

Sistemi za kontrolu verzija

Sistemi za kontrolu verzija

- ▶ Eng. *Version Control Systems* [**VCS**]
- ▶ Programi koji se koriste kada je potrebno pratiti verzije projekta i/ili kada više učesnika radi na projektu
- ▶ Danas su nezaobilazni u procesu razvoja softvera
- ▶ Datoteke koje se prate zajedno sa svom istorijom promena se nalaze na udaljenom serveru - **repozitorijum**

Sistemi za kontrolu verzija

- ▶ Svaka izmena koja se sačuva na repozitorijumu (**commit**) pored samih izmena čuva i informacije o tome ko je napravio izmenu, kad je izmena napravljena i šta je tačno izmenjeno
- ▶ Dostupan je istorijat izmena, pa je u svakom trenutku moguće pristupiti ranijim verzijama projekta i svim relevantnim podacima

Sistemi za kontrolu verzija

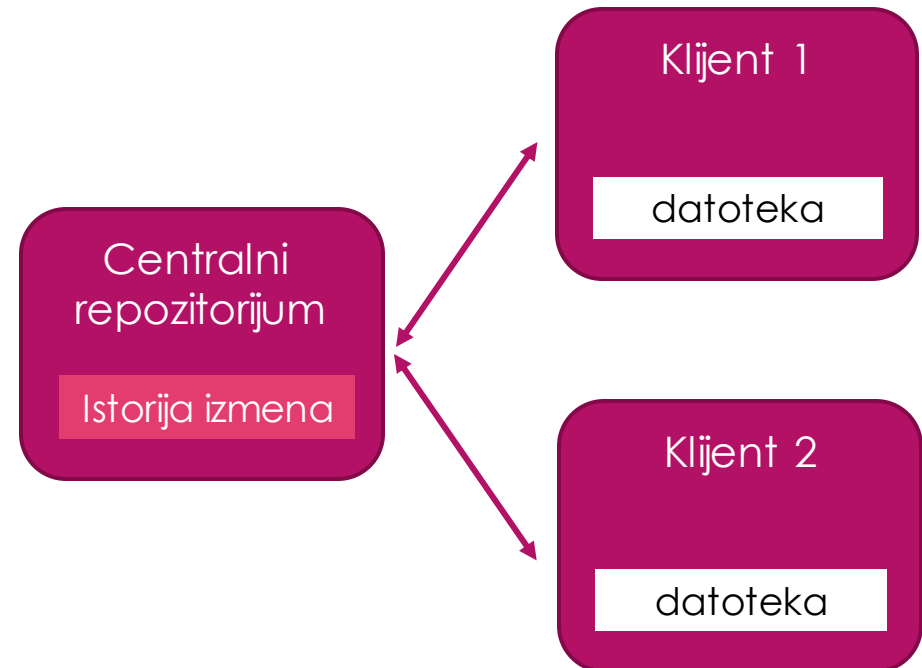
- ▶ Rad sa repozitorijumima i VCS sistemima u velikoj meri olakšava kolaborativni razvoj softvera (u odnosu na ručno održavanje)
 - ▶ omogućavaju integraciju promena nastalu od strane različitih članova tima
 - ▶ omogućavaju dobijanje informacija ko je, kada, gde, promenio određene linije koda
- ▶ Upravljanje personalnim projektima je takođe olakšano jer je mogućnost greške ili gubitka podataka svedena na minimum
- ▶ Ne moraju biti ograničeni samo na programerske projekte!
- ▶ Omogućavaju formiranje više alternativnih tokova razvoja projekta, što pruža podršku eksperimentisanju i nezavisnom razvoju delova projekta

Sistemi za kontrolu verzija

- ▶ Dva osnovna tipa sistema za kontrolu verzija:
 1. **Centralizovani** (Subversion – SVN, CVS, ...)
 2. **Distribuirani** (Git, Mercurial...)
- ▶ Ova podela je načinjena na osnovu načina na koji je implementiran centralni repozitorijum

