Verzioniranje izvorne kode

prof. dr. Matjaž. B. Jurič

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Laboratorij za integracijo informacijskih sistemov

Verzioniranje izvorne kode

- Težave, do katerih lahko prihaja pri ekipnem razvoju programske kode, kadar ne uporabljamo sistema za verzioniranje izvorne kode:
  - Redundantno delo (dva razvijalca hkrati razvijata isto funkcionalnost).
  - Otežena koordinacija dela med več avtorji.
  - Ni sledljivosti nad opravljenimi spremembami.
  - Težje zaznavanje konfliktov in njihovo razreševanje.
  - Oteženo izvajanje sprememb (npr. odprava napak) v starejših verzijah.
  - Oteženo upravljanje pravic dostopa (avtentikacija).

Verzioniranje izvorne kode

- Sistem za upravljanje verzij (**Version Control System**) je programski produkt, ki skrbi za verzioniranje in poenostavljeno izmenjavo datotek ter tako olajša skupinsko delo.
  - **Centralizirani** sistem za upravljanje verzij (npr. SVN)
  - **Porazdeljeni** sistem za upravljanje verzij (npr. Git, Mercurial)

- Sistemi za upravljanje izvorne kode se najpogosteje uporabljajo pri razvoju programske opreme (**Source Code Control System**).

- Upravljanje izvorne kode spada v okvir upravljanja konfiguracij (**SCM – Source Configuration Management**).

Verzioniranje izvorne kode

- Trenutno najbolj popularni sistemi za upravljanje verzij:
  - Git
  - Apache Subversion (SVN)
  - Mercurial
  - CVS (Concurrent Versions System)
  - IBM ClearCase
  - IBM Rational Team Concert
  - itd.

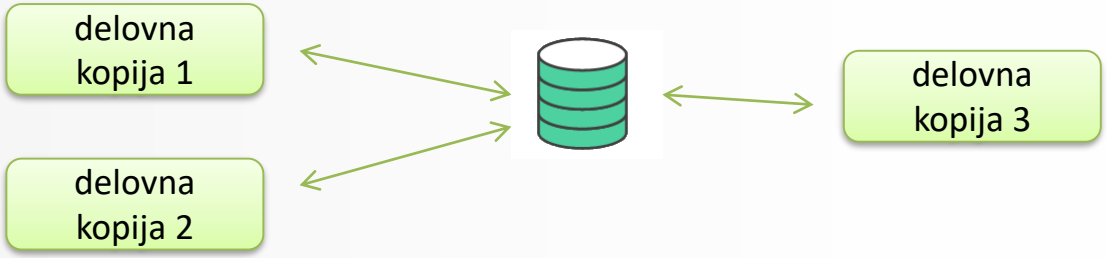
Verzioniranje izvorne kode

- Ključna komponenta sistema za upravljanje verzij je **skladišče** (**repozitorij**), ki predstavlja shrambo (centralno ali distribuirano shrambo) izvorne kode.

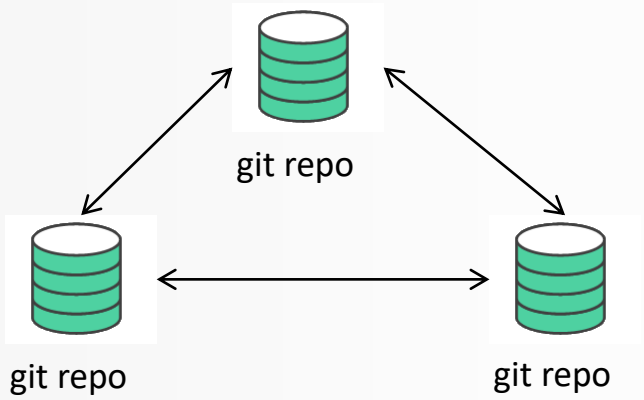
- Razvijalec najpogosteje izvaja naslednje akcije:
  - **checkout/merge**: prevzem izvorne kode iz repozitorija in kreiranje delovne kopije.
  - **update/fetch/pull**: posodabljanje delovne kopije s spremembami iz repozitorija.
  - **commit/push**: objava lokalnih sprememb v repozitorij.

Sodelovanje

- Centralizirano
  - Sodelovanje med strežnikom in razvijalci

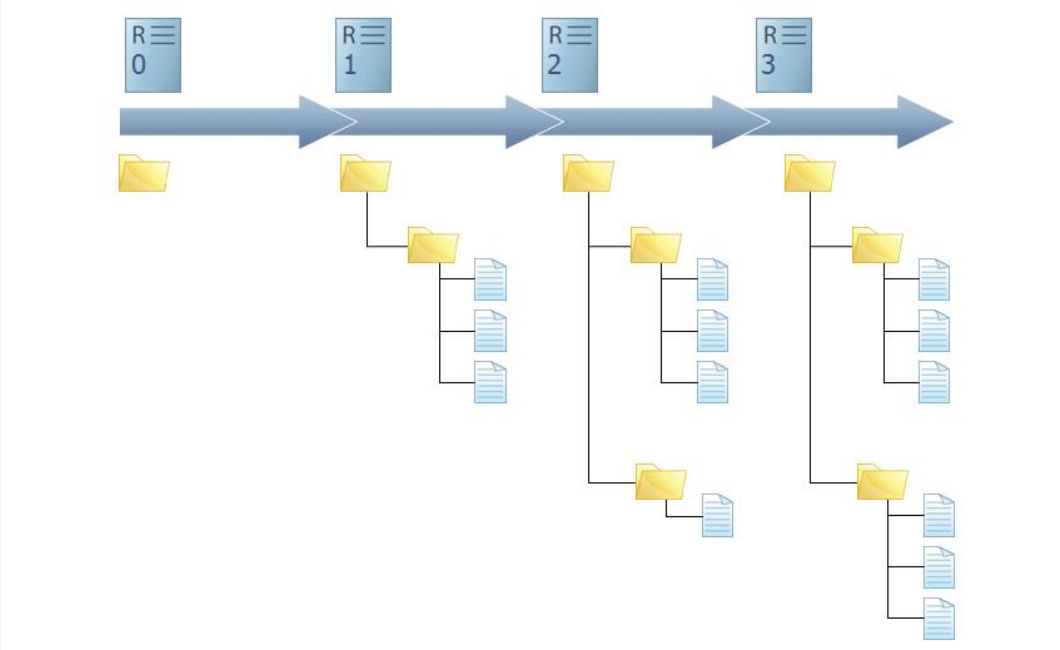


- Porazdeljeno
  - Sodelovanje med repozitoriji



Verzije oz. revizije

- Verziji pravimo tudi revizija.
- Za vsako objavljeno spremembo se spremeni (poveča) verzija verzionirane datoteke.
- Zadnjo revizijo v repozitoriju običajno imenujemo HEAD revizija.



Pomembne operacije

- Nekatere pomembne operacije, ki jih podpira večina SCM sistemov:
  - **checkout/merge**: prevzem izvorne kode iz repozitorija in kreiranje delovne kopije.
  - **update/fetch/pull**: posodabljanje delovne kopije s spremembami iz repozitorija.
  - **commit/push**: objava lokalnih sprememb v repozitoriju.
  - **import**: uvoz doslej neverzioniranih datotek oz. map v repozitorij.
  - **export**: izvoz datotek oz. map iz repozitorija, tako da te več niso verzionirane.
  - **diff**: ogled razlik med dvema različnima revizijama.

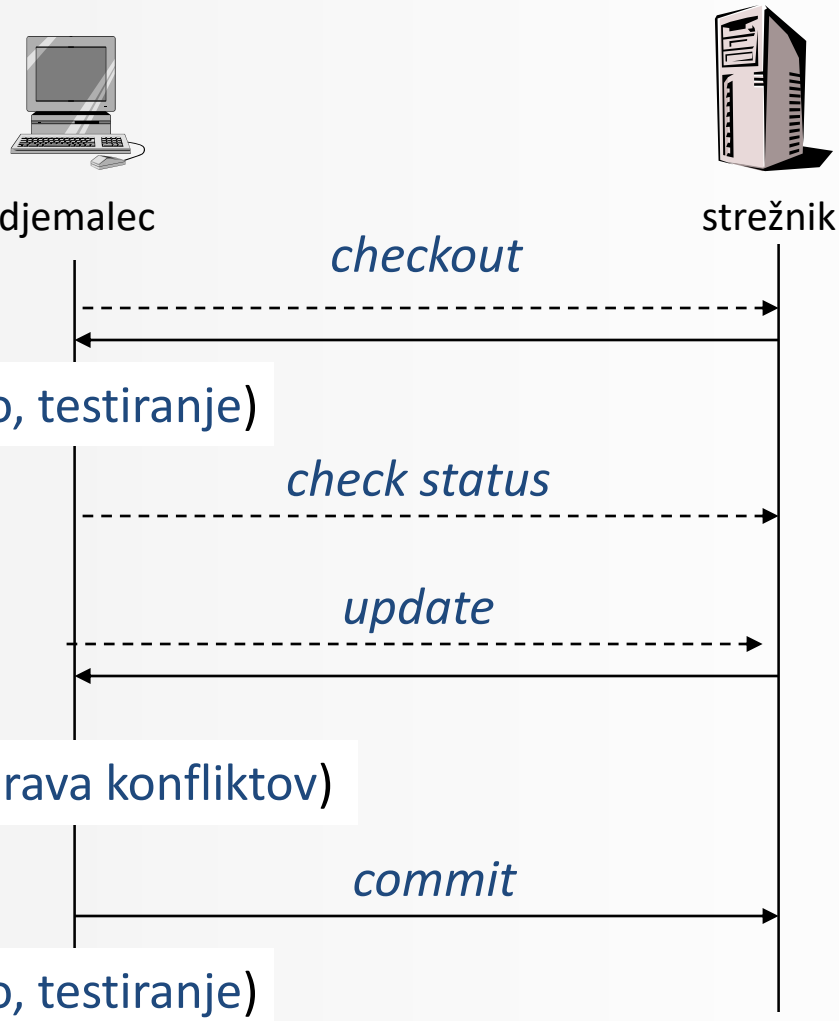
Pomembne operacije

- Nekatere pomembne operacije, ki jih podpira večina SCM sistemov:
  - **delete/rm**: izbris datotek oz. map iz repozitorija.
  - **lock**: zaklep virov v repozitoriju, da drugim razvijalcem preprečimo spreminjanje.
  - **revert**: razveljavitev vseh lokalnih sprememb.
  - **copy/branch**: izdelava kopije datoteke oz. mape v repozitoriju. Najpogosteje se uporablja za kreiranje novih vej (branches) ali posnetkov (snapshots).
  - **switch/checkout**: prestavitev delovne kopije na drugo vejo ali oznako.
  - **merge**: združevanje sprememb iz dveh vej.

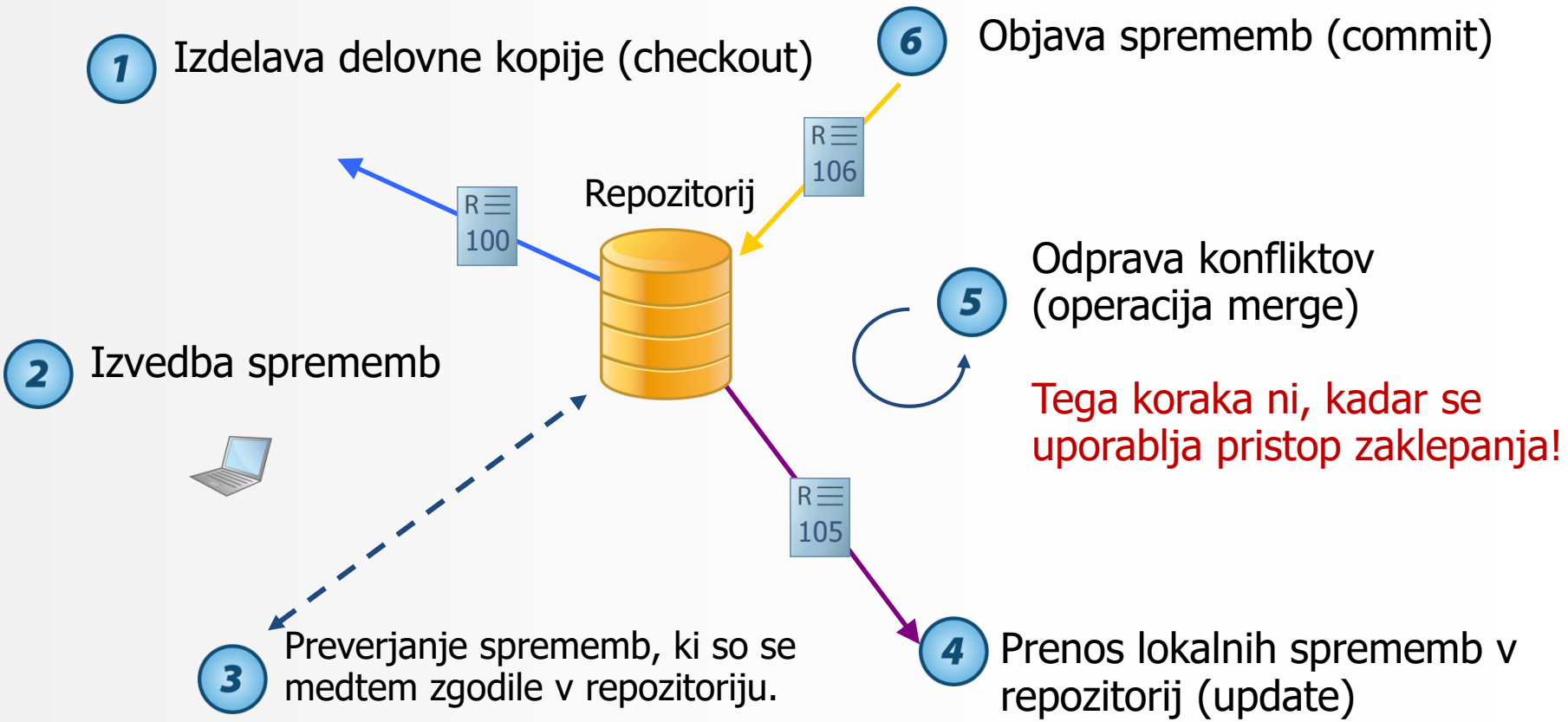
Modeli verzioniranja

- Uporabljata se dva modela sistemov za upravljanje verzij:
  - **Kopiraj-spremeni-združi (Copy-modify-merge)**:
    - Več ljudi lahko hkrati spreminja isto datoteko
    - Potrebno je razreševanje konfliktov
    - Princip optimističnega zaklepanja (Optimistic locking)
  - **Zakleni-spremeni-odkleni (Lock-modify-unlock):**
    - Datoteko lahko naenkrat spreminja le en uporabnik
    - Zaklepanje datotek
    - V velikih projektih je “čakanje na odklep” neučinkovito
    - Princip pesimističnega zaklepanja (Pessimistic locking)

Delovni cikel

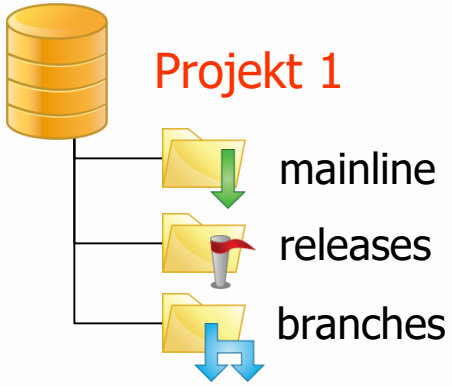


Delovni cikel

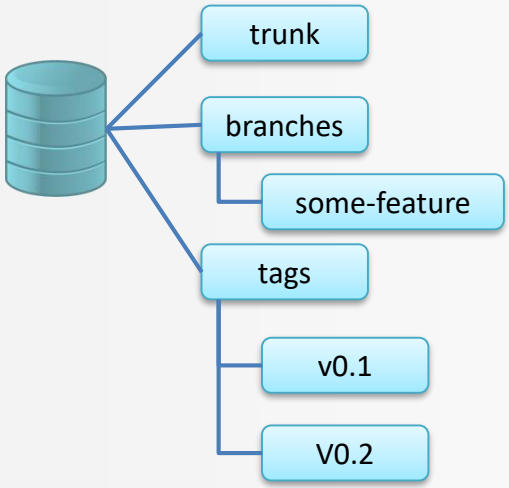


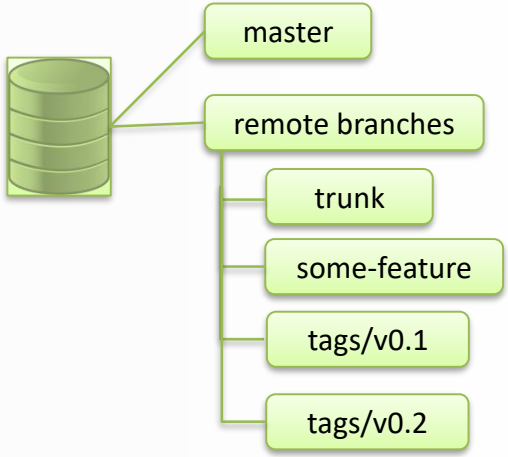
Struktura repozitorija

- Priporočena struktura repozitorija:
  - Glavna veja razvoja (Main Line)
  - Vzporedne veje razvoja (Branches)
  - Posnetki (Snapshots)



- Uporabljamo lahko tudi druge konvencije poimenovanja.





Posnetek (Snapshot)

- Zakaj potrebujemo posnetke?
  - Arhiviranje izdanih verzij
  - Izdelava posnetka trenutnega stanja (snapshot)

- Tipično poimenovanje:
  - Release-1.0.0
  - REL-0.3.0RC1

- Naziv posnetka mora biti enoličen

- **Vsebine posnetkov ne smemo spreminjati** !!

Veja (Branch)

- Branch predstavlja vzporedno vejo razvoja, ki je ločena od glavne veje.

