit.

`git help command` – help na određenu git komandu

Za brisanje i premještanje fajlova ne preporučuje se korišćenje standardnih Linux komandi, već treba koristiti specijalne git komande koje između ostalog rade commit i informišu repozitorijum o promjenama:

`git rm` – brisanje fajla iz repo-a

`git mv` – preimenovanje fajla iz repo-a

Podešavanje remote repozitorijuma

`git clone URL_do_git_repozitorijuma` – kreiranje lokalne kopije remote repozitorijuma

Dvije varijante za `URL_do_git_repozitorijuma`

– `http://89.188.32.141/zarkoz/lab4.git`

– `git@89.188.32.141:zarkoz/lab4.git`

U prvoj varijanti je potrebno unijeti username i password ukoliko repozitorijum nije javan. Da bi se koristila druga varijanta, potrebno je kreirati ssh ključ i kopirati ga na git server. Nakon podešavanja ssh ključa, server više neće zahtijevati unošenje korisničkih podataka.

Uputstvo za kreiranje ssh ključa može naći na sljedećem linku:

https://docs.gitlab.com/ee/user/ssh.html#add-an-ssh-key-to-your-gitlab-account

Podešavanje remote repozitorijuma

Informacije o repozitorijumu se čuvaju u folderu .git koji nalazi u glavnom folderu repozitorijuma. U ovom folderu se nalazi i konfiguracioni fajl .git/config. Primjer sadržaja config fajla je dat ispod.

```
[remote "origin"]
        url = git@89.188.32.141:zarkoz/lab4.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
```

Gornje linije znače da je remote repo-u lab4 dodijeljeno ime origin. Prilikom korišćenja komande fetch biće download-ovane sve grane sa remote repo-a, dok prilikom korišćenja komandi pull ili push biće mergovane lokalna grana master i remote grana master.

Podešavanje remote repozitorijuma

Ukoliko smo repozitorijum kreirali lokalno, a ne pomoću komande git clone, da bi uploadovali fajlove na udaljeni server, najprije je potrebno dodati informaciju o remote repo-u:

```
git remote add origin http://89.188.32.141/zarkoz/git.git
```

Ova komanda zapravo dodaje informacije o remote repo-u u .git/config fajl. Naziv origin nije obavezan, a samim tim je istom komandom moguće sa lokalnim repo-om povezati više različitih remote repo-a (pod različitim imenima).

URL remote repo-a se može promijeniti na sljedeći način

```
git remote set-url origin
http://89.188.32.141/zarkoz/GITT.git
```

dok se naziv remote grane mijenja pomoću komande

```
git remote rename origin novi_naziv
```

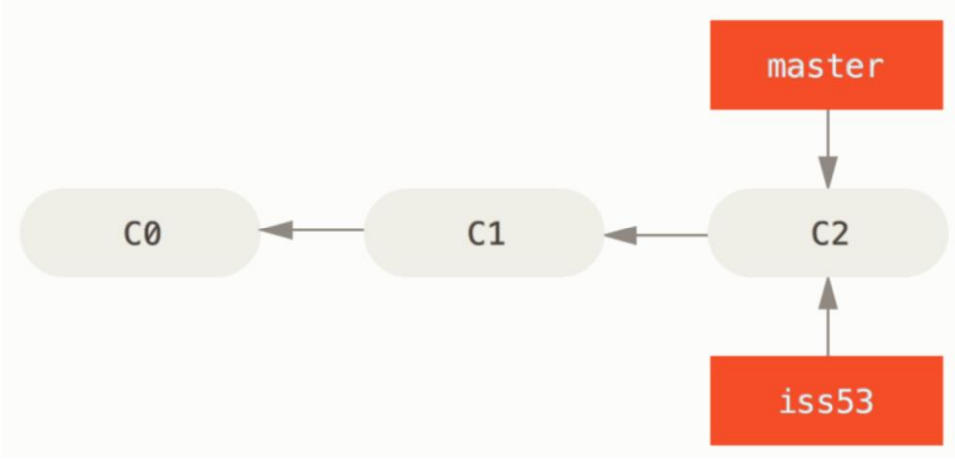
Kreiranje i rad sa granama

Grananje je jedan od ključnih koncepata u distribuiranim VCS. Nova grana, koja se nadovezuje na tekuću, se kreira pomoću komande:

```
git branch naziv_grane
```