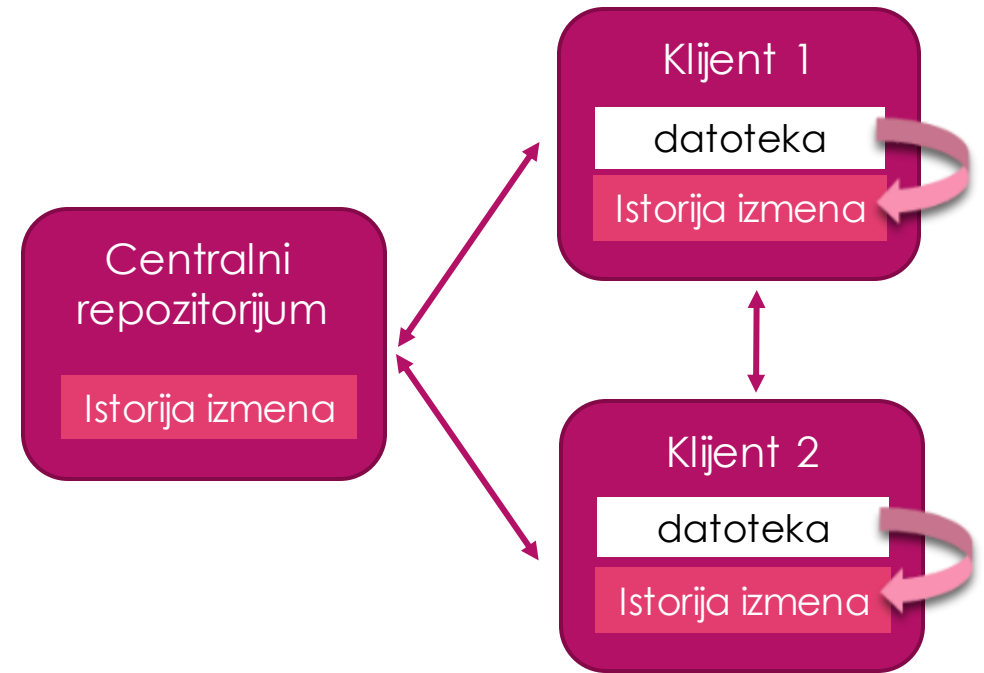
Centralizovani VCS

- ▶ Glavni repozitorijum sa svom istorijom se nalazi na centralnom serveru, dok klijenti preuzimaju svako svoju verziju aktuelnih datoteka
- ▶ Jednostavnija implementacija
- ▶ Osnovni nedostaci ovakve arhitekture su nemogućnost *offline* rada, loša skalabilnost i centralizovani server koji predstavlja *single point of failure* za ceo projekat



Distribuirani VCS

- ▶ Svaki klijent preuzima potpunu kopiju projekta sa servera
- ▶ Svaki čvor predstavlja ravnopravan repozitorijum sa **kompletnom istorijom** izmena (bilo koji može biti proglašen za glavni u slučaju otkaza servera)
- ▶ Dobra skalabilnost – pogodan za velike projekte





GIT

Uvod

- ▶ Distribuirani VCS
- ▶ Osnovna stvar koja razlikuje Git od ostalih VCS sistema (i centralizovanih i distribuiranih) je način na koji Git organizuje podatke u repozitorijumu:
 - ▶ ostali VCS-ovi čuvaju osnovne kopije datoteka i evidentiraju samo eventualne izmene uvede nad tim kopijama
 - ▶ Git za svaku uvedenu izmenu kreira novu sliku stanja celog sistema tako što kreira nove datoteke sa uvedenim izmenama
 - ▶ ovo omogućava timovima da ne budu vezani za samo jedan tok razvoja.

Uvod

- ▶ S obzirom da lokalni git repozitorijumi poseduju kompletnu bazu sa istorijom izmena, potpuni rad sa zajedničkim repozitorijumom je omogućen i bez mrežnog pristupa centralnom serveru
- ▶ Sve načinjene izmene se primenjuju samo na lokalni repozitorijum do trenutka dok se ne pošalju na centralni repozitorijum

Instalacija

- ▶ Preuzeti instalaciju sa službene web stranice: <https://git-scm.com/downloads>
- ▶ Prvobitna namena je korišćenje iz komandne linije
- ▶ Grafička okruženja za Git dostupna su na linku: <https://git-scm.com/downloads/guis>
- ▶ Eclipse ima integrisanu podršku za rad sa Git-om
- ▶ Provera uspešne instalacije: `git -version`

Podešavanja

- ▶ Nakon instalacije, dobra praksa je da se odmah podesi ime i email adresa koje će Git koristiti prilikom čuvanja izmena:

```
git config --global user.name "Ime Prezime"  
git config --global user.email mailadresa@nesto.com
```