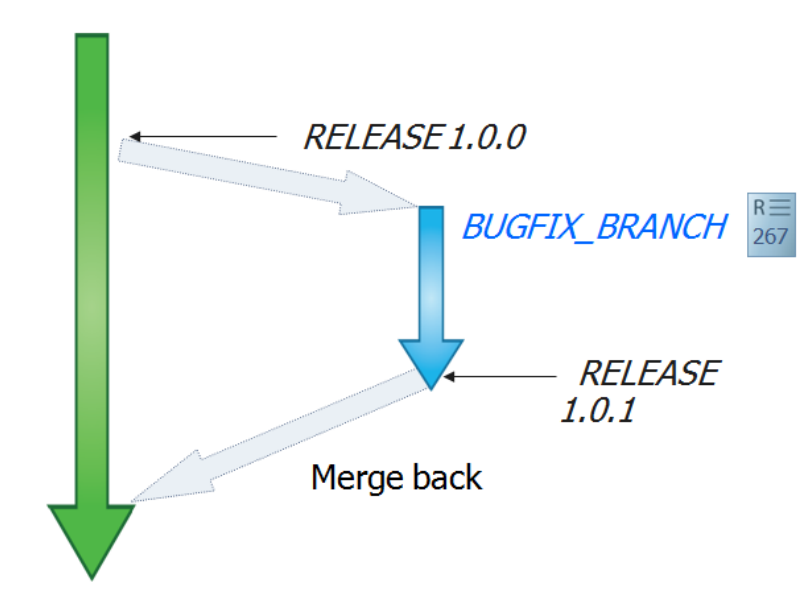
- Kdaj je priporočljivo uporabiti branch?

  - ❑ Primer 1:
    - ➢ Tik pred izdajo nove verzije produkta, se en del ekipe pripravlja na izdajo (release) in želi zgolj testirati in odpravljati hrošče v okviru obstoječe funkcionalnosti, medtem pa drugi del ekipe pa nadalje razvija nove funkcionalnosti.
    - ➢ Da si obe skupini razvijalcev “ne hodita v zelje” lahko del ekipe, ki pripravlja release, kreira svojo vejo (branch) in tako v miru testira in odpravlja napake, ostali v razvoju pa normalno še naprej uporabljajo glavno vejo.
    - ➢ Po izdaji nove verzije se na podlagi brancha za izdano verzijo kreira posnetek izdaje (release snapshot).
    - ➢ Nato ves nadaljnji razvoj do izdaje nove verzije zopet poteka v okviru glavne veje.

Veja (Branch)

- Kdaj je priporočljivo uporabiti branch?

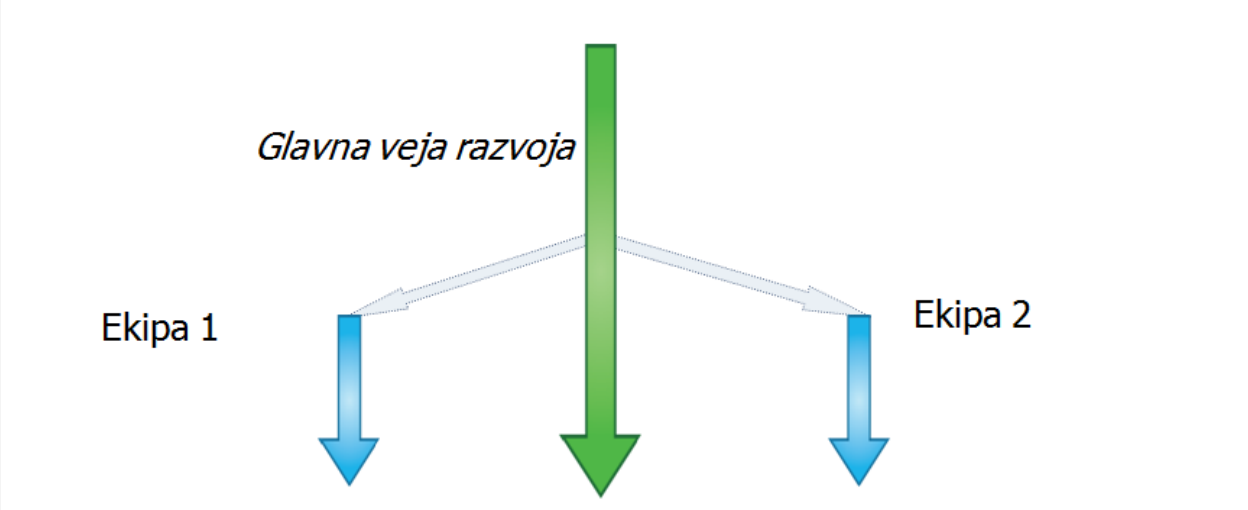
  - Primer 2:
    - Uporabniki odkrijejo pomemben hrošč v starejši verziji (npr. 1.0.0), za katero imamo na voljo ustrezen posnetek (RELEASE 1.0.0).
    - Na podlagi posnetka kreiramo vejo (BUGFIX_BRANCH) in v tej veji odpravimo napako.
    - Izdamo novo verzijo (1.0.1) in zanjo kreiramo posnetek (RELEASE 1.0.1).
    - Na koncu vse spremembe iz veje prenesemo nazaj v glavno vejo (merge).



Veja (Branch)

- Kdaj je priporočljivo uporabiti branch?

  - Primer 3:
    - Pri razvoju velikega projekta sodeluje več ekip, ki želijo biti med seboj čim bolj izolirane, da se izognejo pogostim konfliktom.
    - Vsaka ekipa uporablja svoj branch, v katerega je potrebno v rednih časovnih intervalih (najbolje enkrat dnevno) z operacijo merge prenesti vse spremembe iz glavne veje razvoja.
    - Prav tako je potrebno v rednih časovnih intervalih v glavno vejo prenesti vse spremembe iz vseh vzporednih vej.



Konflikti

- Konflikt nastane, ko bila datoteka lokalno spremenjena, prav tako pa je bila spremenjena tudi verzija v repozitoriju, ali pri združevanje dveh različnih vej razvoja (operacija merge).

- Do konfliktov lahko pride samo pri uporabi modela **Kopiraj-spremeni-združi**, ko se viri ne zaklepajo in lahko več ljudi hkrati spreminja datoteke.

- Konflikt lahko razrešimo na tri načine:
  - Združitev vseh sprememb v eno datoteko.
  - Zavržba lokalnih sprememb in sprejetje oddaljene verzije.
  - Prepis verzije v repozitoriju z lokalnimi spremembami.

Dobre prakse pri verzioniranju izvorne kode (1/2)

- Uporaba priporočene strukture repozitorija (glavna veja, vzporedne veje in posnetki).
- Za vsak projekt se priporoča kreiranje svojega repozitorija.
- Prenesemo samo tiste datoteke/projekte, ki jih dejansko potrebujemo.
- **Pred vsako akcijo commit najprej preverimo status spremenjenih datotek**!
- Akciji commit in update izvajamo **čim bolj pogosto**.
- V repozitorij **nikoli** ne objavimo kode ki vsebuje napake!
- Ob vsakem objavi podamo komentar, ki opisuje spremembe.

Dobre prakse pri verzioniranju izvorne kode (2/2)

- Datoteke, ki jih nočemo verzionirati (npr. .class, .jar) označimo z ukazom ignore.
- Za vsako novo verzijo kreiramo posnetek.
- Veje (branches) uporabljamo za odpravljanje napak, eksperimentiranje ter pri velikih projektih, ko se želimo izogniti pogostim konfliktom.
- Priporočljivo se je izogibati prevelikemu številu vzporednih vej, saj je združevanje vej težavno opravilo. Vejo kreiramo samo če jo res potrebujemo (velja za SVN, Git teh težav nima).

Verzioniranje izvorne kode

- Prednosti uporabe sistema za upravljanje verzij:
  - Lažja koordinacija dela v razvojni skupini
  - Sledljivost nad spremembami
  - Napredno upravljanje pravic dostopa
  - Lažje zaznavanje konfliktov in njihovo razreševanje
  - Poenostavljeno vzdrževanje izdanih verzij
  - Poenostavljena izdaja novih verzij

Razlike Git - SVN

- SVN
  - Centraliziran sistem na nadzor verzij
  - Vzpostavitev repozitorija na drugi lokaciji kot je razvijalno okolje
  - Vsebino projekta shranjuje kot datoteke
  - Kapacitete diska so nujne le za velikost verzije projekta
  - Lažji in manj kompleksnejši za uporabo

- GIT
  - Porazdeljen sistem za nadzor verzij (Distributed Version Control System)
  - Lokalni razvijalni repozitorij je omogočen za deljeno branje
  - Vsebino projekta shranjuje kot metapodatke
  - Potrebne večje kapacitete diska (celotna zgodovina verzioniranega projekta)
  - Zelo hitro delovanje, ker je večina operacij izvedenih znotraj lokalnega repozitorija
  - Lokacija dela ni pomembna, ker ne rabimo bit povezani z repozitorijem, ko želimo upravljat z verzijami datotek

GIT

Git

- Trenutno najbolj popularen odprto-kodni sistem za upravljanje verzij.

- Razvit je bil kot naslednik propadlega sistema DVCS (Distributed version control system) BitKeeper.

- Temelji na osnovi **„trenutnega posnetka“** (angl. snapshot)
  - Pri vsaki potrditvi spremembi kode se ustvari posnetek vseh datotek. V primeru, da se določena datoteka ni spremenila, se le ta ne shrani ponovno, ampak se shrani le povezava do identične datoteke, ki je že shranjena.

Git Uporaba

- **Google**
  - Android projekti
- **Microsoft**
  - Izvorna koda ogrodja Asp.Net MVC je dosegljiva na codeplex-u
- Glavni projekti skupnosti Java (**JBoss, Spring, Apache, Eclipse**) so v procesu migracije iz SVN v Git.

Prednosti

- Redundanca in podvajanje podatkov
  - Zaradi večjega števila celotnih repozitorijev je tveganje, da izgubimo podatke manjše
- Visoka razpoložljivost
  - Kljub težavam z omrežje lahko razvoj aplikacij poteka nemoteno naprej, saj je omogočeno delo z lokalnim repozitorijem.
- Samo en imenik .git na repozitorij
  - V nasprotju s SVN, kjer je imenik .svn definiran v vsaki mapi posebej kar lahko privede do težav
- Lastnosti, kot so »ignore« je veliko lažje upravljati kot v SVN
- Prijazno sodelovanje
  - Git je enostaven za sodelovanje večjega števila uporabnikov znotraj istega projekta

Delovanje

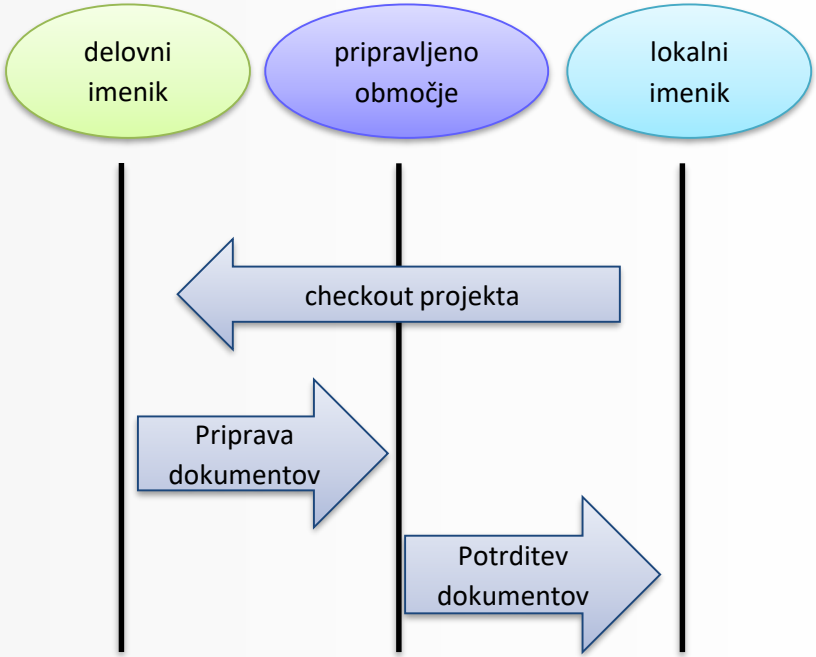
- **Celotna zgodovina projekta je na lokalnem disku**
  - Posledično lokacija dela in dostopnost do omrežja pri razvoju nista več ovira.
- **Večina operacij Git dela le z lokalnimi podatki in viri**
  - V splošnem tako niso potrebne informacije o drugih računalnikih znotraj omrežja.
- **Integriteta**
  - Pred vsako potrditvijo se izvede preverjanje s pomočjo „ checksum“. Posledično ni možno spremeniti vsebine datoteke ali mape, ne da bi Git vedel za spremembe. Za „checksumming“ se uporablja zgoščena vrednost SHA-1.
- **Dodajanje podatkov**
  - V splošnem, Git ob vsaki potrditvi spremembe le doda vse relevantne podatke v podatkovno bazo Git in ne izbriše vsebine starih datotek. S tem se prepreči, da se podatki hitro izgubijo.

Stanje datotek

- Datoteke znotraj Git so lahko v treh različnih fazah:
  - **Spremenjena** (modified)
    - Datoteka je spremenjena, vendar še ni potrjena v podatkovno bazo
  - **Pripravljena** (staged)
    - Spremenjena datoteka je označena in pripravljena za dokumentiranje pri naslednji potrditvi snapshot-a.
  - **Potrjena** (commited)
    - Podatki so varno shranjeni v lokalni podatkovni bazi

Glavne komponente Git repozitorija

- Iz tega sledi, da je projekt Git sestavljen iz treh glavnih komponent
  - **Delovni imenik** (working directory) – enojni checkout verzije projekta. Datoteke so izvožene iz podatkovne baze imenika Git in razvijalcem na razpolago za modifikacijo
  - **Pripravljeno območje** (staging area) – datoteka, ki shranjuje informacije o naslednji potrditvi
  - **Imenik Git** (Git directory – repository) – prostor kjer Git shranjuje metapodatke in objekte podatkovne baze projekta

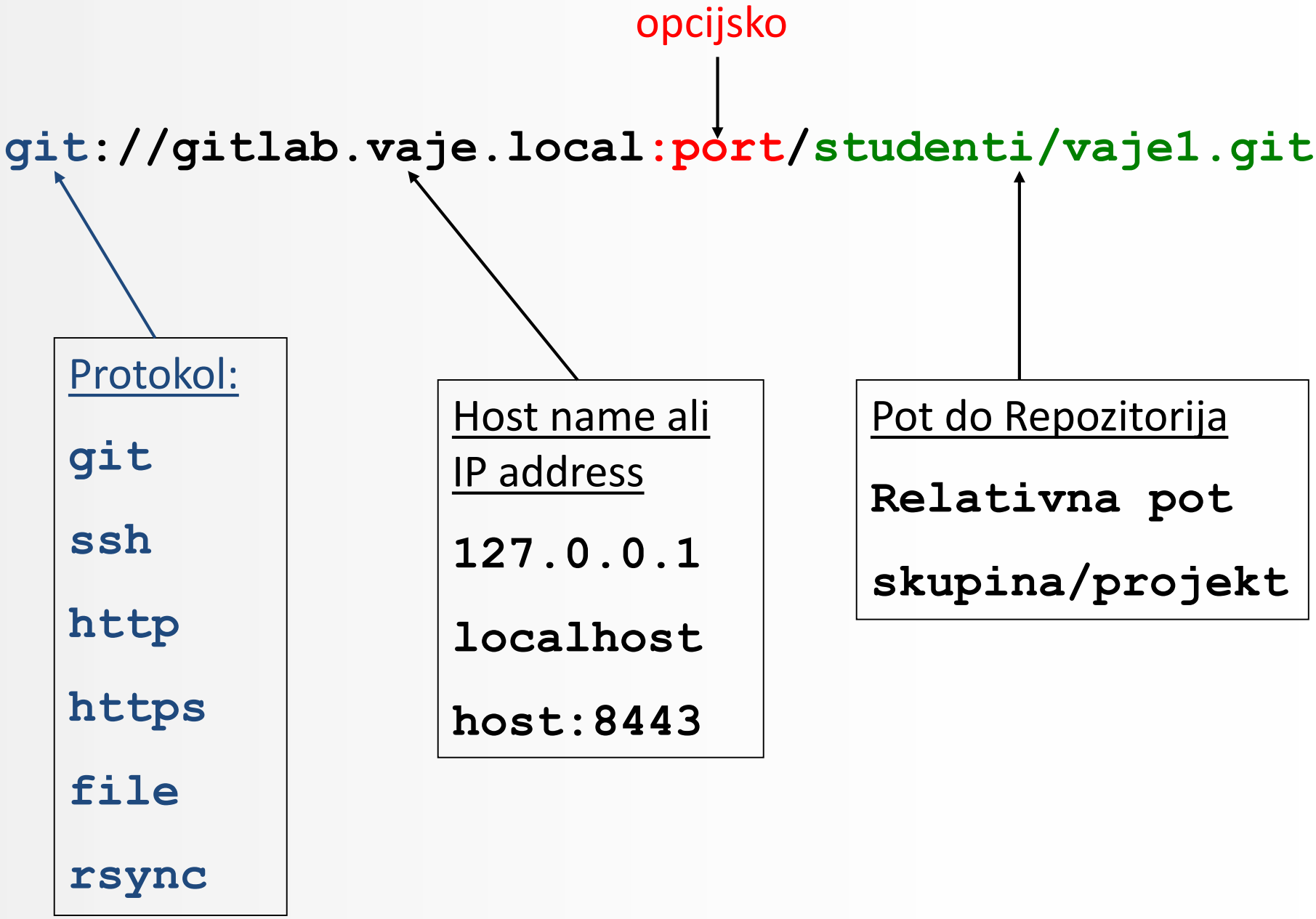


Dostop do Git repozitorija

- Podprti so naslednji načini dostopa do repozitorija:
  - Git protokol (`git://host.xz[:port]/path/to/repo.git/`)
  - Lokalni dostop (`file:///path`)
  - http in https dostop (`http[s]://host.xz[:port]/path/to/repo.git/`)
  - SSH protokol preko TCP/IP (`ssh://[user@]host.xz[:port]/path/to/repo.git`)
  - rsync protokol (`sync://host.xz/path/to/repo.git/`)