Spisak svih grana se može dobiti pomoću komande `git branch`, dok u drugu granu se prelazi pomoću komande

```
git checkout naziv_grane
```



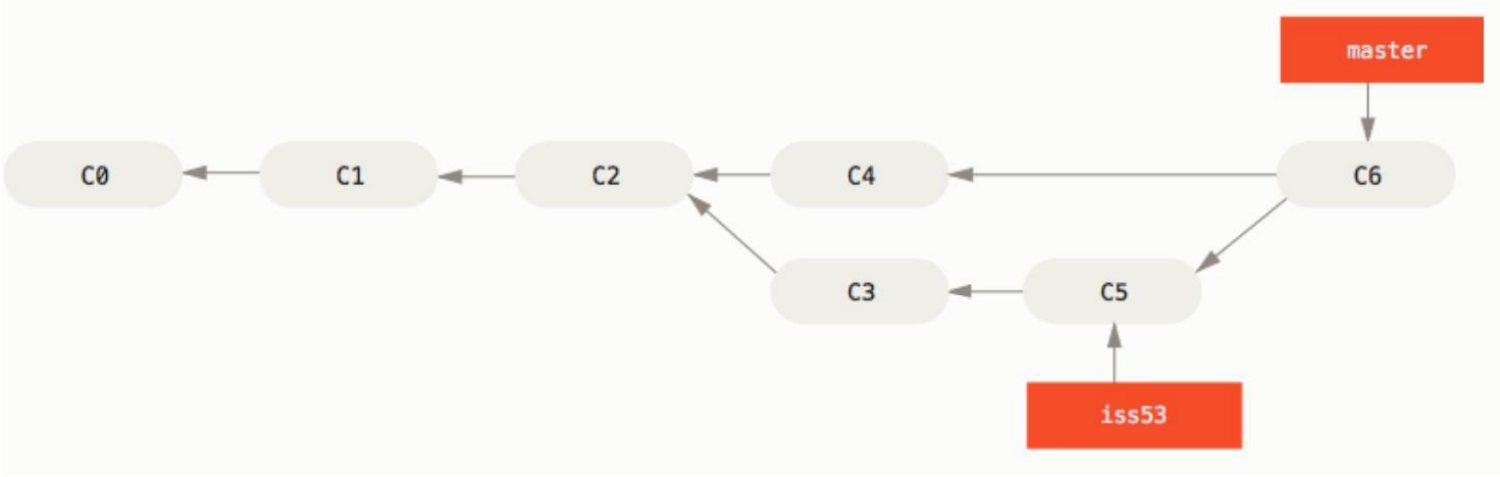
Kreiranje i rad sa granama

Nakon nekog vremena sporednu granu treba spojiti (mergovati) sa glavnom granom, i to se radi pomoću komande:

```
git merge naziv_sporedne_grane
```

Prije izvršavanja gornje komande, potrebno je pozicionirati se u granu koju želimo da updejtujemo (npr. master). Prilikom spajanja grana mogu nastati konflikti. Razlike između dvije grane se mogu prikazati pomoću komande

```
git diff grana1 grana2
```



Download/upload sa remote repo-a

Komande pull i fetch se koriste za download grana sa remote repo-a (ažuriranje lokalnog repo-a).

`git pull` – download grana sa remote repo-a i merge-ovanje sa lokalnim granama (može doći do konflikata koje treba razriješiti). Updejtuju se one grane koje su specificirane u .git/config fajlu.