- ▶ `--global` će izvršiti podešavanja za **svaki** repozitorijum na računaru, ukoliko se izostavi, podešavanja se vrše samo za onaj repozitorijum na koji smo trenutno pozicionirani iz komandne linije
- ▶ lista trenutno podešenih vrednosti se može dobiti komandom:

```
git config --list
```

Kreiranje lokalnog repozitorijuma

► Dva načina:

1. Inicijalizacijom novog (praznog) repozitorijuma

`git init` je komanda koja će prazan kreirati lokalni repozitorijum. Kreiraće se sakriveni folder **.git** unutar direktorijuma iz kojeg smo pokrenuli komandu (*radnog direktorijuma*)

2. Kloniranjem postojećeg repozitorijuma sa udeljanog servera

`git clone <putanja_do_repozitorijuma> \ <lokalni_direktorijum_gde_se_klonira>` je komanda koja će kreirati lokalni repozitorijum sa svim fajlovima i istorijom izmena sa udeljenog servera. Takođe se u okviru radnog direktorijuma kreira sakriveni folder **.git** koji zapravo označava da je folder git repozitorijum

► U svakom trenutku je moguće dobiti informacije o stanju repozitorijuma komandom `git status`

Praćenje datoteka

- ▶ Kreiranje datoteka u radnom direktorijumu neće navesti Git da prati izmene nad njima
- ▶ Pokretanje praćenja verzija nad nekom datotekom ili direktorijumom se vrši komandom `git add <naziv_datoteke>`:

<code>git add index.html</code>	(praćenje <i>index.html</i> datoteke)
<code>git add css</code>	(praćenje <i>css</i> direktorijuma i svih datoteka unutar njega)
<code>git add css/index.css</code>	(praćenje samo <i>index.css</i> datoteke iz direktorijuma <i>css</i>)
<code>git add .</code>	(praćenje svih izmenjenih ili novih datoteka)

- ▶ Komanda `git add` se mora izvršiti za sve datoteke koje su nove, ali i za sve postojeće datoteke nad kojima su izvršene neke izmene (nalaze se u stanju *modified*)
- ▶ Naknadne izmene nad praćenim datotekama neće biti evidentirane automatski. Prilikom svake izmene potrebno je ponoviti `git add` komandu!

Postavljanje izmena na repozitorijum

► Vršiti se u dve faze:

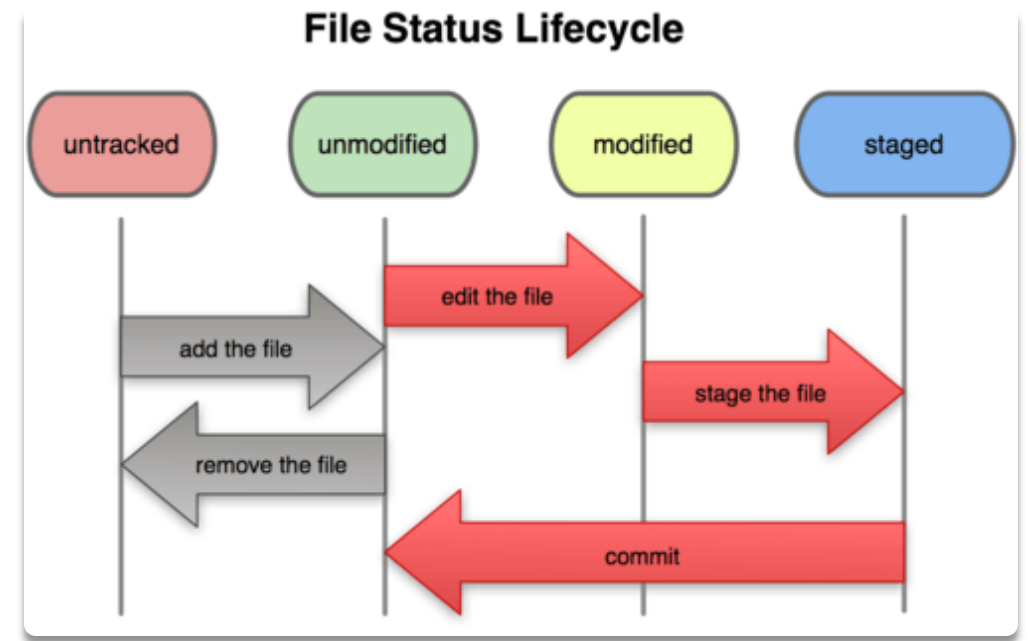
1. postavljanje promena u deo repozitorijuma koji se zove **staging area** (evidentiranje izmena). Datoteke se u ovaj deo repozitorijuma dodaju komandom `git add`. Za objekte koji pripadaju ovom delu se kaže da su u *staged* stanju.
2. "pakovanje" evidentiranih izmena u **commit**. *Commit* operacija predstavlja prebacivanje izmene iz *staging area* na trenutnu granu repozitorijuma. Format naredbe: `git commit -m "opis izmena"`