- Git ukaze lahko izvajamo:
  - Ročno s pomočjo ukazne vrstice (posh Git)
  - S pomočjo GUI orodij, kot sta npr. GitExtension, KDiff.
  - S pomočjo vtičnikov v razvojnih orodjih (npr. Git plugin za Eclipse - EGit)

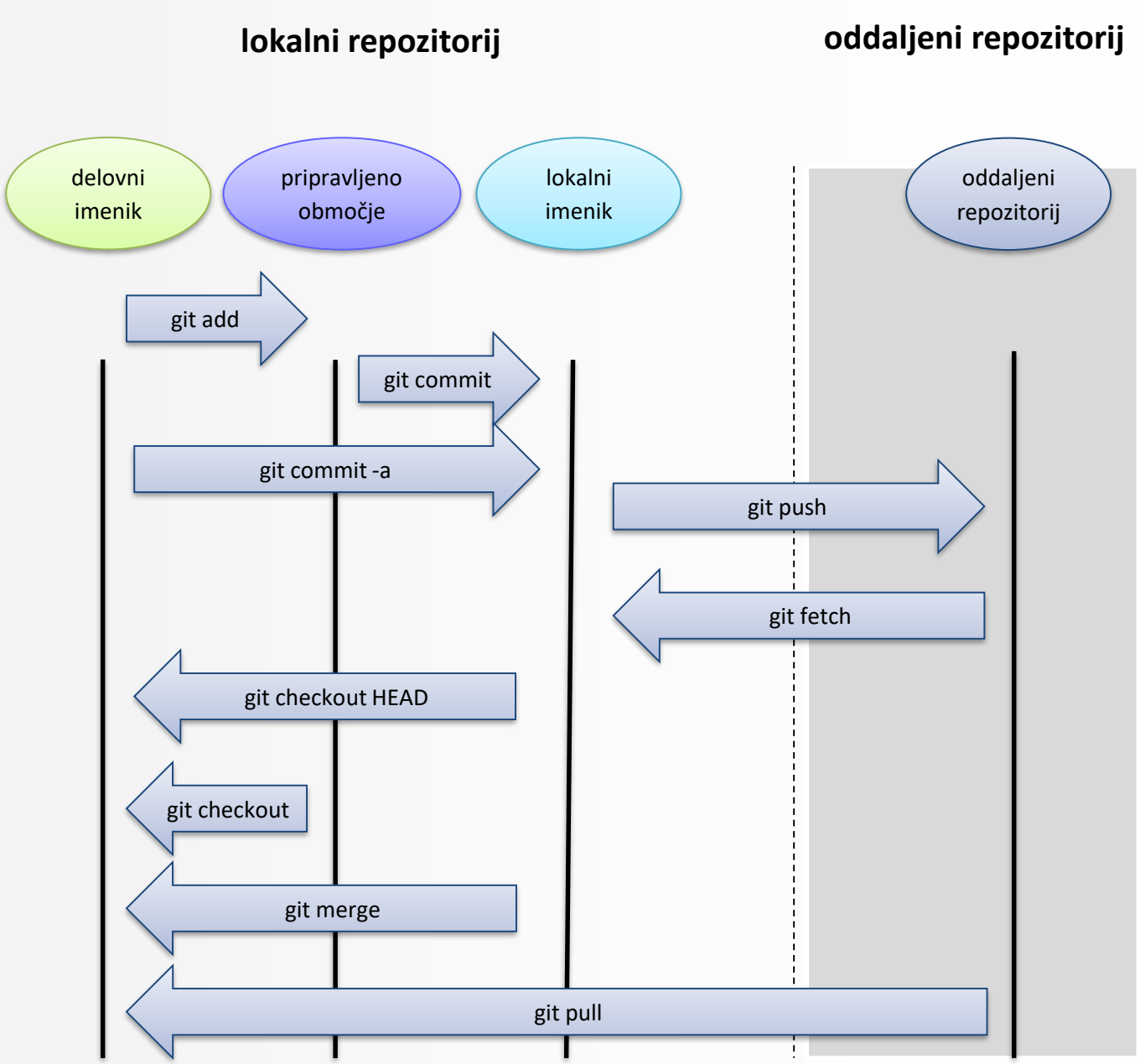
Dostop do Git repozitorija



Osnovni delovni tok Git

1. Modifikacija datotek znotraj delovnega imenika
2. Priprava dokumentov – dodajanje trenutnih posnetkov v pripravljeno območje
   - **git add**
3. Potrjevanje datotek – trajno shranjevanje trenutnih posnetkov iz pripravljenega območja v lokalni imenik
   - **git commit**
4. Potisni (push) spremembe lokalnega imenika na oddaljeni repozitorij
   - **git push**
5. Povleci (pull) spremembe oddaljenega repozitorija v lokalni repozitorij
   - **git pull** ali kombinacija
   - **git fetch**
     git merge
6. Posodobitev datotek v delovnem imeniku, da se le ta ujemajo z verzijo v indeksu oziroma določenega drevesa
   - **git checkout**

Delovni tok z datotekami

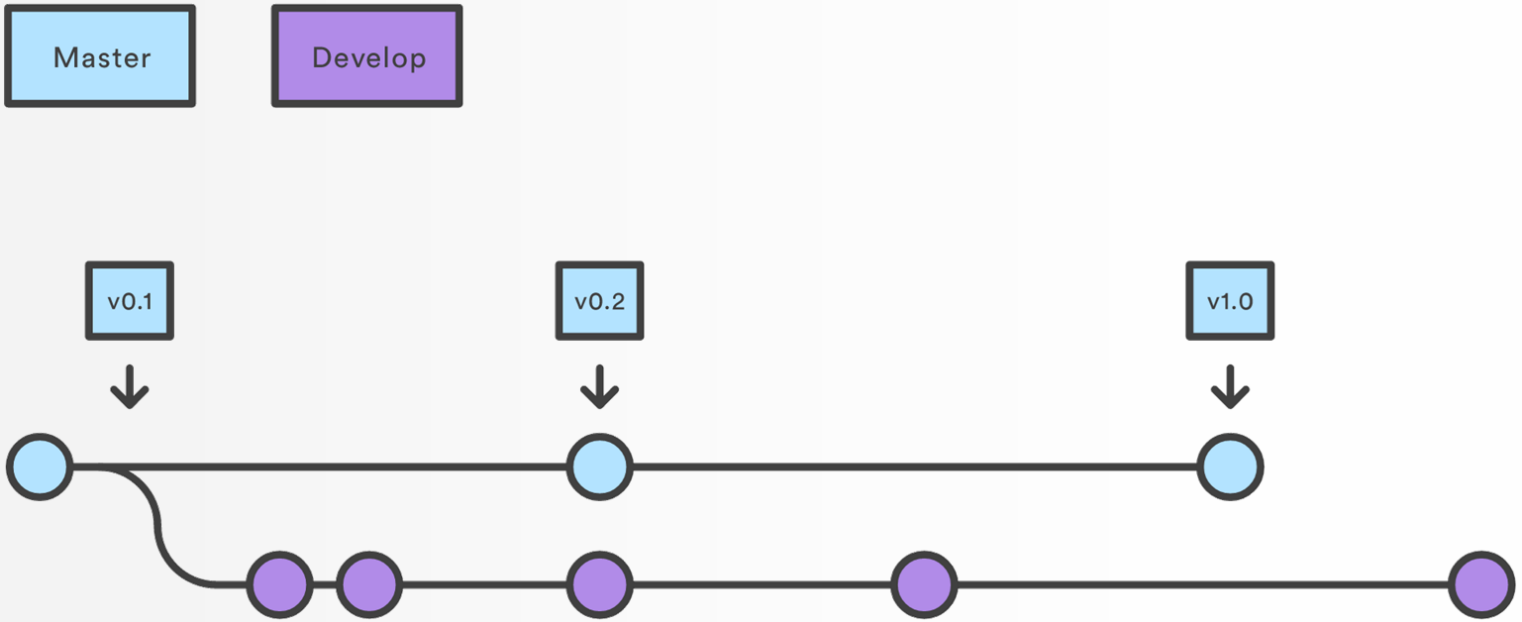


Ukazni poziv

- Poda informacije o stanju datotek veje v kateri se nahajamo
- Oblika ukaznega poziva
  - **[{HEAD-name} +A ~B -C !D | +E ~F -G !H !]**
    - {HEAD-name} - ime trenutne veje v kateri se nahajamo
      - **Modrozelena** – delujoča veja se ujema z oddaljeno vejo
      - **Zelena** – delujoča veja je pred oddaljeno vejo
      - **Rdeča** – delujoča veja je za oddaljeno vejo
      - **Rumena** – delujoča veja je tako pred in za oddaljeno vejo
    - ABCD predstavlja indeks in EFGH predstavlja delovni imenik
      - + = dodani dokumenti
      - ~ = spremenjeni dokumenti
      - - = odstranjeni dokumenti
      - ! = dokumenti v konfliktu
    - ! Pomeni da ni nesledljivih datotek

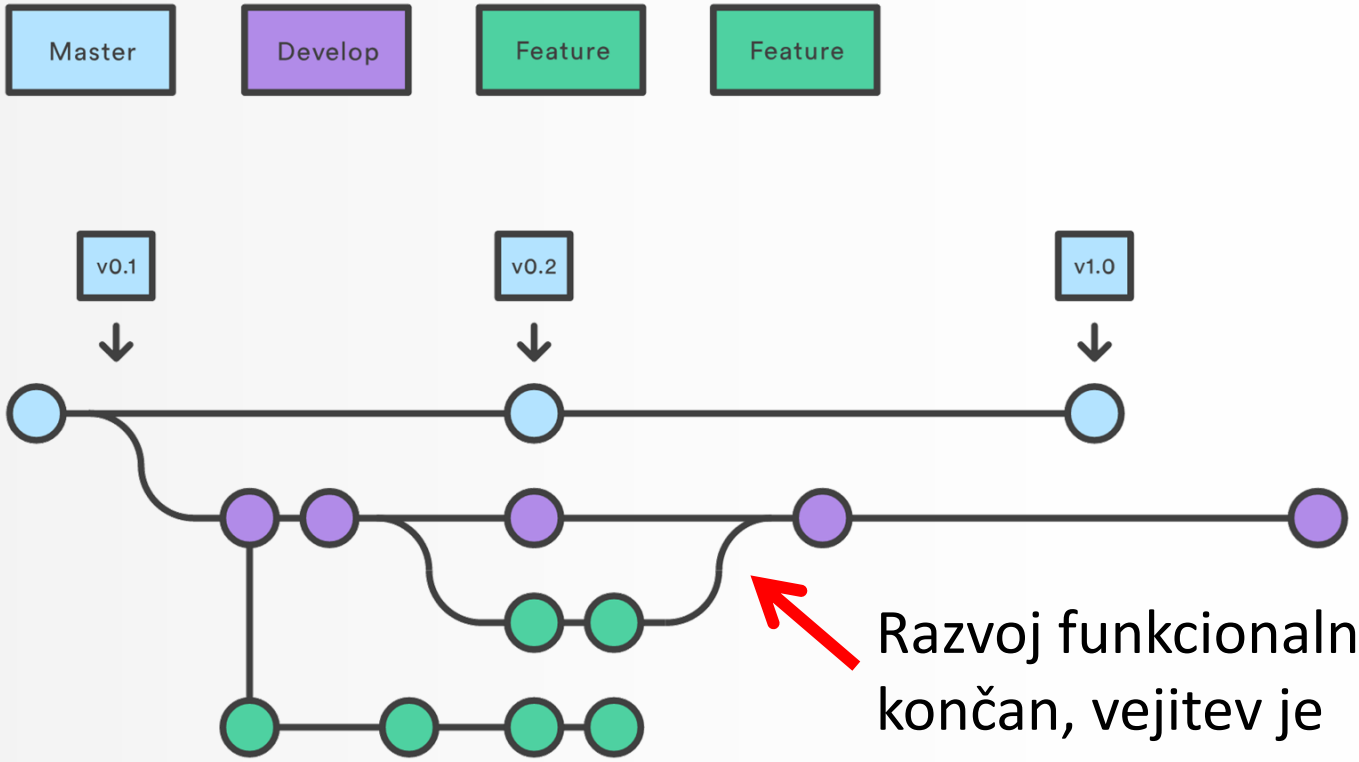
Gitflow Workflow – *Master* in *Develop*

- Enostavno gledano uporabljamo za razvoj programske opreme dve veji:
  - *Master* - veja s stabilnimi verzijami za javnost/naročnika
    - Vsaka verzija ima navadno svojo oznako (*tag*) – v0.1, v0.2, v1.0
  - *Develop* - veja, ki vsebuje razvojne posnetke



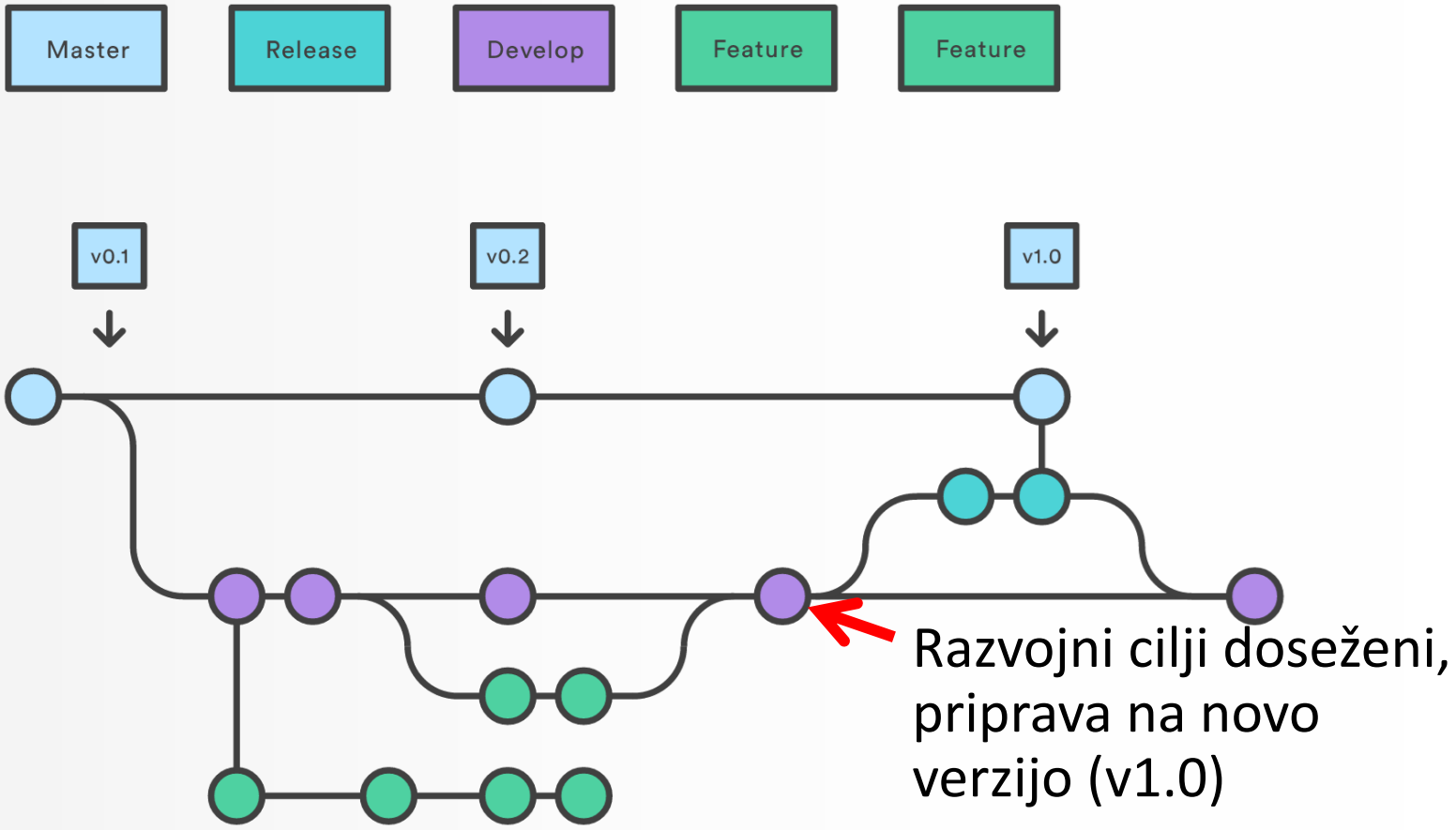
Gitflow Workflow - *Feature*

- Znotraj razvojne veje ustvarimo dodatno vejitev, za razvoj določene funkcionalnosti
- Po dokončanju funkcionalnosti, vejitev združimo z razvojno vejo
- Veje za razvoj funkcionalnosti nikoli ne vejimo ali združujemo s stabilno oz. *master* vejo