Po default-u, git clone komanda klonira samo master granu, pa ako želimo da downloadujemo novu granu sa repo-a (npr. develop), moramo je prvo kreirati:

```
git branch -b develop
```

Nakon toga moramo reći lokalnim repo-u da želimo da pratimo tu granu

`git branch -u origin/develop develop` (ovo će promijeniti config fajl),

i tek nakon toga možemo izvršiti git pull komandu da povučemo sadržaj grane.

Download/upload na remote repo

Ako želimo da updejtujemo samo određene grane, tada se koristi sljedeća varijanta komande

```
git pull origin master
```

gdje origin označava naziv remote repo-a, a master granu koju želimo da updejtujemo.

Komanda `git fetch` takođe vrši download sa remote repo-a, ali se ništa ne ažurira dok se ne napravi uvid u razlike (manje agresivna od git pull). Ova komanda downloaduje čitav repozitorijum (pogleda config fajl). Da bi se promjene sačuvale, potrebno je izvršiti git merge komandu, pri čemu prije toga treba kreirati lokalnu granu, ukoliko ona ne postoji.

Slično kao i kod pull komande, varijanta ispod samo „fetchuje“ specificiranu granu.

```
git fetch origin master
```

Download/upload na remote repo

Za upload lokalnog repo-a na remote repo koristi se komanda

```
git push naziv_repoa naziv_grane
```

Ukoliko se samo otkuca komanda

```
git push
```

onda će se master grana uploadovati u origin/master, jer je tako specificirano u .git/config fajlu.

Varijante komande

```
git push -u origin develop
```

ili

```
git push --set-upstream origin develop
```

će uploadovati develop u origin/develop, ali će i izmijeniti config fajl.

Za šta sve možemo koristiti VCS?

- Čuvanje source kodova programa
- Verzionisanje programa
- Verzionisanje dokumenata
  - Mogu se verzionisati i tekstualni i binarni dokumenti. Doduše promjene se teže prate kod binarnih dokumenata
- VCS se može koristiti za šerovanje fajlova, a i za rad na zajedničkom dokumentu (na primjer za seminarski rad)
- VCS se može koristiti kao alat za kreiranje i čuvanje rezervnih kopija