Saveti oko *commit*-a:

- Logički nezavisne izmene organizovati u odvojenim *commit*-ima
- Jedan *commit* treba biti moguće opisati kroz rečenicu ili dve
- Težiti ka davanju prefiksa opisu *commit*-a (*add*, *fix*, itd.) kako bi se ukazalo na tip izmene

Git file status lifecycle

- ▶ Postavljanje jednog fajla u *staging area*
`git add <ime_fajla>`
- ▶ Postavljanje svih fajlova u *staging area*
`git add .`
- ▶ Brisanje fajla iz fajl sistema i postavljanje te izmene u *staging area*
`git rm <ime_fijla>`
- ▶ Poništavanje izmena
`git reset HEAD <ime_fijla>`



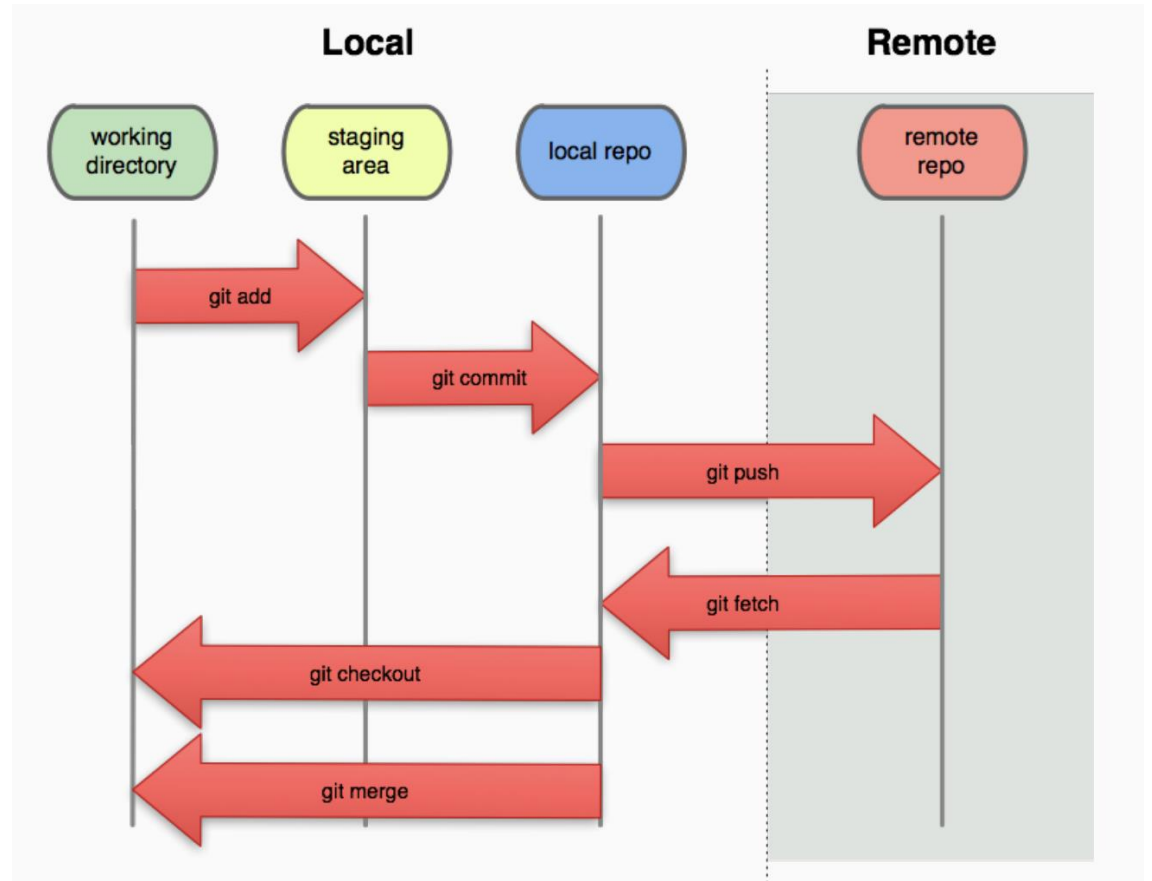
Postavljanje izmena na repozitorijum

- ▶ Operacija `commit` izmene šalje na lokalni repozitorijum
- ▶ Operacija `push` šalje lokalne izmene na udaljeni repozitorijum (može više `commit-a`)
- ▶ Uobičajeni koraci:

```
git add .  
git commit -m "Tekst commit poruke"  
git push origin master
```

referenca na
udaljeni repozitorijum
(*upstream*)

naziv grane
repozitorijuma na
koju se izmene šalju



Ignorisanje datoteka

- ▶ Pokretanje `git add` komande nad celim direktorijumima može dovesti do toga da se među praćenim datotekama nađu i neke koje ne želimo da imamo na repozitorijumu (npr. kompajlirani fajlovi nisu potrebni, dovoljno je da imamo `source` fajlove, logovi, biblioteke...)
- ▶ Git omogućava način da navedemo **pravila za ignorisanje određenih datoteka ili direktorijuma** prilikom pokretanja komande `git add`