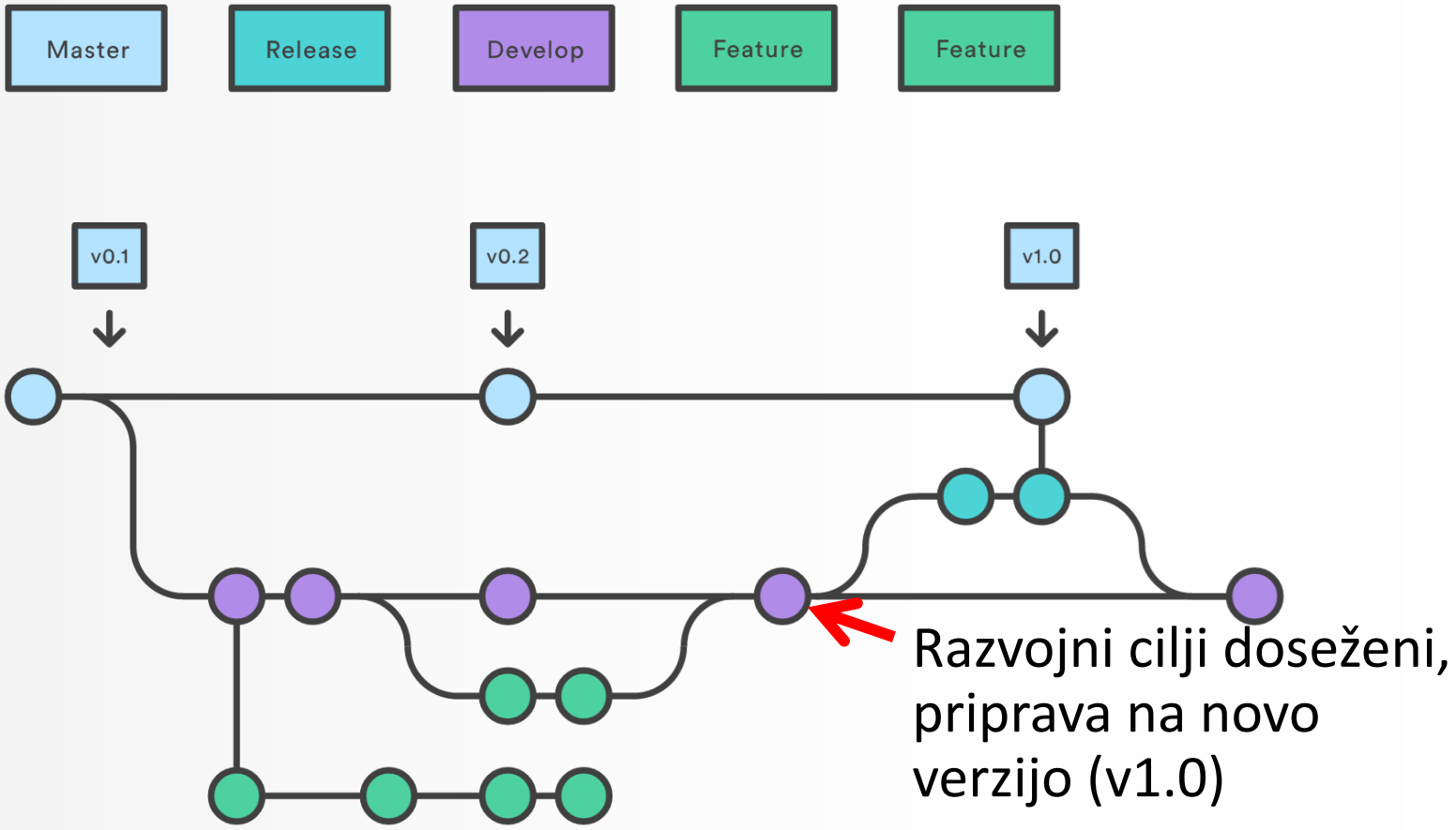
Gitflow Workflow - *Release*

- Ko v razvojni veji dosežemo zastavljene cilje, iz razvojne veje odcepimo vejo, ki vsebuje kandidate za izdajo (RC *release candidate*)
- Ta cepitev pomeni začetek novega razvojnega cikla, od te točke naprej se v kandidata za izdajo ne dodaja funkcionalnosti, vejo se dopolnjuje le s popravki in dokumentacijo



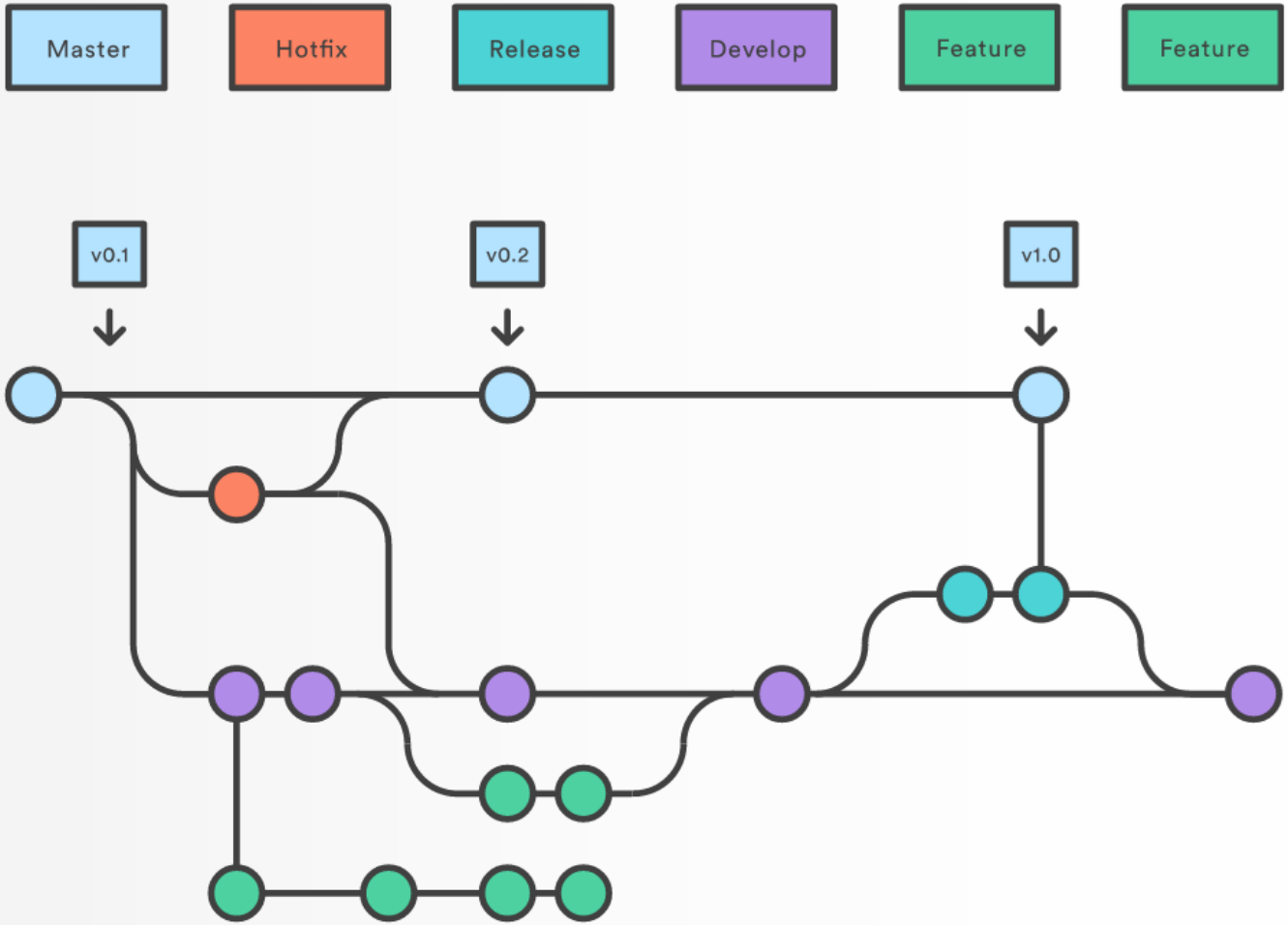
Gitflow Workflow – *Release (cnt.)*

- Ko je verzija pripravljena za javnost/naročnika jo združimo s stabilno/ *master* vejo, hkrati pa tudi z razvojno vejo.
- Uporaba ločene veje za kandidate za izdajo omogoča, da lahko razvojna ekipa med dokončevanjem podrobnosti za trenutno verzijo nemoteno razvija funkcionalnosti za naslednjo



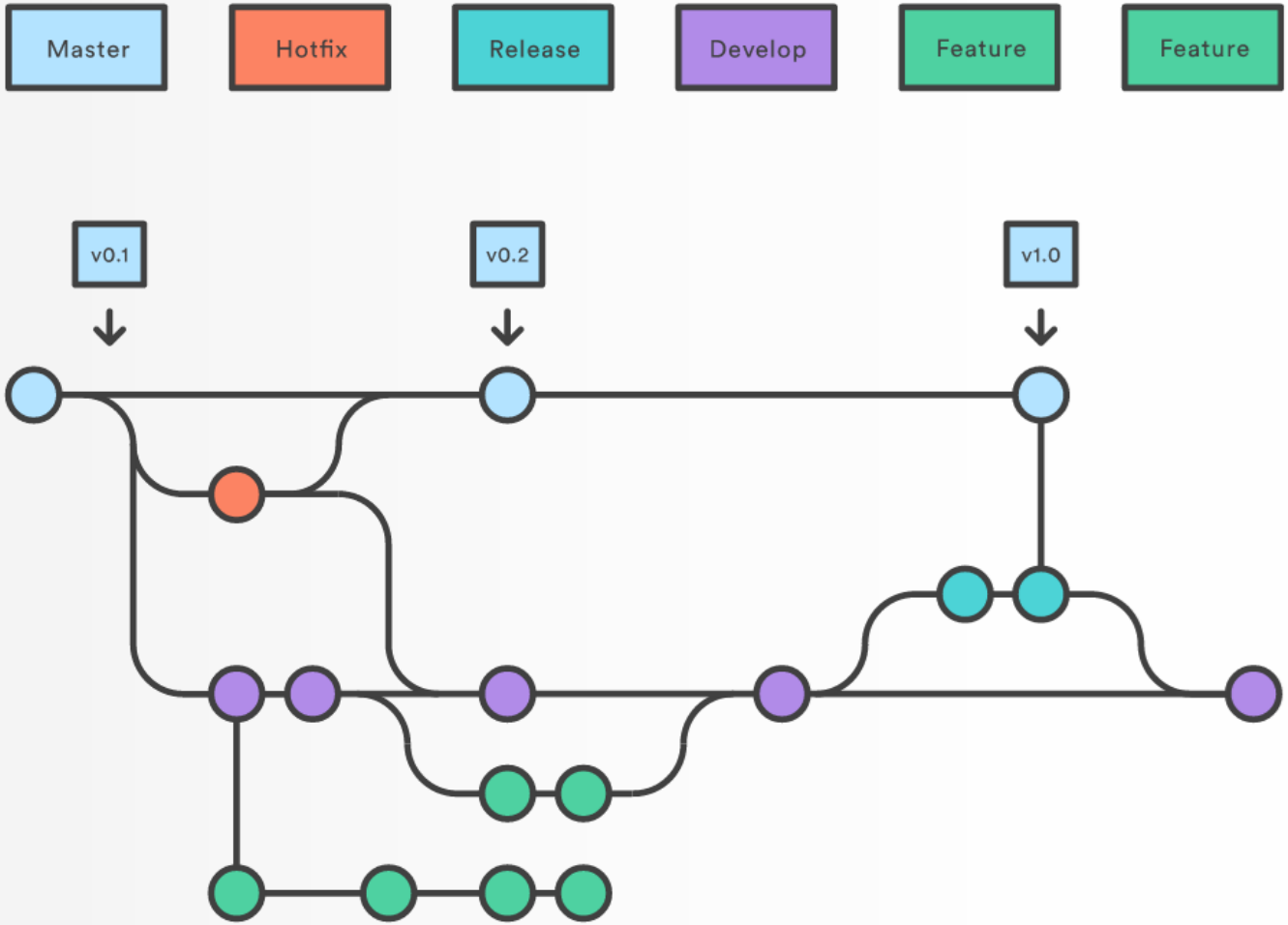
Gitflow Workflow – *Hotfix*

- Vzdrževalno ali *hotfix* vejo uporabljamo za popravke izdanih verzij. To je edina veja, ki naj bi se vejila od stabilne/*master* veje. Po končanem popravku, spremembo uveljavimo tako na stabilni/*master* veji, kot na razvojni veji

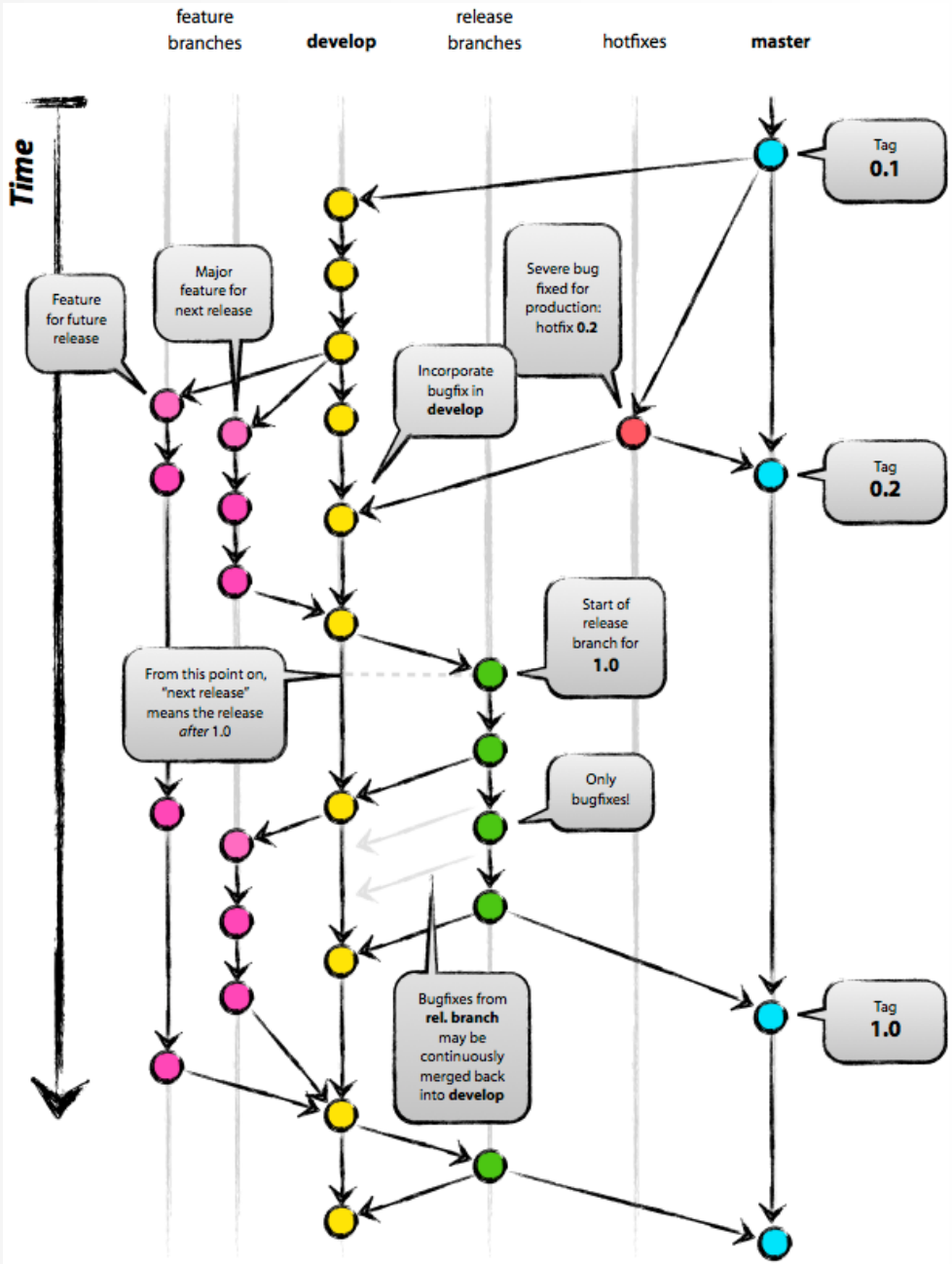


Gitflow Workflow – *Hotfix (cnt.)*

- Stabilna verzija s popravkom naj ima novo oznako (`v0.1` → `v0.2`)
- Ločena vzdrževalna ali *hotfix* veja nam omogoča, da popravljanje hroščev ne prekine ostalega razvojnega procesa



Git Workflow



Kreiranje repozitorija GIT s pomočjo dveh pristopov

1. Kreiranje novega projekta v GIT repozitorij
   - V direktorju, kjer želimo kreirati nov projekt uporabimo ukaz **git init <ime_projekta>**. Kreira se skelet Git repozitorija (.git).