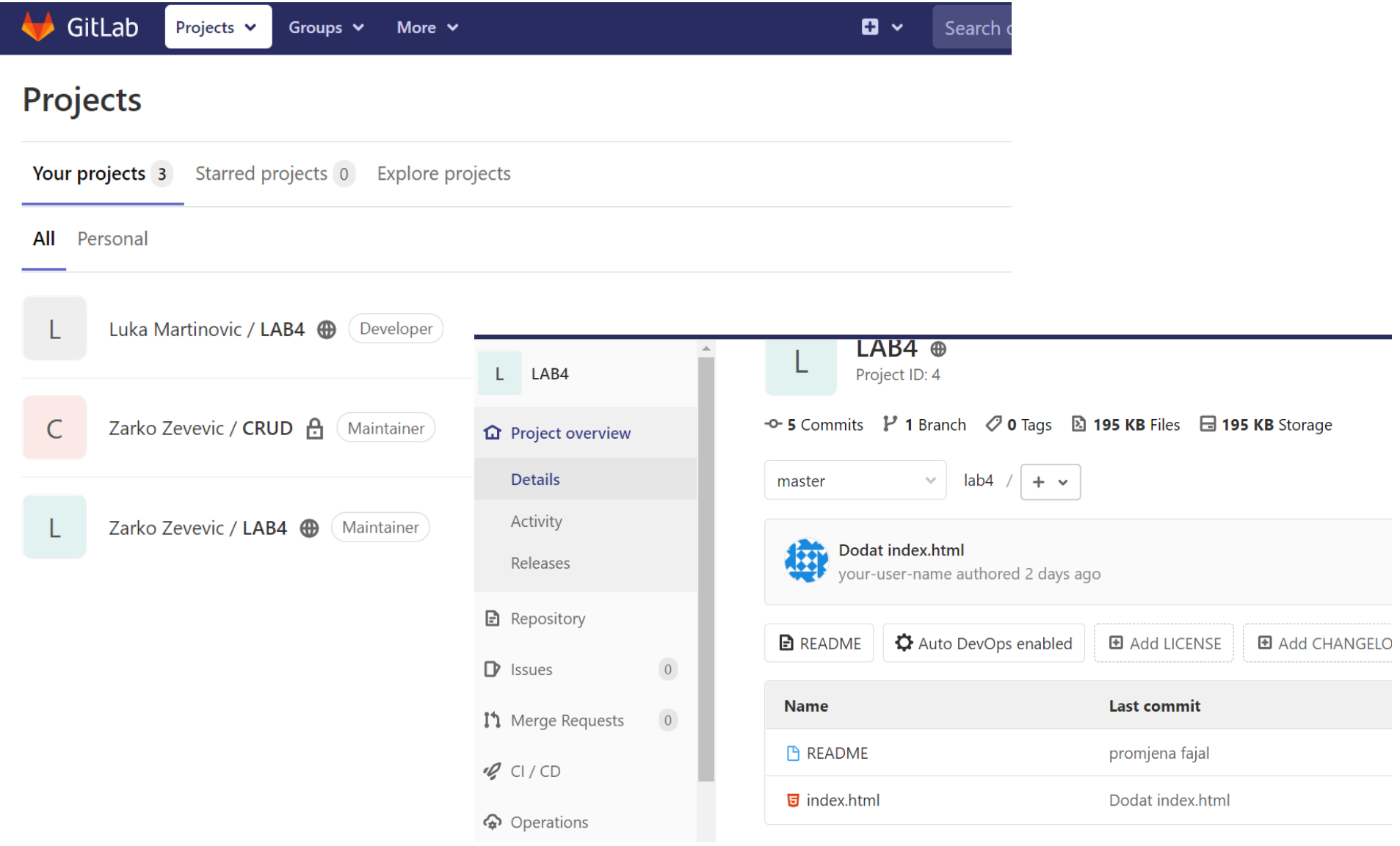
GitLab

- GitLab – open source web-based manager git repozitorijuma

  http://89.188.32.141/vas_user/lab4

- Svaki student ima svoj nalog i prostor za smještanje repozitorijuma
- Repozitorijumi mogu da se šeruju sa drugim studentima
- Članovima tima se mogu dodijeliti odgovarajuće uloge
- Repozitorijumi mogu da se download-uju na Linux server (lokalna kopija), dok se na serveru (GitLab) nalazi glavna verzija.

GitLab


GitLab
Projects
Groups
More
Search
Projects
Your projects 3
Starred projects 0
Explore projects
All
Personal
L
Luka Martinovic / LAB4
Developer
C
Zarko Zevevic / CRUD
Maintainer
L
Zarko Zevevic / LAB4
Maintainer
L
LAB4
Project overview
Details
Activity
Releases
Repository
Issues 0
Merge Requests 0
CI / CD
Operations
L
LAB4
Project ID: 4
5 Commits
1 Branch
0 Tags
195 KB Files
195 KB Storage
master
lab4 /
+
Dodat index.html
your-user-name authored 2 days ago
README
Auto DevOps enabled
Add LICENSE
Add CHANGELO
Name
Last commit
README
promjena fajal
index.html
Dodat index.html