- ▶ Pravila se navode u datoteci sa imenom **.gitignore** koji se kreira u repozitorijumu

.gitignore

► Primer .gitignore fajla:

```
# ovo je komentar

# ignoriši sve datoteke u log folderu:
log/

# ignoriši sve tekstualne datoteke:
*.txt

# ali ne i datoteku 'users.txt':
!users.txt

# ignoriši sve css datoteke bilo gde u css folderu:
css/**/*.*css
```

Uklanjanje datoteka

- ▶ Iz radnog direktorijuma i iz *staging area*: `git rm <naziv_datoteke>`
- ▶ Samo iz *staging area*: `git rm -cached <naziv_datoteke>`

Poništavanje izmena

- ▶ Dopuna prethodnog *commit*-a novim izmenama: `git commit --amend`
- ▶ Vraćanje datoteke u *unmodified* stanje: `git checkout -- <naziv_datoteke>`
- ▶ Prebacivanje u stanje određenog *commit*-a: `git checkout <hash>`
- ▶ Prebacivanje u stanje određenog *commit*-a i brisanje kompletne istorije nastale posle tog *commit*-a: `git reset -hard <hash>`

git stash komanda

- ▶ Koristimo je kako bi lokalno spremili trenutno stanje radnog direktorijuma (bez *commit*-ovanja) kako bi mogli da se prebacimo na rad na nečemu drugom
- ▶ Promene je moguće spremiti više puta, pri čemu se za svako spremanje kreira novi zapis u lokalnom repozitorijumu
- ▶ Nakon spremanja izmena, radni direktorijum se vraća u stanje poslednjeg *commit*-a
- ▶ Lista spremljenih izmena se može dobiti komandom `git stash list`
- ▶ Vraćanje spremljenih izmena se vrši komandom `git stash apply`
- ▶ Ovo će vratiti poslednje spremljeno stanje (sa vrha liste)