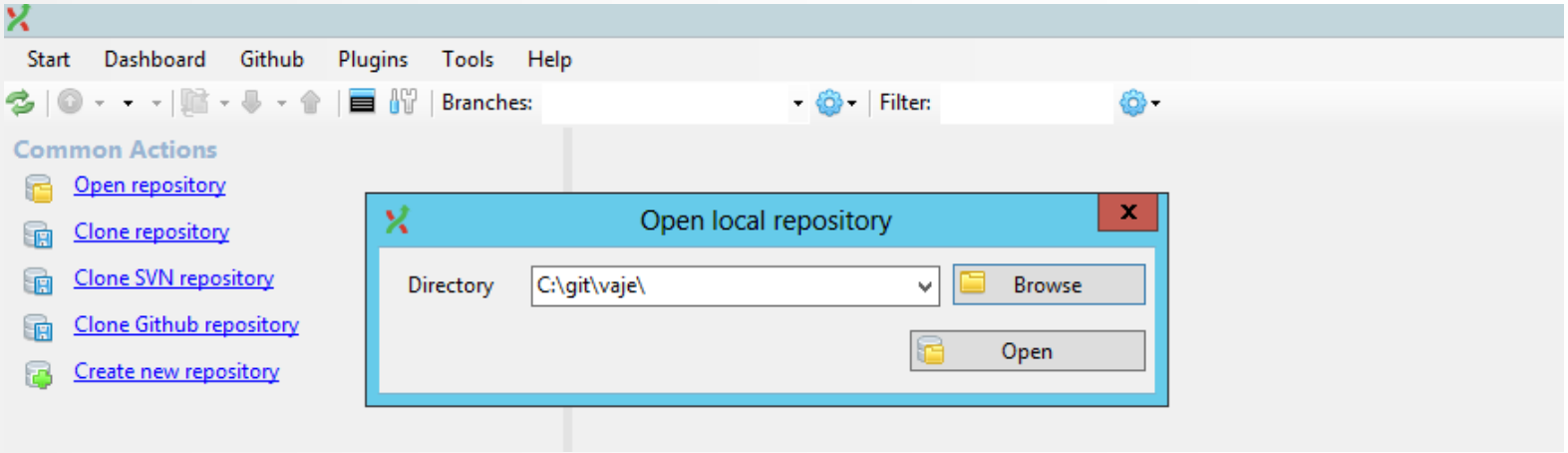
```
C:\git> git init novi_projekt
Initialized empty Git repository in C:/git/novi_projekt/.git/
C:\git> _
```

2. Kloniranje obstoječega repozitorija Git iz drugega strežnika
   - Na lokalnem disku ustvarimo kopijo obstoječega repozitorija Git
   - Uporaba ukaza **git clone <ime_repozitorija> <ime_imenika>**
   - Podoben ukazu **svn checkout**, vendar pri Git ustvarimo celotno kopijo repozitorija (celotno zgodovino sprememb) in ne samo zadnje verzije projekta kot pri SVN

```
C:\git> git clone git@gitlab.cloud.local:vaje/vaje.git vaje
Cloning into 'vaje'...
Warning: Permanently added 'gitlab.cloud.local,192.168.29.148' (RSA) to the list of known hosts.
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (3/3), done.
Checking connectivity... done
C:\git> _
```

GitExtension

- Grafični uporabniški vmesnik za nadzor Git repozitorija
- Odpiranje repozitorija



Konfiguracija repozitorija GIT

- Uporaba ukaza **git config**
- Konfigurira se lahko vse od informacij o uporabniku do preferenc obnašanja samega repozitorija
- Uporaba zastavice **--global** za nastavitev možnosti trenutnega uporabnika

- Primeru uporabe:
  - **git config --global user.name <ime_uporabnika>**
  - **git config --global user.email <email_uporabnika>**
  - **git config --global --edit**
    - Odpre globalno konfiguracijsko datoteko v tekstovnem urejevalniku z namenom ročnega urejanja

```
C:\git> git config --global user.name Andrej_Kocbek
C:\git> git config --global user.email andrej@kocbek.si
C:\git> git config --global --edit
```

Shranjevanje sprememb

- Uporaba ukaza **git add**
  - Dodajanje novih datotek
  - Doda spremembe, ki so znotraj delovnega imenika v staging območje
  - Spremembe niso vidne v repozitoriju Git , dokler ne uporabimo ukaza **git commit**

- Uporaba ukaza **git commit**
  - Potrdi se trenuten posnetek v lokalni repozitorij
  - Nismo prisiljeni do interakcije centralnega repozitorija dokler koda ni za to pripravljena
  - V primeru, da ne dodamo zastavice **–m <sporočilo>** (npr. git commit –m sporočilo), se nam odpre urejevalnik besedila, v katerega vpišemo sporočilo

```
C:\git\vaje [master]> git add Vaja2.java
C:\git\vaje [master +1 ~0 -0]> git commit -m "Dodali smo novo datoteko"
[master 1f0f43d] Dodali smo novo datoteko
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Vaja2.java
C:\git\vaje [master]>
```

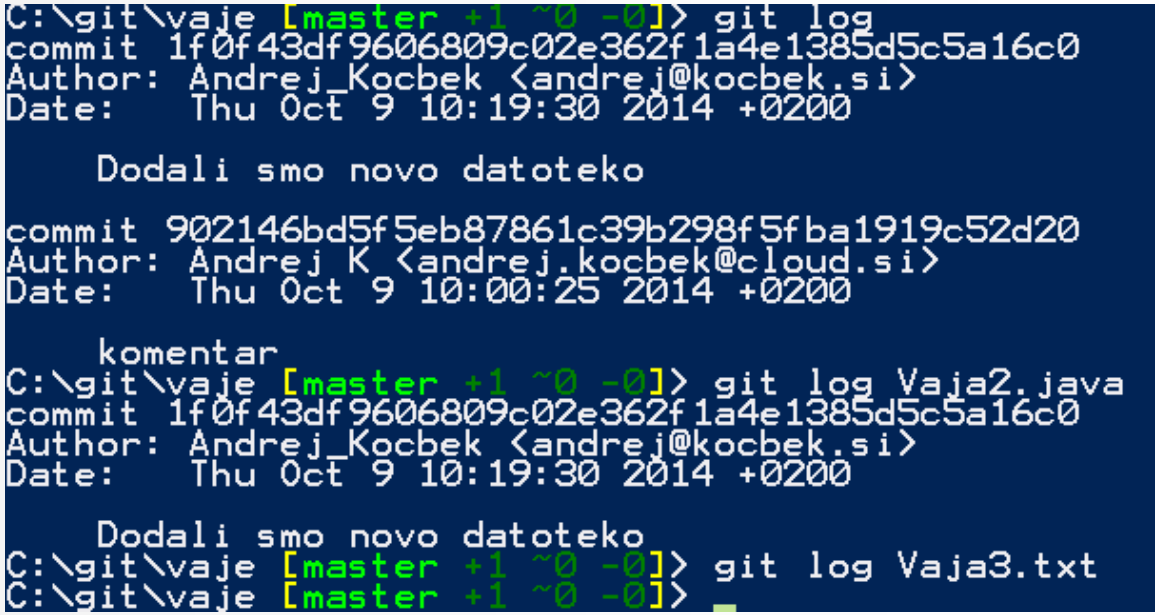
Prikaz stanja

- Uporaba ukaza **git status**
  - Prikaže stanje delovnega imenika in staging območja (npr. katere datoteke so in katere niso staged, katerim datotekam Git ne sledi, itd.).
  - Ne prikazuje informacij, ki se tičejo zgodovine potrditve projekta. V ta namen se uporablja ukaz **git log**.

```
C:\git\vaje [master]> git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
C:\git\vaje [master]> git add Vaja3.txt
C:\git\vaje [master +1 ~0 -0]> git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   Vaja3.txt
#
C:\git\vaje [master +1 ~0 -0]> _
```

Pregled zgodovine

- Uporaba ukaza **git log**
  - Prikazuje potrditve trenutnih posnetkov.
  - Omogoča prikaz seznama zgodovine projekta, iskanje po specifičnih spremembah in filtracijo trenutnih posnetkov.
- Primeri uporabe:
  - **git log -n 4**
    - prikaže zgodovino zadnjih 4 potrditev
  - **git log <ime_datoteke>**
    - Prikaže zgodovino potrditve le določene datoteke



Delo z drugimi repozitoriji

- Uporaba ukaza **git remote**
  - Omogoča kreiranje, pogled in odstranjevanje povezav do drugih repozitorijev

- Primer uporabe
  - **git remote**
    - Prikaže vse oddaljene povezave do repozitorijev
  - **git remote add <ime> <url>**
    - Ustvari novo povezavo do oddaljenega repozitorija
  - **git remote rm <ime>**
    - Odstrani povezavo do oddaljenega repozitorija

```
C:\git\vaje [master +1 ~0 -0]> git remote
origin
C:\git\vaje [master +1 ~0 -0]> git branch -r
  origin/HEAD -> origin/master
  origin/master
C:\git\vaje [master +1 ~0 -0]> git remote add vaje2 git@gitlab.cloud.local:vaje/vaje2.git
C:\git\vaje [master +1 ~0 -0]> git remote
origin
vaje2
C:\git\vaje [master +1 ~0 -0]> _
```

Uvoz sprememb

- Uporaba ukaza **git fetch**
- Omogoča uvoz potrditev oddaljenega repozitorija v naš lokalni repozitorij
- S tem je omogočen pregled sprememb preden jih integriramo v našo kopijo projekta

- Primer uporabe
  - **git fetch remote**
    - Uvozi vse spremembe repozitorija. Dodano vključi vse zahtevane sprememb iz drugih odvisnih repozitorijev
  - **git fetch remote branch**
    - Podobno kot zgornji ukaz, le da v tem primeru uvozimo specifično podvejo projekta

```
C:\git\vaje [master +1 ~0 -0]> git fetch vaje2
Warning: Permanently added 'gitlab.cloud.local,192.168.29.148' (RSA) to the list of known hosts.
```

Povleci spremembe

- Uporaba ukaza **git pull**
  - Združitev sprememb v lokalni repozitorij
  - Združuje ukaza **git fetch** in **git merge** v en sam ukaz
  - Podoben je ukazu **svn update**.

- Primer uporabe
  - **git pull <remote>**
    - Povleče vse spremembe podveje oddaljenega repozitorija in jih takoj združi z lokalno kopijo projekta.
    - Primer uporabe git fetch in git merge bi bil:
      - **git fetch <remote>**
      - **git merge origin/<current-branch>**

```
C:\git\vaje [master]> git pull
Warning: Permanently added 'gitlab.cloud.local,192.168.29.148' (RSA) to the list of known hosts.
Already up-to-date.
C:\git\vaje [master]> git pull
Warning: Permanently added 'gitlab.cloud.local,192.168.29.148' (RSA) to the list of known hosts.
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (2/2), done.
From gitlab.cloud.local:vaje/vaje
   2e8e34e..7ce1296  master     -> origin/master
Updating 2e8e34e..7ce1296
Fast-forward
 Vaja4.java | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Vaja4.java
C:\git\vaje [master]>
```

Potisni spremembe

- Uporaba ukaza **git push**
  - Prenos potrditev lokalnega repozitorija v oddaljeni repozitorij
  - Je nasprotni ukaz od **git fetch**

Primer uporabe

git push remote branch

- Potisnemo specifično podvejo projekta na oddaljeni repozitorij. Na oddaljenem repozitoriju se ustvari lokalna podveja s spremembami. Če je rezultat potiska na oddaljenem repozitoriju non-fast-forward združitev, nam Git ne bo dovolil izvoza.
- Da zaobidemo non-fast-forward združitev, uporabimo zastavico **--force.**

git push <remote> --all

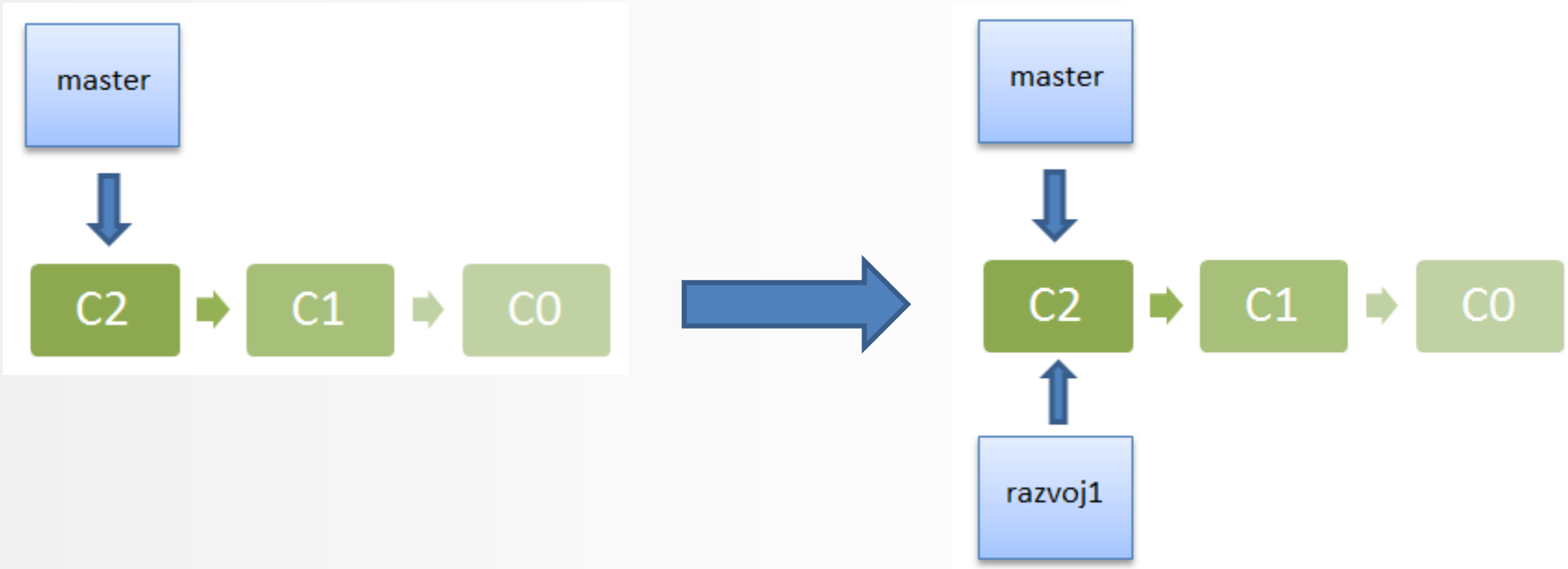
- Na oddaljeni repozitorij potisnemo vse lokalne podveje

```
C:\git\vaje [master]> git status
# On branch master
# Your branch is ahead of 'origin/master' by 2 commits.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
C:\git\vaje [master]> git push
Warning: Permanently added 'gitlab.cloud.local,192.168.29.148' (RSA) to the list of known hosts.
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 460 bytes | 0 bytes/s, done.
Total 5 (delta 1), reused 0 (delta 0)
To git@gitlab.cloud.local:vaje/vaje.git
   902146b..2e8e34e  master -> master
C:\git\vaje [master]> git status
# On branch master
nothing to commit, working directory clean
C:\git\vaje [master]>
```

Vejitve in združitve

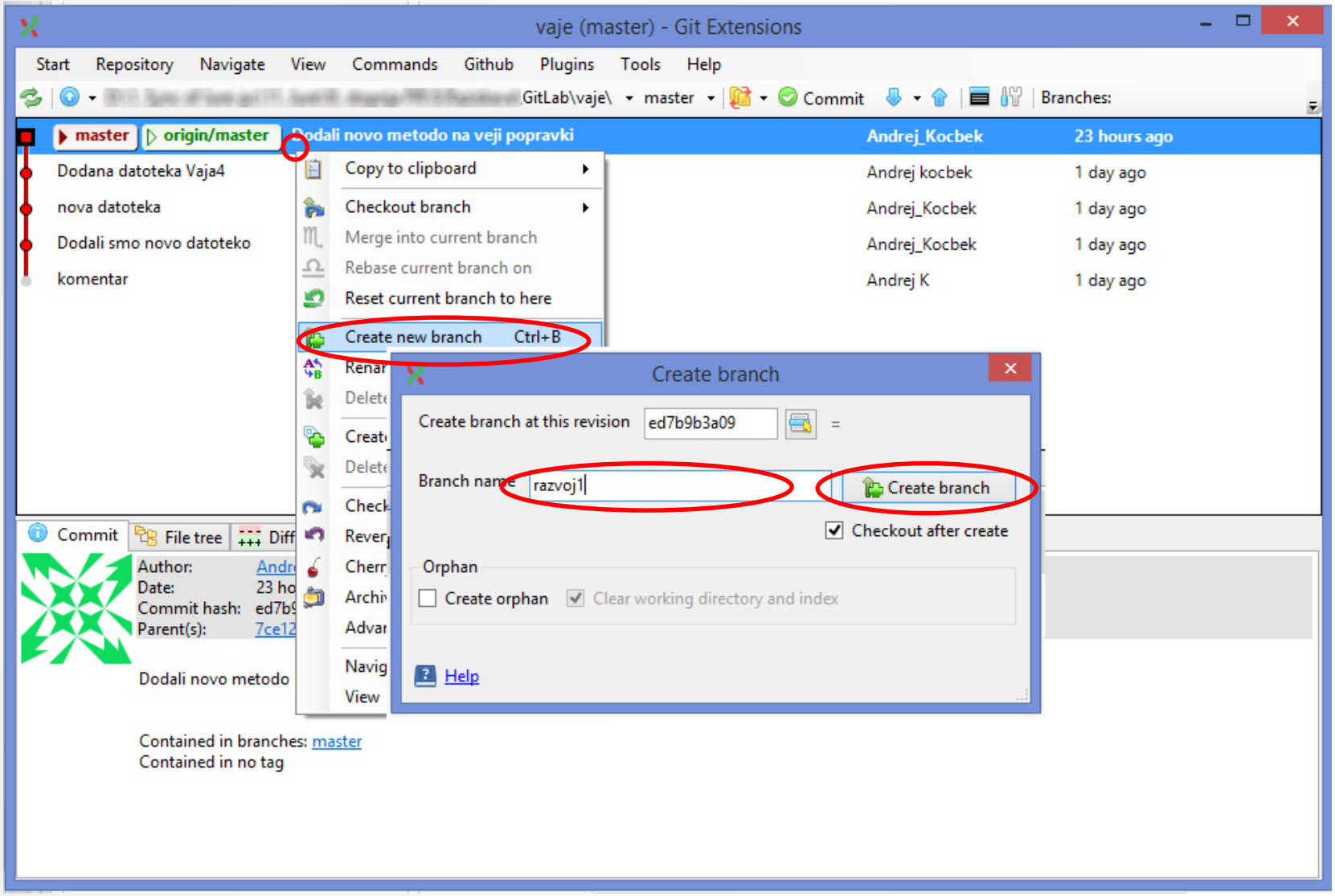
- Uporaba ukaza **git branch <ime_veje>**
  - Ustvari novo vejo razvoja
  - Prehod na novo vejo izvedemo z ukazom **git checkout <ime_veje>**
  - Uporaba zastavice –b (switch) nam hkrati ustvari in izvede prehod na novo vejo

```
C:\git\vaje [master]> git branch razvoj1
C:\git\vaje [master]> git checkout razvoj1
Switched to branch 'razvoj1'
C:\git\vaje [razvoj1]> git checkout -b razvoj2
Switched to a new branch 'razvoj2'
C:\git\vaje [razvoj2]>
```