GitLab CI/CD

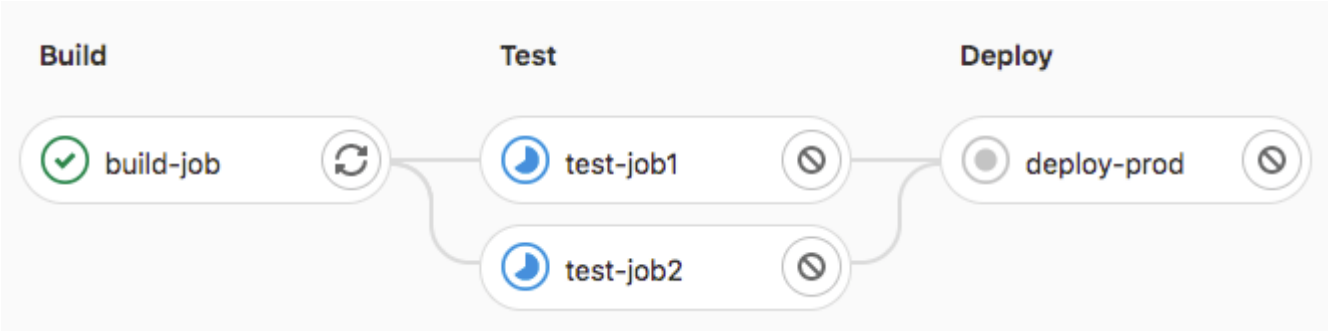
- CI/CD je skraćenica koja se odnosi na kontinuiranu integraciju (CI) i kontinuiranu isporuku/raspoređivanje (CD) softvera.
- CI (continuous integration) je razvojna praksa koja podrazumijeva čestu integraciju koda i automatsko testiranje svih izmjena na projektu. Glavni cilj CI procesa je detekcija grešaka u raznim fazama razvoja projekta.
- CD (continuous delivery) podrazumijeva brzu i kontinuiranu isporuku softvera, bez internih kašnjenja. Nakon što CI završi testiranje, CD putem automatizovanih procedura postavlja novu verziju softvera u testno okruženje.
- CD (continuous deployment) se odnosi na finalnu isporuku softvera korisnicima (u proizvodno okruženje). Ovaj korak može biti manuelan ili takođe automatizovan.

GitLab CI/CD

- GitLab CI/CD je uveden 2015 godine. Za privatne repozitorijume GitLab nudi 400 besplatnih minuta mjesečno. Korisnici mogu besplatno da povežu i koriste sopstveni GitLab runner, bez ograničenja.
- GitHub je ovu mogućnost uveo 2017 godine. Za privatne repozitorijume i besplatne naloge GitHub nudi 2000 besplatnih minuta na Linux mašinama (ili 1000 minuta na Windows-u ili 200 na macOS-u)
- Automatizovane procedure za testiranje, integraciju i isporuku softvera se nazivaju **tokovi** (**pipelines**).
- Tokovi se sastoje od više **poslova** (**jobs**) koji se izvršavaju u **fazama** (**stages**).

GitLab CI/CD - Jobs

- **Poslovi (jobs)** definišu šta želimo da odradimo u našem **toku**.
  - Izvršavaju ih takozvani **runner**-i
  - Izvršavaju se u fazama/etapama
- **Faze** definišu kada i na koji način se pokreću poslovi
  - Npr, faza u kojoj se testira softver se završava nakon faze u kojoj se kod kompajlira (build-uje)
- Poslovi u svakoj fazi se obavljaju **paralelno**
  - Ukoliko se svi poslovi u nekoj fazi uspješno završeni, prelazi se na sljedeću fazu
  - Ako jedan posao u fazi ne uspije, sljedeća faza se (obično) ne izvršava



Kako se pokreće CD/CI pipeline?

- Svaka izmjena koda + commit u repozitorijumu automatski pokreće pipeline
- Upload (push) koda u GitLab repozitorijum
- Pipeline se može ručno pokrenuti preko GitLab UI-a
- Moguće je „zakazati“ izvršenje pipeline-a u određeno doba dana/nedjelje
- Moguće je definisati pravila za automatsko pokretanje pipeline-a (npr. kada se kreira tag)
- Pipeline se može pokrenuti i putem API-ja

```
curl --request POST \ --form token=<token> \ --form
ref=<ref_name> \
"http://89.188.32.141/api/v4/projects/<project_id>/t
rigger/pipeline"
```

<token> – kreira se u podešavanjima
<ref_name> - naziv grane

.gitlab-ci.yml fajl

Da bi koristili GitLab CI/CD potrebno je najprije da u root direktorijumu projekta kreirate **.gitlab-ci.yml** fajl. Ovaj fajl prati YAML sintaksu i između ostalog u njemu je moguće definisati