Stanje repozitorijuma

- ▶ `git status`
- ▶ Prikazuje izmene koje:
 - ▶ treba da budu postavljene u *staging area*
 - ▶ su postavljene u *staging area*, ali nisu na repozitorijum
- ▶ Prikazuje i informacije kao što su koja je trenutno aktivna grana i broj *commit*-a koji nisu poslani na udaljeni repozitorijum

```
udd — -bash — 80x24
~/Documents/Faks/udd/udd — -bash
[Katarinas-MacBook-Pro:udd katarinapreradov$ git add doc1.txt
[Katarinas-MacBook-Pro:udd katarinapreradov$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   doc1.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   eBooksClient/package.json

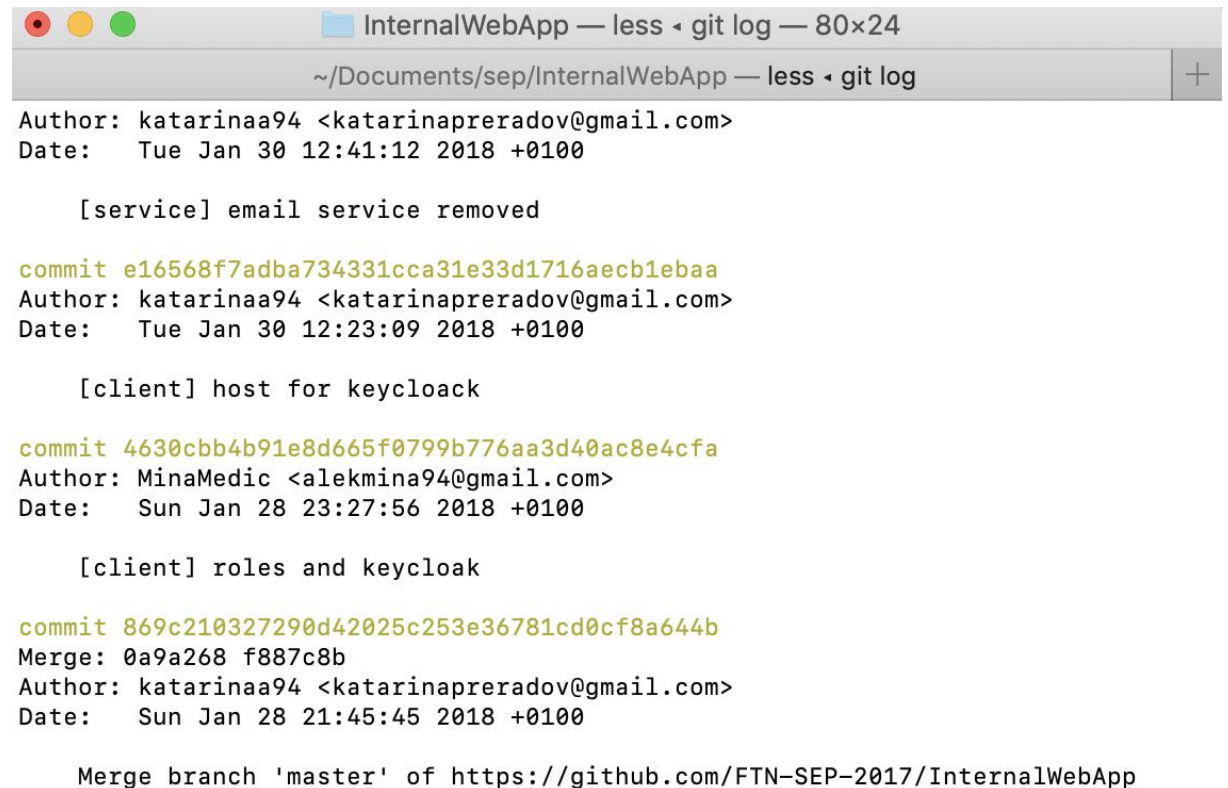
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .DS_Store
    doc2.txt

Katarinas-MacBook-Pro:udd katarinapreradov$
```

Istorija repozitorijuma

- ▶ `git log`
- ▶ Prikazuje sve *commit*-ove
- ▶ Za svaki *commit* je prikazan opisom, autor i datum
- ▶ Svaki *commit* ima svoj jedinstven identifikator (*hash*)



```
InternalWebApp — less ◀ git log — 80x24
~/Documents/sep/InternalWebApp — less ◀ git log
Author: katarinaa94 <katarinapreradov@gmail.com>
Date: Tue Jan 30 12:41:12 2018 +0100

[service] email service removed

commit e16568f7adba734331cca31e33d1716aecb1ebaa
Author: katarinaa94 <katarinapreradov@gmail.com>
Date: Tue Jan 30 12:23:09 2018 +0100

[client] host for keycloak

commit 4630cbb4b91e8d665f0799b776aa3d40ac8e4cfa
Author: MinaMedic <alekmina94@gmail.com>
Date: Sun Jan 28 23:27:56 2018 +0100

[client] roles and keycloak

commit 869c210327290d42025c253e36781cd0cf8a644b
Merge: 0a9a268 f887c8b
Author: katarinaa94 <katarinapreradov@gmail.com>
Date: Sun Jan 28 21:45:45 2018 +0100

_ Merge branch 'master' of https://github.com/FTN-SEP-2017/InternalWebApp
```


Preuzimanje ranije izmene

► `git checkout <hash>`

*Hash kod željenog
commit-a*

► Ovom komandom se lokalni repozitorijum može vratiti u stanje u kakvom je bio u određenom trenutku