Vejitve in združitve

- Kreiranje veje v Git Extensions, ki omogoča GUI
  - Desni klik na želen posnetek → Create new branch

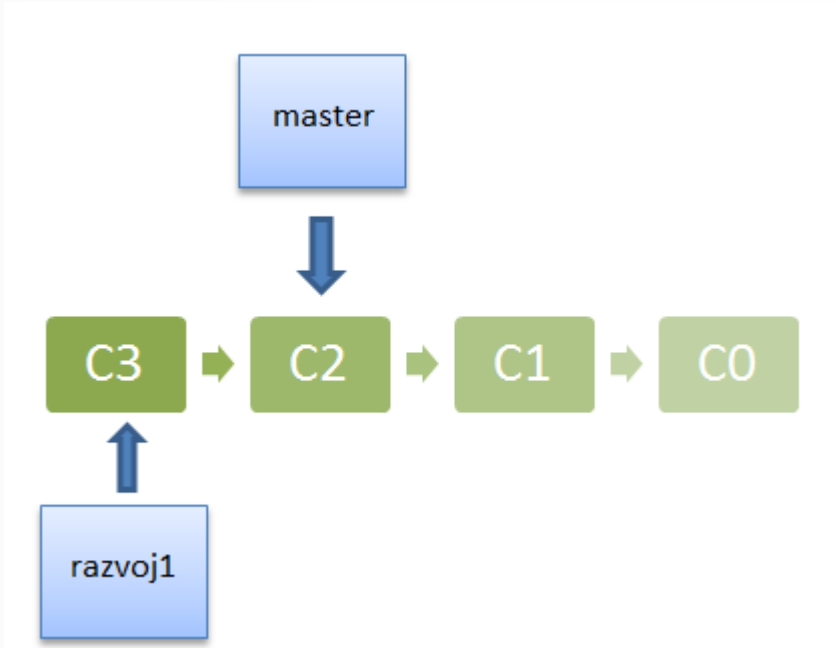


Vejitve in združitve

- Spremenimo datoteko Vaja2.java
- Izvedemo potrditev sprememb

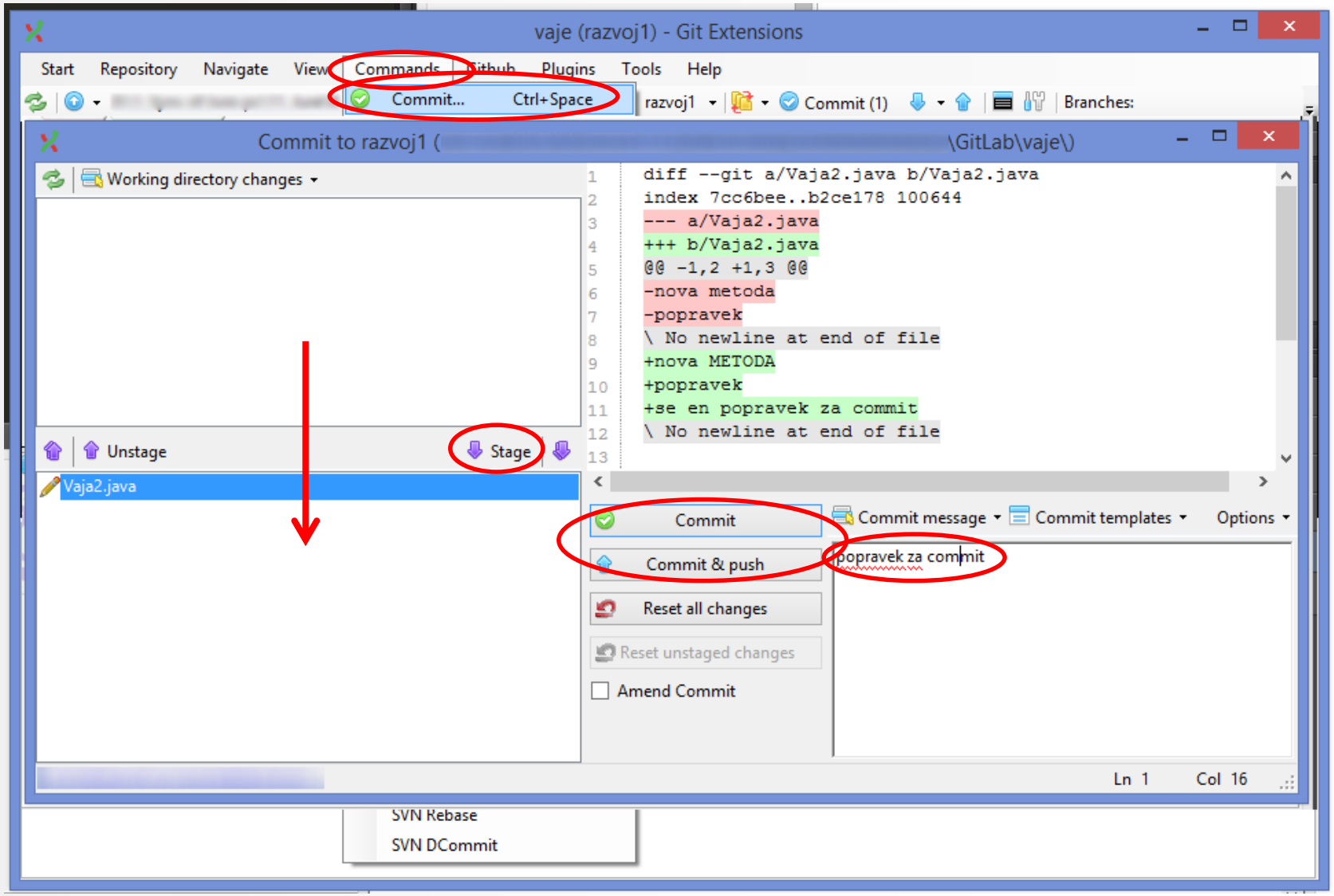
```
C:\git\vaje [razvoj1]> git commit -a -m "Dodali novo metodo na veji razvoj1"
[razvoj1 99c2684] Dodali novo metodo na veji razvoj1
 1 file changed, 3 insertions(+)
```

- V primeru, da želimo opraviti popravke na „master“ veji, enostavno izvedemo prehod na vejo (**git checkout master**) in nadaljujemo z razvojem.
  Spremembe, ki so bile opravljene znotraj razvoj1, tukaj niso vidne.



Vejitve in združitve

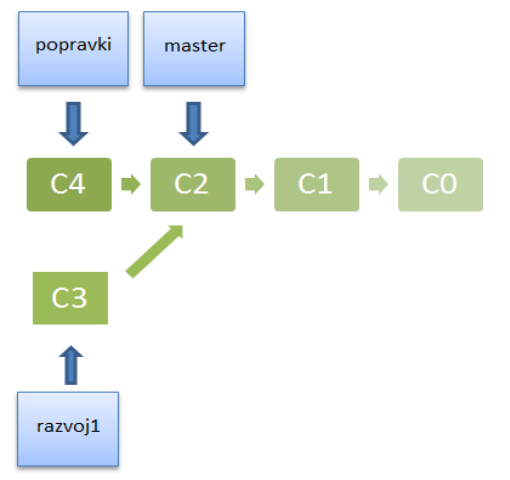
- Potrditev sprememb
  - Commands → Commit…

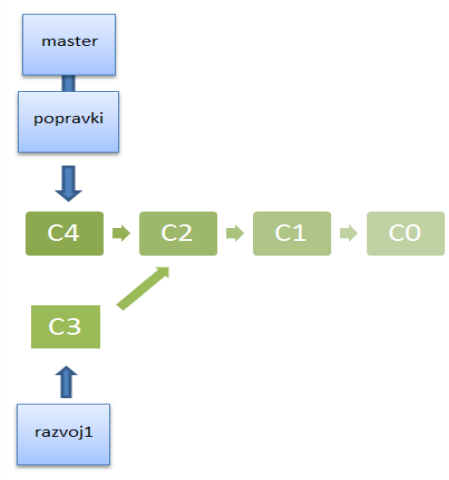


Vejitve in združitve

- V nadaljevanju naredimo novo vejo „popravki“ in ponovno spremenimo datoteko „Vaja2.java“ z drugimi popravki.
- Vsaka veja kaže na svojo potrditev.
- V primeru, da želimo združiti veje, uporabimo ukaz **git merge**

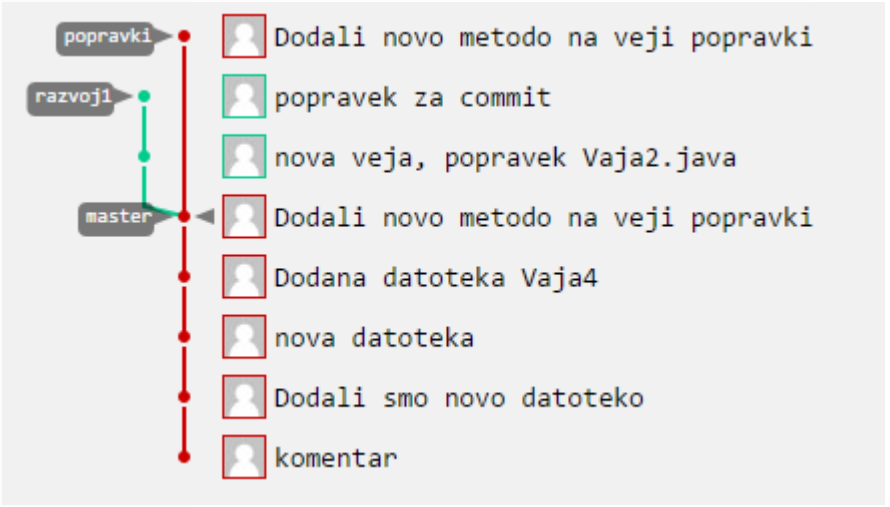
```
C:\git\vaje [master]> git checkout -b popravki
Switched to a new branch 'popravki'
C:\git\vaje [popravki]> git commit -a -m "Dodali novo metodo na veji popravki"
[popravki ed7b9b3] Dodali novo metodo na veji popravki
 1 file changed, 1 insertion(+)
C:\git\vaje [popravki]> git checkout master
Switched to branch 'master'
C:\git\vaje [master]> git merge popravki
Updating 7ce1296..ed7b9b3
Fast-forward
 Vaja2.java | 1 +
 1 file changed, 1 insertion(+)
C:\git\vaje [master]>
```



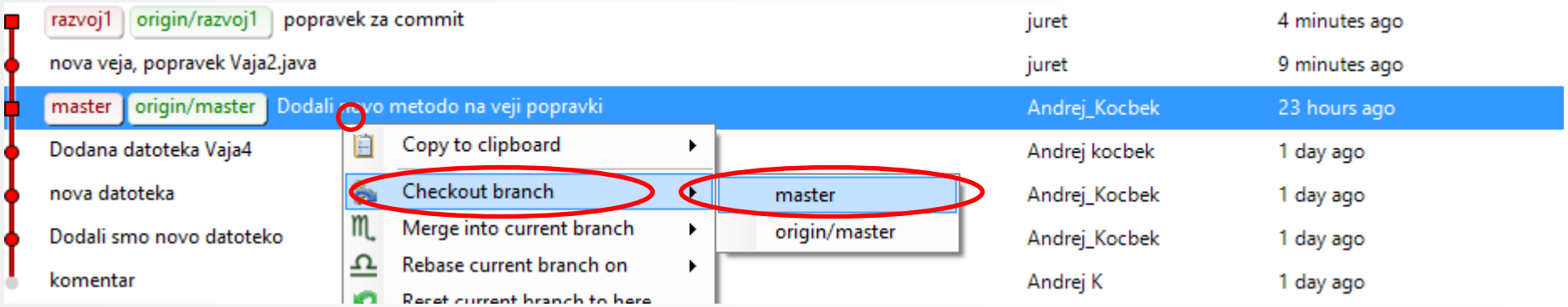


Vejitve in združitve

- Združitev vej
  - Trenutno stanje razvoja:
    - 3 veje (master, razvoj1, popravki)
  - Želja:
    - Prenesti popravke iz veje „popravki“ v master

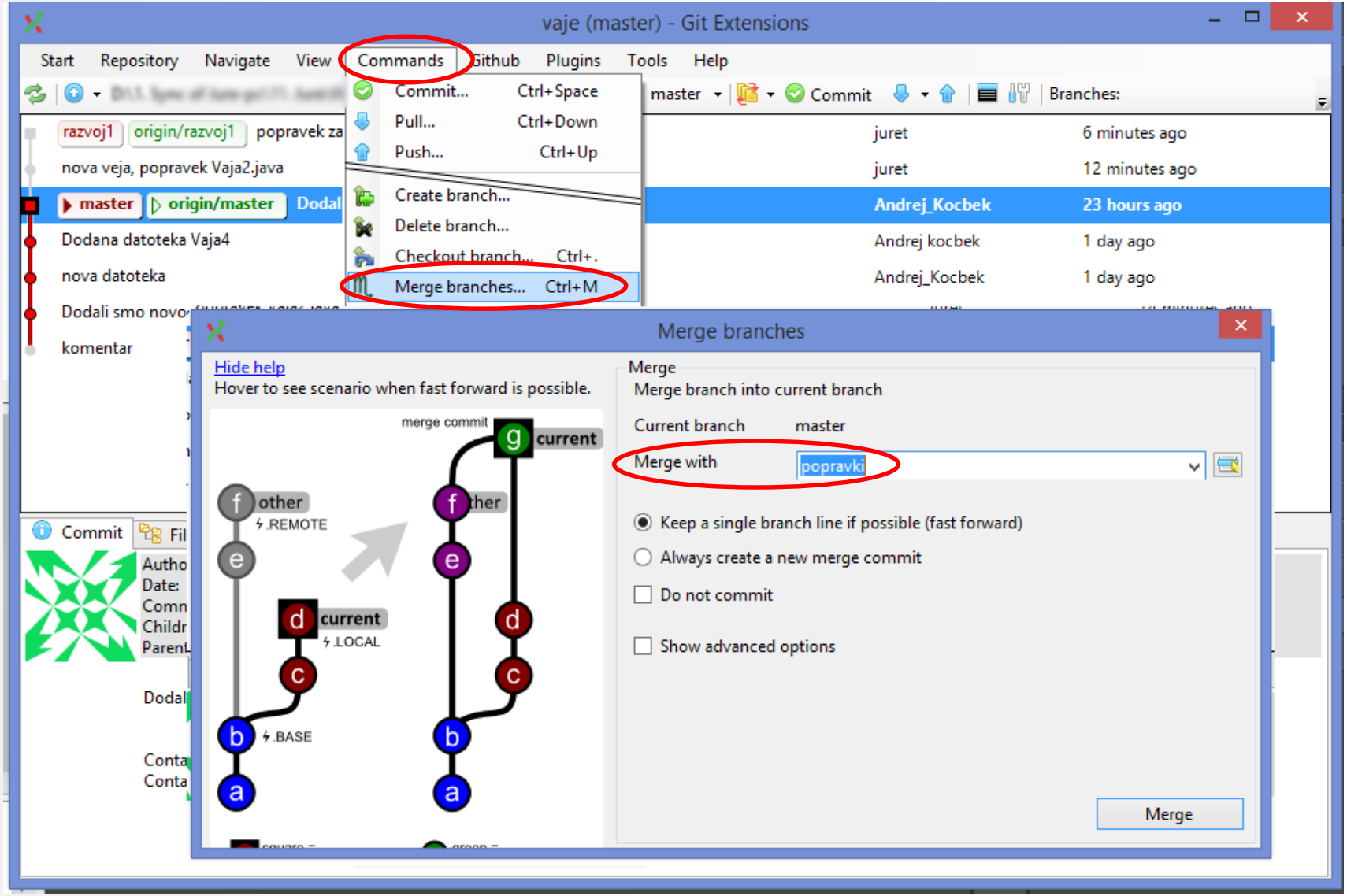


- Zamenjava veje:
  - Premaknemo se v vejo master



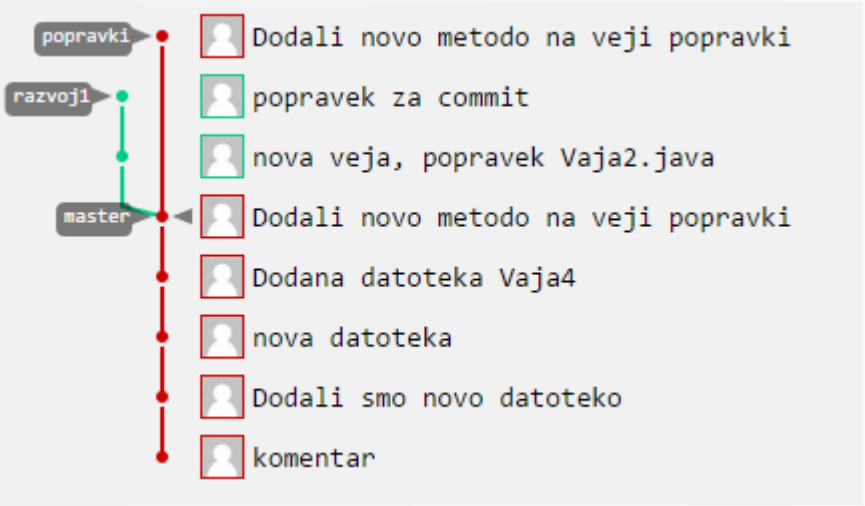
Vejitve in združitve

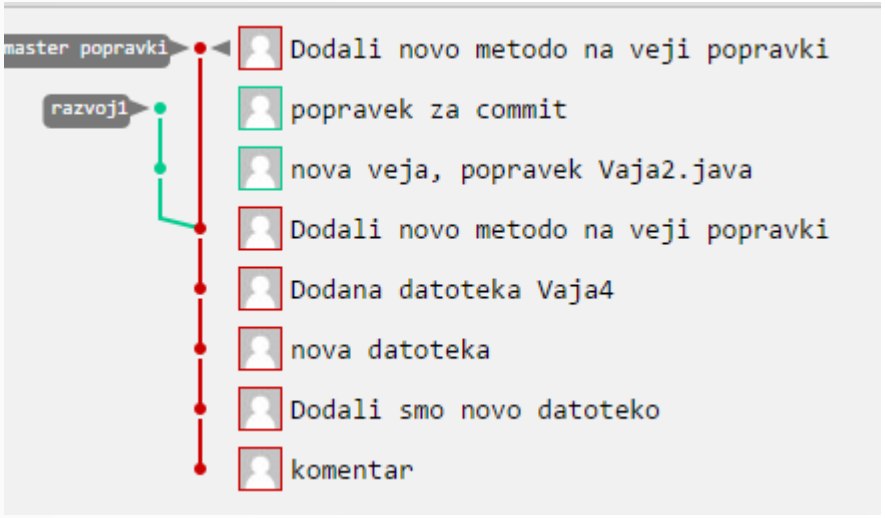
- Združitev vej
  - Commands → Merge branches…