- Zadatke/faze koje želite da se izvršavaju, na primjer testiranje i deploy aplikacije
- Druge konfiguracione fajlove i šablone koje želite da uključite
- Biblioteke koje treba instalirati.
- Komande koje želite da izvršite sekvencijalno ili paralelno
- Lokaciju na koju vršite deploy aplikacije
- Da li želite da se se skripta pokreće automatski ili ručno, ili pak da postoji neki okidač

Primjer .gitlab-ci.yml fajla

```
build-job:
  stage: build #build faza
  script:
    - echo "Hello, $GITLAB_USER_LOGIN! #varijabla koja vraća
test-job1:                              #vaš user name
  stage: test #test faza
  script:
    - echo "This job tests something" #skripta se izvršava na
test-job2:                              #ranner-u
  stage: test
  script:
    - echo "This job tests something"
    - sleep 20                        #poslovi u istoj fazi se
deploy-prod:                          #izvršavaju paralelno
  stage: deploy #deploy faza
  script:                    #varijabla koja vraća naziv grane
    - echo "Deploys something from $CI_COMMIT_BRANCH branch."
  environment: production
```

Primjer .gitlab-ci.yml fajla

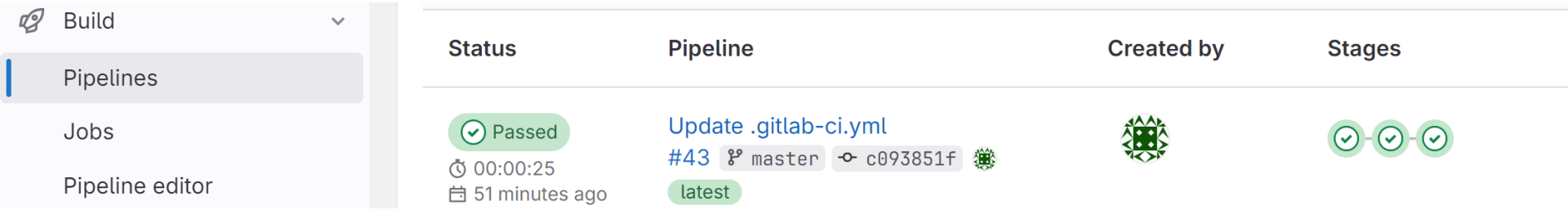
```
stages:
  - build
  - test

build_nodejs_app:
  stage: build
  image: node:lts-alpine     #definišemo sliku kontejnera
  script:                    #koji treba pokrenuti
    - npm install            # preuzimanje bilioteka
    - npm run build
  artifacts:
    paths:
      - dist/

test_nodejs_app:
  stage: test
  image: node:lts-alpine
  script:
    - npm install
    - npm run test
```

Primjer .gitlab-ci.yml fajla

```
build-job:
  stage: build
  script:
    - echo "Hello, $GITLAB_USER_LOGIN!"
test-job1:
  stage: test
  script:
    - pwd
    - tidy -o output.html --doctype omit --clean yes --tidy-mark no
index.html
  artifacts:
    paths:
      - output.html
deploy-prod:
  stage: deploy
  script:
    - usr=zarkoz
    - echo "This job tests something"
    - mkdir -p /home/$usr/public_html/git
    - cp -r ./* /home/$usr/public_html/git
```

Provjera statusa pipeline-a i poslova

Spisku pokrenutih pipeline-ova za dati projekat pristupamo klikom na **Build** > **Pipelines**