Rad sa granama

- ▶ Omogućavaju izolovan rad na različitim komponentama sistema i pružaju dobru podršku testiranju ideja
- ▶ Novokreirani repozitorijumi imaju samo jednu granu koja se zove **master**
- ▶ Razvijamo komponentu i kad smo zaokružili celinu spajamo (**merge**) kod komponente sa glavnom granom
- ▶ Preporuka je da **master** grana (glavna) ima samo *merge* čvorove => da
- ▶ se ništa ne radi na master grani direktno
- ▶ **HEAD** pokazivač pokazuje na kojoj grani i čvoru smo trenutno pozicionirani

Rad sa granama

- ▶ Kreiranje grane: `git branch <ime_grane>`
- ▶ Pozicioniranje na granu: `git checkout <ime_grane>`
 - ▶ Pomera HEAD pokazivač na datu granu i menja sadržaj radnog direktorijuma tako da oslika stanje repozitorijuma u datom čvoru
- ▶ Prikaz svih grana: `git branch`
 - ▶ U spisku prikazanih grana stoji * pored imena grane na koju smo trenutno pozicionirani
- ▶ Brisanje grane: `git branch -d <ime_grane>`

Spajanje grana

- ▶ Prilikom spajanja, potrebno je vratiti se na roditeljsku granu i pozvati komandu `git merge <druga_grana>`
 - ▶ u granu na kojoj smo trenutno pozicionirani ubacuje i izmene napravljene na grani `<druga_grana>`
 - ▶ Izmene se "spajaju" tako da novi čvor sadrži izmene napravljene i na prvoj i na drugoj grani

```
git checkout master
Switched to branch 'master'

git merge develop
Updating 1e0fb07..d656b7b
Fast-forward
index.html | 3 +--
1 file changed, 1 insertion(+), 2 deletions(-)
```

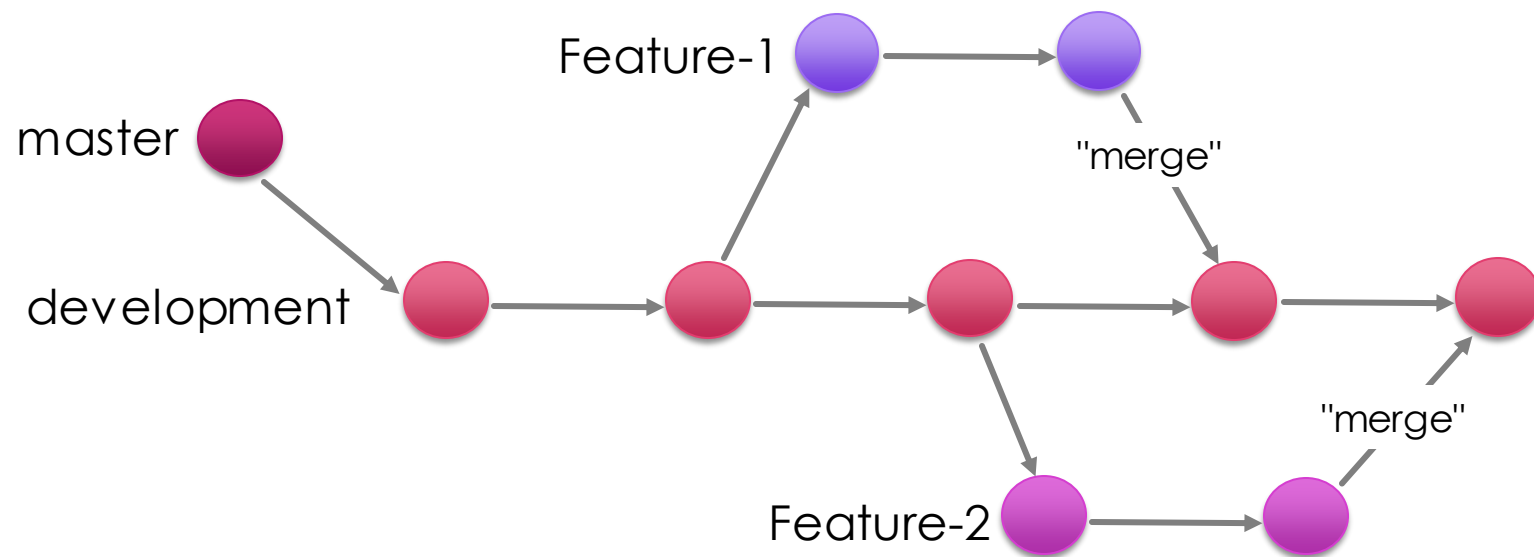
git rebase

- ▶ Drugi način spajanja grana
- ▶ Ne kreira novi commit
- ▶ Uglavno sliži u svrhe dopunjavanja grana izmenama sa master grane

