

Sistemi za kontrolu verzija

Sistemi za kontrolu verzija

- Koriste se kada je potrebno pratiti **verzije projekta** i/ili kada **više programera radi na projektu**
- Bazirani su na repozitorijumima koda u koji se smeštaju fajlovi i folderi koji čine projekat
- Na repozitorijumu se čuvaju izmene (**commit**) kao i informacije o izmenama:
 - Ko je napravio izmenu?
 - Kad je napravljena?
 - Šta je izmena?
- Dostupan je istorijat izmena projekta na repozitorijumu
- Moguće pristupiti ranijim verzijama projekta i svim relevantnim podacima

Sistemi za kontrolu verzija

- Dva osnovna tipa sistema za kontrolu verzija
 - Centralizovani (SVN, CVS)
 - Distribuirani (**Git**, Mercurial)



Git

Šta je Git?

- Distribuirani sistem za kontrolu verzija
- Razvoj započeo Linus Torvalds, u aprilu 2005. godine, posle promene politike licenciranja BitKeeper-a koji je do tada korišćen za razvoj linux kernela. Dostupan na adresi <http://git-scm.com/>

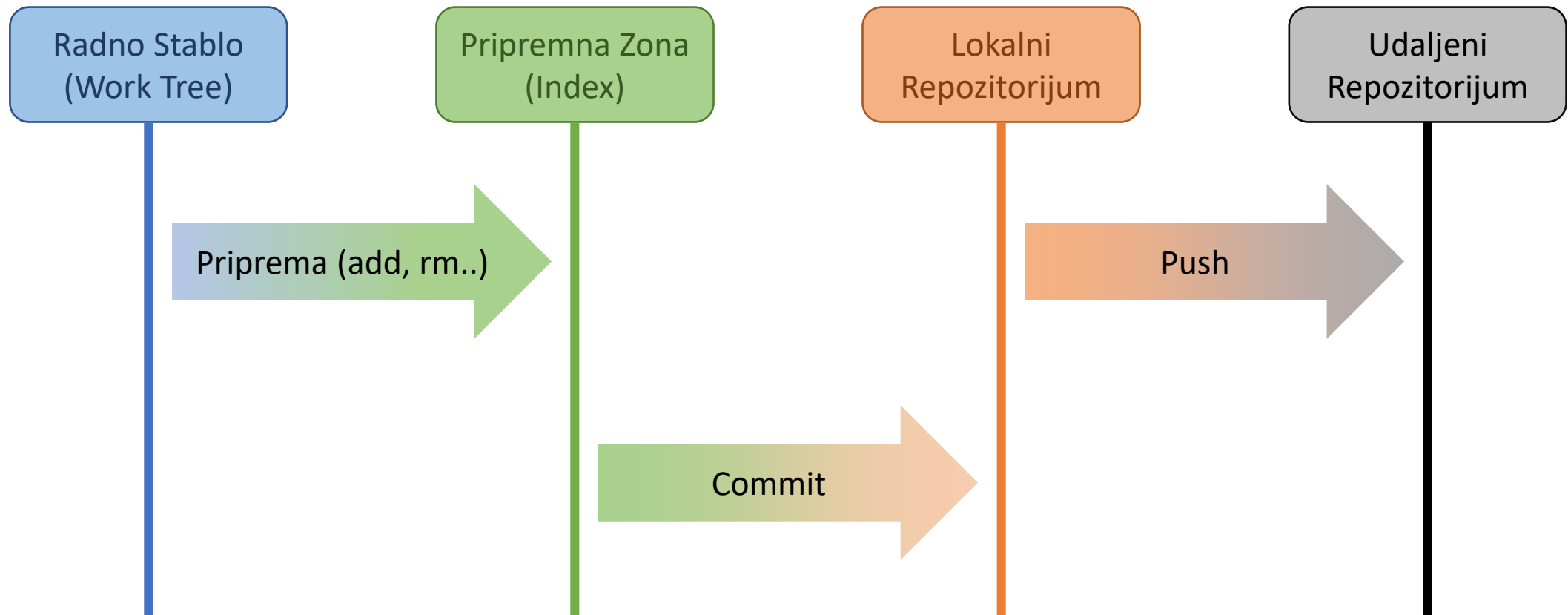
Git repozitorijum

- Sadrži foldere i fajlove sa kompletnom istorijom izmene (verzije)
- Osnova git repozitorijuma je praćenje **SADRŽAJA**, a ne fajlova i promena nad njima
- **Informacije se izračunavaju po potrebi**