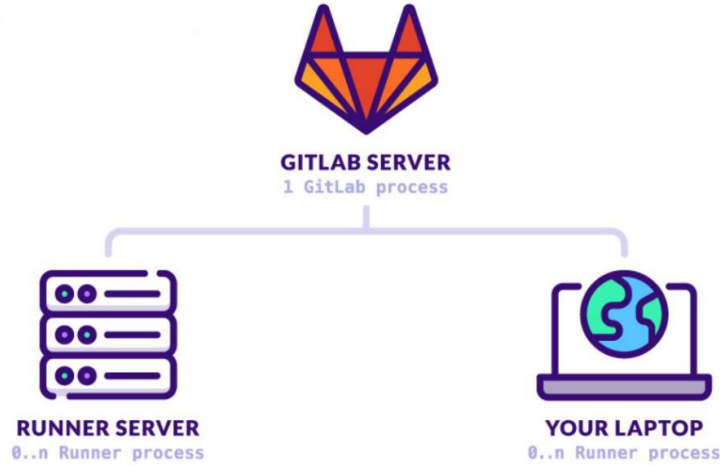


Moguće je provjeriti kako se svaki job izvršava na gitlab runner-u

```
Preparing the "shell" executor                                    00:00
Using Shell (bash) executor...
Preparing environment                                             00:00
Running on sok...
Getting source from Git repository                                00:01
Fetching changes with git depth set to 20...
Reinitialized existing Git repository in /home/zarkoz/builds/mzhm6TBbj/0/za
rkoz/fork/.git/
Checking out c093851f as detached HEAD (ref is master)...
Skipping Git submodules setup
Executing "step_script" stage of the job script                   00:00
$ echo "Hello, $GITLAB_USER_LOGIN!"
Hello, zarkoz!
Job succeeded
```

GitLab CI/CD runneri

- GitLab runner je agent (program) koji izvršava CI/CD pipeline. GitLab i GitHab imaju svoje runner-e koji su dostupni korisnicima (uz određena ograničenja).
- Svaki korisnik takođe može da registruje i instalira runner na svom/iznajmljenom serveru. Runner može biti instaliran na većini platformi (Linux, Windows, MacOS, Cloud provajder, itd.)
- GitLab runner može da testira bilo koji programski jezik (.Net, Java, Python, C, PHP)



GitLab CI/CD runneri

Runner-i mogu biti:

- **Zajednički** (mogu ih koristiti svi projekti i kreira ih GitLab administrator), **grupni** (koriste se više projekata) ili **individualni** (namijenjeni specifičnim projektima, kreira ih korisnik)
- **Tagovani** (koriste se za pokretanja poslova koji imaju isti tag) ili **netagovani** (mogi pokrenuti bilo koji posao bez taga)
- **Zaštićeni** (izvršavaju samo poslove iz zaštićenih git grana) i **nezaštićeni** (izvršavaju bilo koji posao)

Tip runner-a se definiše prilikom registracije. Ovo je način na koji runner zna za koje projekte je dostupan.

Registracija runner-a

Svaki korisnik može da registruje svoj runner klikom na **Settings-> CI/CD-> New project runner**.

Nakon odabira između ponuđenih opcija, GitLab će korisniku vratiti komandu koju treba izvršiti na serveru na kojem se pokreće runner (na ovaj način se runner registruje).

```
gitlab-runner register
--url http://89.188.32.141
--token glrt-E5pJ9WSqsyorwAmh6r4i
```

Program gitlab-runner (koji je prethodno instaliran na serveru) će od korisnika tražiti da odabere tip izvršioca (executor) i nakon uspješne registracije potrebno je pokrenuti komandu

```
gitlab-runner run
```

GitLab CI/CD izvršioci

GitLab izvršioci (executori):

- **Shell** – komande se izvršavaju direktno na serveru na kojem je instaliran gitlab runner
- **Docker** – komande se izvršavaju unutar kontejnera koji je kreiran na nekom serveru
- **Docker autoscaler** – slično kao docker, s tim što se kontejneri automatski kreiraju
- **K8s** – poslovi se izvršavaju na Kubernetes klasteru
- **Virtuelna mašina** – komande se izvršavaju unutar virtuelne mašine. GiLab.com koristi ovaj tip runner-a.
- **ssh** – gitlab se putem ssh loguje na server i direktno izvršava komande. U ovom procesu nije potrebno koristiti gitlab-runner agent.