Udaljene grane

- ▶ Sve kreirane grane se nalaze na lokalnom repozitorijumu
- ▶ Ukoliko želimo da objavimo našu granu na udaljenom serveru koristimo sledeću naredbu: `git push origin <naziv_grane>`

Grananje



Konflikti

- ▶ Automatsko spajanje izmena će biti uspešno izvršeno samo ako u granama koje se spajaju ne postoje izmene u istim linijama istih datotetka.
- ▶ Ako ovo nije slučaj, datoteka čiji sadržaj nije mogao biti "spojen" je u konfliktu
- ▶ U datoteci koja je u konfliktu označene su konfliktne linije

Primer konfliktnog fajla

- ▶ Pretpostavka: linija 1 je menjana na dve različite grane
- ▶ Na mestu linije 1 git je izgenerisao sledeći kod:

```
<<<<<< HEAD:index.html
contact : email.support@github.com
=====
please contact us at support@github.com
>>>>>> develop:index.html
```

- ▶ Potrebno je srediti sadržaj konfliktnog datoteke tako da se obrišu oznake i neželjeni sadržaj
- ▶ Nakon što se uredi sadržaj, potrebno je dodati datoteku na *staging area* koristeći `git add` komandu (jer će sređivanje konflikta generisati novu izmenu), što razrešava konflikt

Rad sa udaljenim repozitorijumom

- ▶ Iako su svi repozitorijumi u Git-u ravnopravni, najčešći scenario za timski rad je da postoji jedan javno dostupan repozitorijum koji čuva izmene svih članova tima
- ▶ Ovom udaljenom repozitorijumu se obično dodeljuje ime **origin**
- ▶ Pri kloniranju repozitorijuma, za origin se postavlja repozitorijum čiji se sadržaj klonira
- ▶ `git remote add origin <adresa_repozitorijuma>`
- ▶ Lokalni git repozitorijum može da bude podešen tako da radi sa više udaljenih repozitorijuma

Slanje izmene na udaljeni repozitorijum

- ▶ Slanje sadržaja lokalnog repozitorijuma na udaljeni repozitorijum

```
git push <ime_ili_adresa_repozitorijuma> <ime_grane_koju_saljemo>
```

- ▶ Primer: `git push origin master`

Preuzimanje izmena sa udaljenog repozitorijuma

- ▶ **Preuzimanje izmena** sa udaljenog repozitorijuma:

```
git fetch <ime_ili_adresa_repozitorijuma>
```

- ▶ Primer: `git fetch origin` (origin se podrazumeva, pa se može i izostaviti)

- ▶ Preuzete izmene je potrebno onda **spojiti** sa lokalnom granom:

```
git merge origin/master (spajanje se radi sa master granom udaljenog repozitorijuma)
```

- ▶ Prethodne dve naredbe se mogu objediniti u jednu:

```
git pull <ime_ili_adresa_repozitorijuma>
```

- ▶ Radi fetch, pa zatim merge

Konflikti pri preuizimanju izmena sa udaljenog repozitorijuma

- ▶ Dva korisnika inicijalno povuku istu verziju datoteke (`git pull`), nakon čega oba korisnika izmene datoteku i sačuvaju izmene u lokalni repozitorijum
- ▶ Oba korisnika izvrše `git push` naredbu da pošalju izmene na udaljeni repozitorijum:
 - ▶ prvi `push` prolazi bez problema i čuva se izmena na repozitorijumu
 - ▶ drugi `push` zahteva od korisnika da preuzme najnoviju verziju udaljenog repozitorijuma pre nego što može da izvrši `push`
 - ▶ ukoliko u ovom trenutku izvrši operaciju `pull`, drugi korisnik dobija konflikt
 - ▶ konflikt se razrešava na isti način kao kod `merge`-a lokalnih grana.
- ▶ preporuka je raditi `git fetch` i onda `git merge` kako bi imali veću kontrolu nad procesom i veću svest o tome šta se zapravo zmenilo