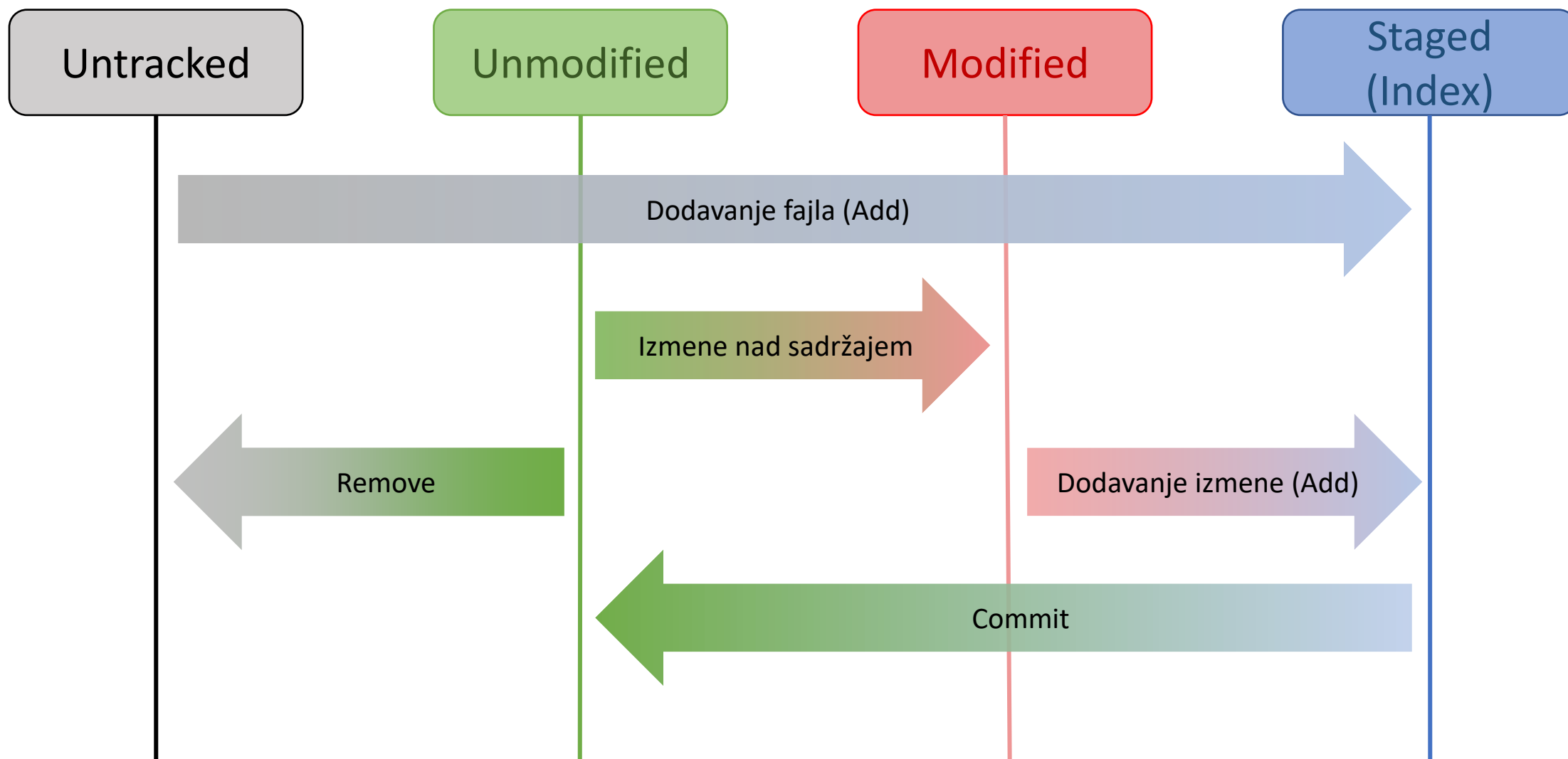
Osnovni Workflow


1. Kloniranje udaljenog repozitorijuma
2. Dodavanje, brisanje, izmene fajlova u radnom stablu (**Working Tree**)
 - Working tree je npr. lokalni fajl sistem
3. Dodavanje izmena u pripremnu zonu (**Index**)
4. Trajno beleženje promena (**Commit**)
 - Promene se prvo beleže lokalno
5. Postavljanje promena na udaljenom (**remote**) repozitorijumu (**Push**)