Vejitve in združitve

- Združitev vej
  - Stanje pred združitvijo popravkov iz „popravki“ v master
  - Stanje po združevanju

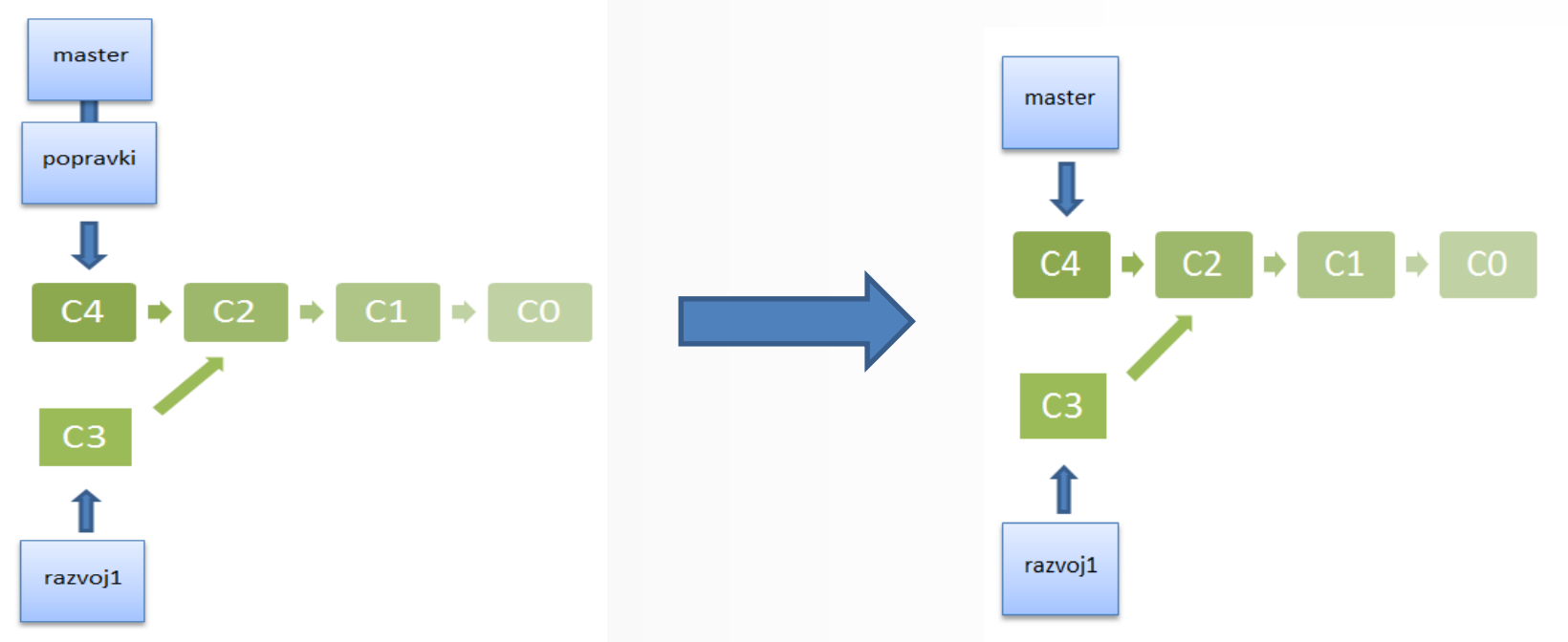




Vejitve in združitve

- Po združitvi se veji „master“ in „popravki“ sklicujeta na isto potrditev -> veja „popravki“ ni več potrebna in se lahko pobriše
- Uporaba zastavice **–d**, ki predstavlja delete

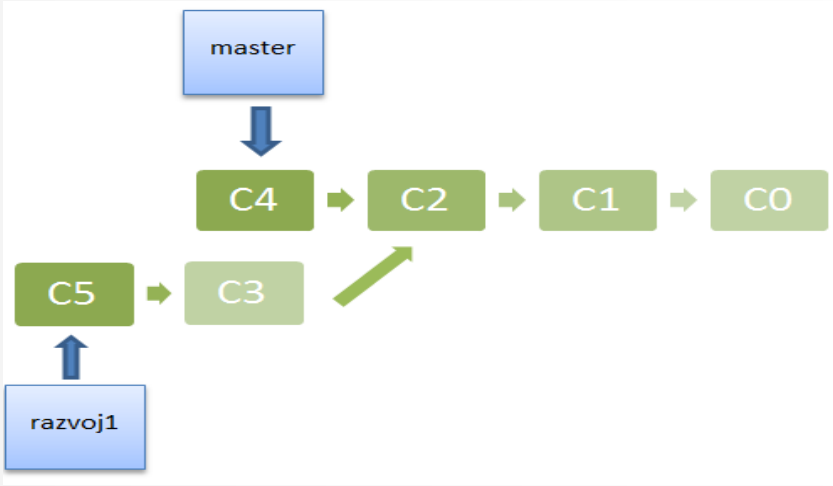
```
C:\git\vaje [master]> git branch -d popravki
Deleted branch popravki (was ed7b9b3).
C:\git\vaje [master]>
```



Vejitve in združitve

- V primeru, da nadaljujemo z razvojem na veji „razvoj1“, se ustvari nova potrditev znotraj te veje.

```
C:\git\vaje [master]> git checkout razvoj1
Switched to branch 'razvoj1'
C:\git\vaje [razvoj1]> git commit -a -m "dopolnitev metode na razvoj1"
[razvoj1 bcc44eb] dopolnitev metode na razvoj1
 1 file changed, 1 insertion(+), 1 deletion(-)
C:\git\vaje [razvoj1]>
```



- V primeru, da želimo spremembe, ki so bile narejene pri veji „popravki“, uporabiti znotraj veje „razvoj1“, izvedemo združitev veje „master“ (vsebuje popravke veje „popravki“, ki smo jo nato izbrisali)

Vejitve in združitve - konflikti

- V primeru, da želimo združiti datoteke, katerim smo spremenili isti del različno, pride do konflikta.

```
C:\git\vaje [razvoj1]> git merge master
Auto-merging Vaja2.java
CONFLICT (content): Merge conflict in Vaja2.java
Automatic merge failed; fix conflicts and then commit the result.
C:\git\vaje [razvoj1 +0 ~0 -0 !1 | +0 ~0 -0 !1]>
```

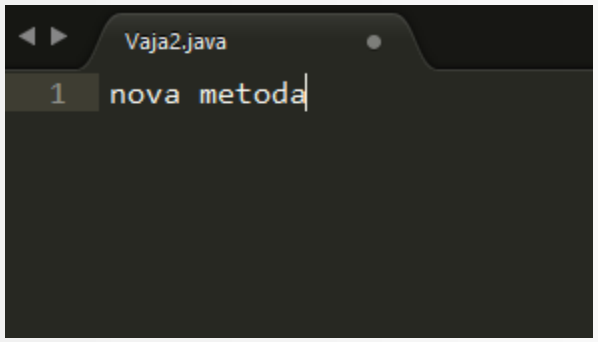
- Za ugotovitev podrobnosti konflikta uporabimo ukaz **git status**

```
C:\git\vaje [razvoj1 +0 ~0 -0 !1 | +0 ~0 -0 !1]> git status
# On branch razvoj1
# You have unmerged paths.
#   (fix conflicts and run "git commit")
#
# Unmerged paths:
#   (use "git add <file>..." to mark resolution)
#
#       both modified:      Vaja2.java
#
no changes added to commit (use "git add" and/or "git commit -a")
C:\git\vaje [razvoj1 +0 ~0 -0 !1 | +0 ~0 -0 !1]>
```

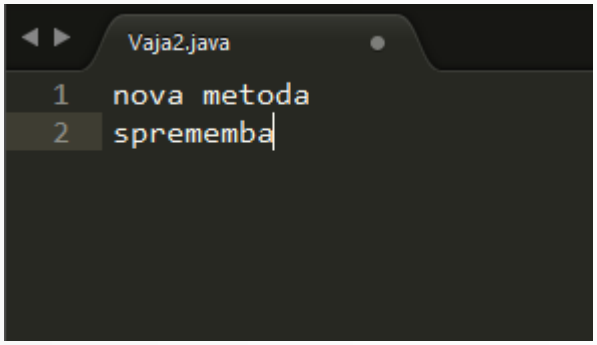
Vejitve in združitve - konflikti

- Nastanek konfliktov - primeri
  - Obe veji sta dodali vrstice v dokument:

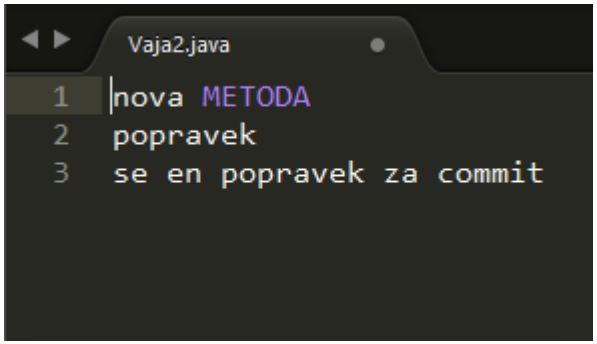
Stanje pred vejitvijo



Veja: master

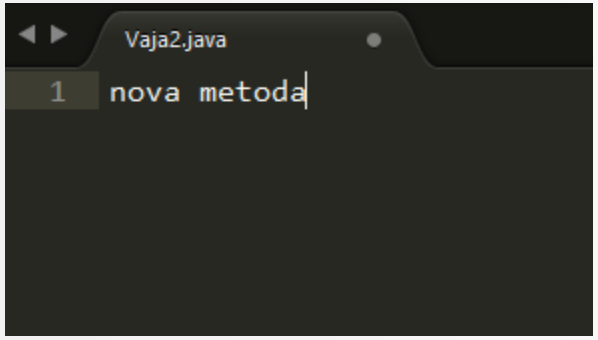


Veja: razvoj1

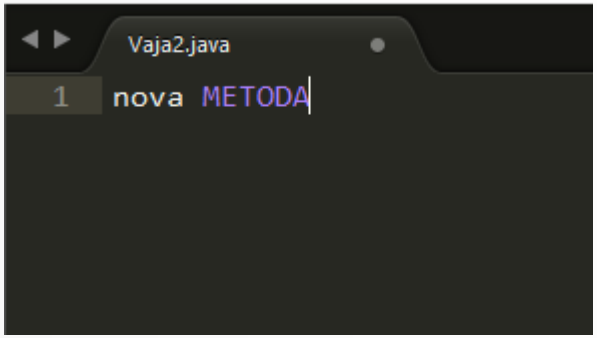


  - Spremembe v isti vrstici:

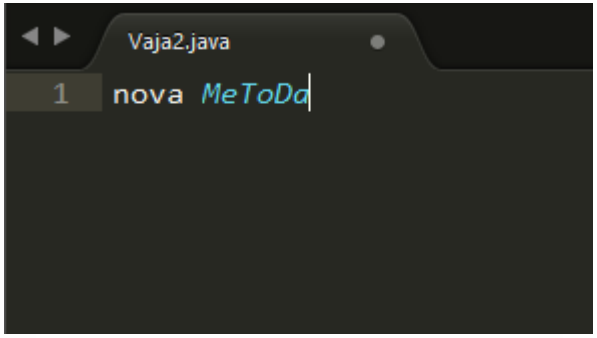
Stanje pred vejitvijo



Veja: master

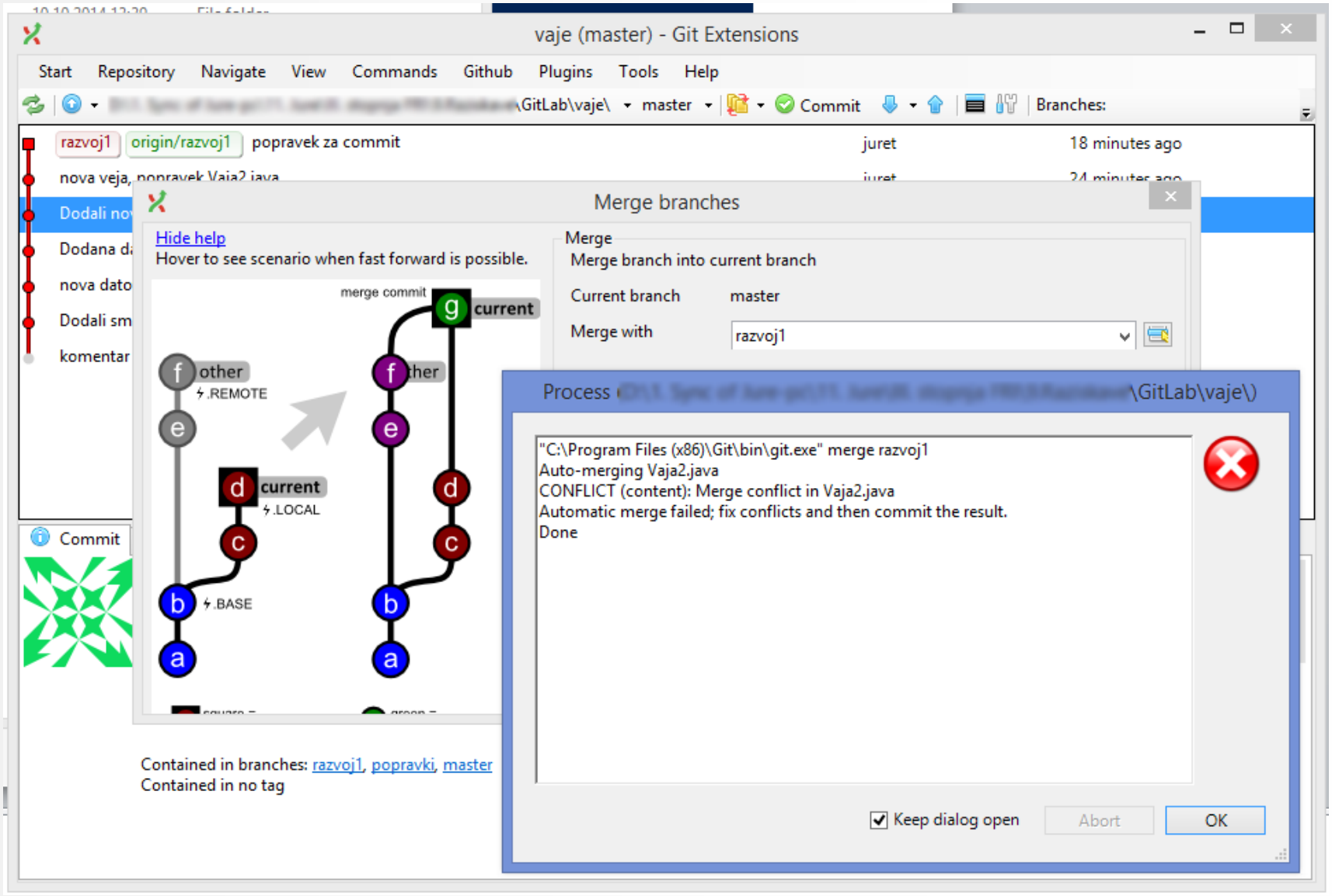


Veja: razvoj1



Vejitve in združitve - konflikti

- Obvestilo o konfliktu



Vejitve in združitve - konflikti

- Git v datoteke, ki jih ne uspemo zaradi konfliktov združit, doda standardne konflikt-rešitev označbe (angl. conflict-resolution markers)
- Označbe pomagajo pri ročni odpravi napak
- Simboli označb:
  - **<<< HEAD** – predstavlja datoteko veje, v kateri se nahajamo (v našem primeru „razvoj1“ veja)
  - **====** – ločnica, loči med vsebinami datoteke posamezne veje
  - **>>> ime_veje** master – predstavlja vsebino datoteke, ki jo želimo združit (v našem primeru „master“ veja)

```
<<<<<<< HEAD
public class Vaja2{
  //dopolnitev metode
}
=======
nova metoda
>>>>>>> master
```

Vejitve in združitve - konflikti

- Za razrešitev konfliktov je potrebno izbrati eno izmed strani ločnice (vsebino kode posamezne veje) ali združimo kodo ročno.
- Odstranimo standardne označbe

```
public class Vaja2{
 //dopolnitev metode

 // nova metoda
}
```

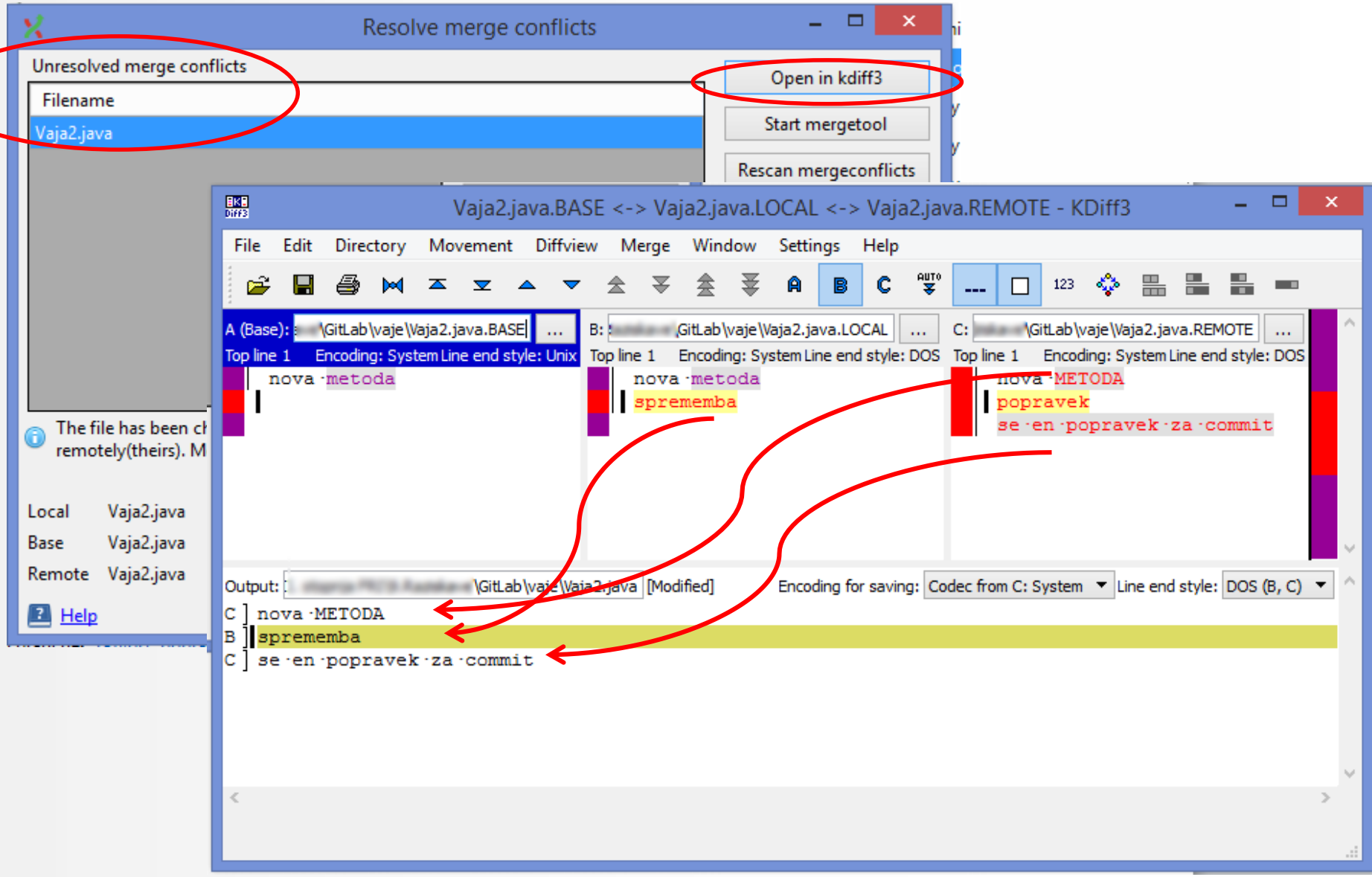
- Izvedemo ukaz **git add** za vsako datoteko, da jo označimo, da smo razrešili vse konflikte
- Sledi ukaz **git commit** in vse spremembe so shranjene v lokalni repozitorij

```
C:\git\vaje [razvoj1 +0 ~0 -0 !1 | +0 ~0 -0 !1]> git add Vaja2.java
C:\git\vaje [razvoj1 +0 ~1 -0]> git mergetool
No files need merging
C:\git\vaje [razvoj1 +0 ~1 -0]> git status
# On branch razvoj1
# All conflicts fixed but you are still merging.
#   (use "git commit" to conclude merge)
#
# Changes to be committed:
#
#       modified:   Vaja2.java
#
C:\git\vaje [razvoj1 +0 ~1 -0]> git commit -m "združili smo vsebino datoteke"
[razvoj1 53defe8] združili smo vsebino datoteke
```

- Za grafični prikaz konfliktov lahko uporabimo ukaz **git mergetool**, ki nam odpre vizualno orodje in nas vodi korak za korak skozi konflikte

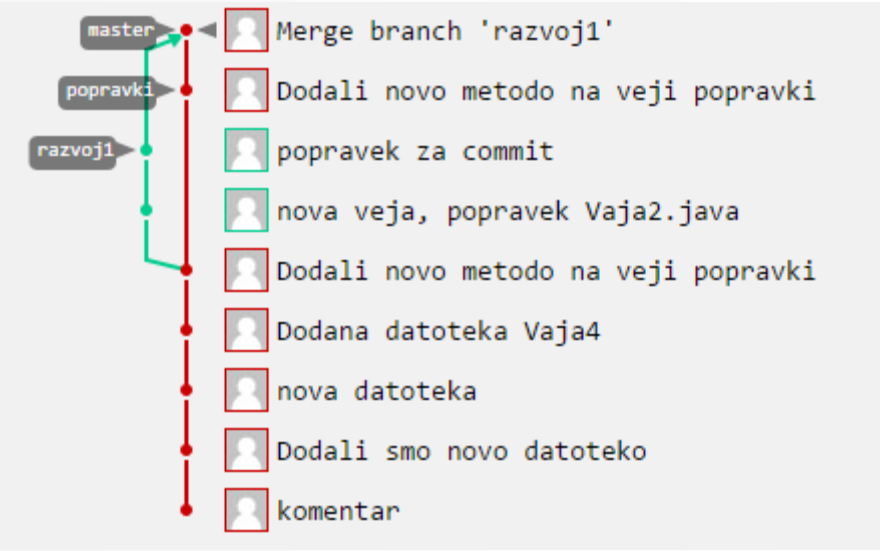
Vejitve in združitve - konflikti

- Razreševanje konfliktov



Vejitve in združitve - konflikti

- Stanje po združitvi:
  - „razvoj1“ je bil združen z master
  - V kolikor „razvoj1“ ne potrebujemo več ga lahko brišemo



Pregled sprememb

- Uporaba ukaza **git diff**
- Prikazuje razliko med HEAD in trenutnim stanjem projekta

```
C:\git\vaje [razvoj1]> git diff
diff --git a/Vaja2.java b/Vaja2.java
index 631600f..9780370 100644
--- a/Vaja2.java
+++ b/Vaja2.java
@@ -4,5 +4,6 @@ public class Vaja2{

  // nova metoda

+ // nova metoda 3
 }

C:\git\vaje [razvoj1 +0 ~1 -0]>
```

- Pri uporabi zastavice **--cached** primerja dodane datoteke z HEAD, v nasprotnem primeru primerja datoteke, ki še niso dodane (added)

```
C:\git\vaje [razvoj1 +0 ~1 -0]> git diff --cached
C:\git\vaje [razvoj1 +0 ~1 -0]> git add Vaja2.java
C:\git\vaje [razvoj1 +0 ~1 -0]> git diff --cached
diff --git a/Vaja2.java b/Vaja2.java
index 631600f..9780370 100644
--- a/Vaja2.java
+++ b/Vaja2.java
@@ -4,5 +4,6 @@ public class Vaja2{

  // nova metoda

+ // nova metoda 3
 }
```

Premik datotek

- Uporaba ukaza **git mv**
- Premakne datoteko, direktorji ali simbolno povezavo (symlink)
- **git mv <vir> <destinacija>**

```
C:\git\vaje [razvoj1]> dir


    Directory: C:\git\vaje


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----         10.10.2014      8:33            storitve
-a---          9.10.2014     10:08          0 Vaja.java
-a---         10.10.2014      8:17         92 Vaja2.java
-a---          9.10.2014     10:26          7 Vaja3.txt
-a---          9.10.2014     11:13          0 Vaja4.java

C:\git\vaje [razvoj1]> git mv .\Vaja2.java storitve
C:\git\vaje [razvoj1 +0 ~1 -0]> dir


    Directory: C:\git\vaje


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----         10.10.2014      8:34            storitve
-a---          9.10.2014     10:08          0 Vaja.java
-a---          9.10.2014     10:26          7 Vaja3.txt
-a---          9.10.2014     11:13          0 Vaja4.java
```

Odstranjevanje

- Uporaba ukaza **git rm**
- Odstrani datoteko iz delovnega drevesa in indeksa

```
C:\git\vaje [razvoj1 +0 ~1 -0]> dir


    Directory: C:\git\vaje


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----          10.10.2014      8:34            storitve
-a---           9.10.2014     10:08          0 Vaja.java
-a---           9.10.2014     10:26          7 Vaja3.txt
-a---           9.10.2014     11:13          0 Vaja4.java


C:\git\vaje [razvoj1 +0 ~1 -0]> git rm .\Vaja4.java
rm 'Vaja4.java'
C:\git\vaje [razvoj1 +0 ~1 -1]> dir


    Directory: C:\git\vaje


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----          10.10.2014      8:34            storitve
-a---           9.10.2014     10:08          0 Vaja.java
-a---           9.10.2014     10:26          7 Vaja3.txt
```

Oznake

- Zakaj potrebujemo oznake (tag-e)?
  - Arhiviranje izdanih verzij
  - Izdelava posnetka trenutnega stanja (snapshot)

- Tipično poimenovanje tag-ov:
  - Release-1.0.0
  - REL-0.3.0RC1
  - v1.2

- Naziv tag-a mora biti enoličen

- **Vsebine tag-ov ne smemo spreminjati** !!

Oznake

- Dve vrsti oznak
  - Lahke (lightweight)
    - Podobna je vejam (branches), ki se ne spremenijo.
    - Le kazalec na specifično potrditev (commit)
    - Običajno uporabi pri izdelavi začasnih oznak
  - Anotacijske (annotated)
    - Shranjene v bazi Git kot objekti
    - Vsebujejo ime izdelovalca oznake, e-pošto, sporočilo in se lahko podpišejo in verificirajo z GNU Privacy Guard (GPG)
    - Dobra praksa je, a se uporabijo anotacijske oznake

- Za pregled označb se uporabi ukaz **git tag**

Lahke oznake

- Uporaba ukaza **git tag <oznaka>** brez dodatnih zastavic -a, -s ali –m

```
C:\git\vaje [razvoj1]> git tag
v0.1
v0.2
C:\git\vaje [razvoj1]> git tag v3.3
C:\git\vaje [razvoj1]> git tag
v0.1
v0.2
v3.3
C:\git\vaje [razvoj1]>
```

- Uporaba ukaza **git show**
  - Prikazuje le informacije o potrditvah

Anotacijske oznake

- Uporaba ukaza **git tag –a <oznaka> -m „sporočilo“**

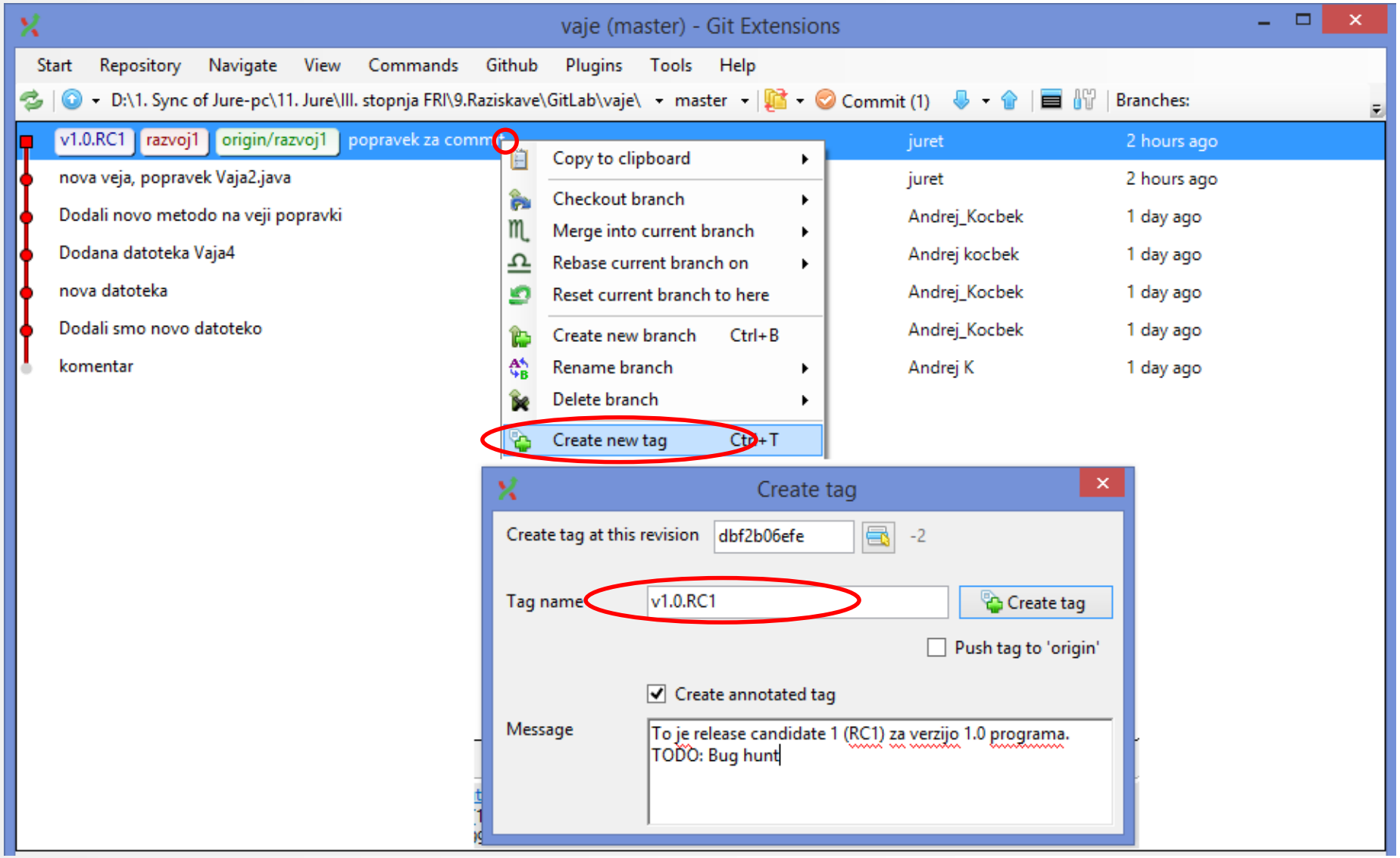
```
C:\git\vaje [razvoj1]> git tag
v0.1
C:\git\vaje [razvoj1]> git tag -a v0.2 -m "nova verzija 0.2"
C:\git\vaje [razvoj1]> git tag
v0.1
v0.2
```

- Uporaba ukaza **git show**
  - Prikaz vseh podatkov oznak, podpis GPG …

```
C:\git\vaje [razvoj1]> git show v0.1
tag v0.1
Tagger: Andrej_Kocbek <andrej@kocbek.si>
Date:   Fri Oct 10 09:01:51 2014 +0200

nova verzija 0.1

commit 26faf1b0c9fcc9974304da79cbe8e3f9cdc2cc68
Author: Andrej_Kocbek <andrej@kocbek.si>
Date:   Fri Oct 10 08:33:48 2014 +0200

    sprememba

diff --git a/storitve/Vaja2.java b/storitve/Vaja2.java
deleted file mode 100644
index 9780370..0000000
--- a/storitve/Vaja2.java
+++ /dev/null
@@ -1,9 +0,0 @@
-
-public class Vaja2{
- //dopolnitev metode
-
- // nova metoda
-
- // nova metoda 3
-}
-
C:\git\vaje [razvoj1]> _
```

Anotacijske oznake

- Uporaba ukaza **git tag –a <oznaka> -m „sporočilo“**



Dodatne funkcionalnosti oznak

- Verifikacija oznak
  - Za preverjanje podpisanih oznak se uporabi ukaz **git tag -v <oznaka>**
  - Potreben je javni ključ podpisnika znotraj zbirke ključev

- Zakasnelo označevanje
  - Omogočeno je označevanje preteklih potrditev
  - Pri označevanju je potreben celoten ali del checksum-a
  - **git tag -a <oznaka> -m „sporočilo“ <checksum>**

```
C:\git\vaje [razvoj1]> git tag
v0.1
v0.2
v3.3
C:\git\vaje [razvoj1]> git log --pretty=oneline
660704274b3aa52087d44c20d0aa6a8c0a0e2347 sd
26faf1b0c9fcc9974304da79cbe8e3f9cdc2cc68 sprememba
8e51788fe9df42c9e6c868842fc4717d00d34815 spremembe
280135b21783562487d375dcc5a961c7fabf9999 spremembe
53defe8468704beabb05eb90b9d48abc0f270be8 združili smo vsebino datoteke
bcc44ebcde834c1e06ae0b325d899b36f350498f dopolnitev metode na razvoj1
ed7b9b3a09aeecc61703fba142eed0718d9ba127 Dodali novo metodo na veji popravki
99c26841fd0b4b13e4f2f5819d3d3d66ceca86e9 Dodali novo metodo na veji razvoj1
7ce129693b99a1d10a4964e48accb0eb5a20d1a9 Dodana datoteka Vaja4
2e8e34ed4a036d574187b4fba6232e5bd9ffda66 nova datoteka
1f0f43df9606809c02e362f1a4e1385d5c5a16c0 Dodali smo novo datoteko
902146bd5f5eb87861c39b298f5fba1919c52d20 komentar
C:\git\vaje [razvoj1]> git tag -a v0.0.1 -m "verzija 0.0.1" 2e8e
C:\git\vaje [razvoj1]> git tag
v0.0.1
v0.1
v0.2
v3.3
C:\git\vaje [razvoj1]> _
```