Osnovni Workflow



Životni ciklus fajla



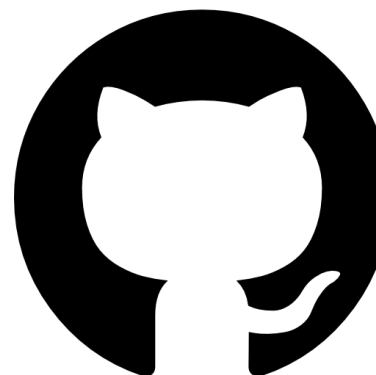


Instalacija
i
Konfiguracija

Popularna hosting rešenja



[GitLab](https://gitlab.com)



[GitHub](https://github.com)

Instalacija

- Git se može preuzeti sa linka: <https://git-scm.com/downloads>
- Windows:
 - Pokretanjem .exe fajla
- Linux:
 - Dolazi po default-u za mnoge distribucije
 - Iz terminala (Debian based distribucije - npr. Ubuntu):

```
$ sudo apt install git-all
```

Konfiguracija

- Konfiguracija se čuva u tekstualnim fajlovima koji imaju strukturu **.ini fajlova**
- Preporučivo je parametre podešavati putem **git config** komande.
- Tri nivoa konfiguracije:
 - **Sistemska**
 - Linux: /etc/gitconfig
 - Windows: <git_installation>\ mingw64\etc\gitconfig
 - **Po korisniku**
 - Linux: ~/.gitconfig
 - Windows: C:\Users\%USER\.gitconfig
 - **Po repozitorijumu**
 - Unutar .git direktorijuma radnog git repozitorijuma

Osnovna konfiguracija

- Konfigurisanje imena i email-a:

```
$ git config --global user.name "Petar Peric"  
$ git config --global user.email "petar.peric@mail.com"
```

- Konfiguracija se može izlistati sa:

```
$ git config --list
```

- Editor za unos log poruka je postavljen na podrazumevani sistemski (najčešće **vi** ili **vim**). Podrazumevani editor se može promeniti:

```
$ git config --global core.editor "<putanja_do_zeljenog_editora>"
```



Osnovne operacije

\$ git help

- Spisak osnovnih komandi git-a se može dobiti sa:

```
$ git help
usage: git [-version] ...
...
The most commonly used git commands are:
add          Add file contents to the index
bisect       Find by binary search the change that introduced a bug
branch       List, create, or delete branches
checkout     Checkout a branch or paths to the working tree
...
```

- Detaljna pomoć za neku komandu se može dobiti sa:

```
$ git help <ime_komande>
```


\$ git status

- Informacije o izmenama, untracked fajlovima, na kojoj se grani nalazimo, itd. se mogu videti pomoću **git status** komande:

```
$ git status
On branch master
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   added_1.py
    new file:   added_2.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    untracked_1.py
    untracked_2.py
    untracked_3.py
    untracked_4.py
```

\$ git **init**, git **clone**

- Kreiranje praznog repozitorijuma:

```
$ git init  
Initialized empty Git repository in  
<putanja_do_praznog_repozitorijuma>
```

- Nakon kreiranja, lokalni repozitorijum je potrebno povezati sa udaljenim:

```
$ git remote add origin <link_ka_udaljenom_repozitorijumu>
```

- Udaljeni repozitorijum (npr. kreiran na GitLab-u ili GitHub-u) možemo i klonirati:

```
$ git clone <link_ka_udaljenom_repozitorijumu>
```

\$ git add

- Untracked fajlovi se mogu dodati u [index](#):

1. Pojedinačno:

```
$ git add <putanja_do_fajla>
```

2. Po direktorijumu:

```
$ git add <putanja_do_foldera>
```

3. Svi unutar repozitorijuma:

```
$ git add --all
```

\$ git rm

- Prethodno `untracked` fajlovi dodati pomoću `git add` se sa `git rm` komandom mogu obrisati:

1. Iz `radnog stabla` i `index-a`:

```
$ git rm <ime_fajla>
```

2. Samo `index-a`:

```
$ git rm --cached <ime_fajla>
```

\$ git commit

- Promena se trajno beleži sa:

```
$ git commit
```

- Ukoliko želimo da odmah ostavimo i poruku:

```
$ git commit -m "moj komentar"
```

\$ git reset

- Sa git reset komandom možemo da poništimo:

1. Fajlove sa statusom **modified** ili koji su **tek dodati**:

```
$ git reset HEAD <ime_fajla>
```

2. **n commit-ova** unazad:

```
$ git reset HEAD~<n>
```

\$ git push

- Za slanje promena na udaljeni repozitorijum:
 - Ovde se promene automatski šalju na našu **upstream** granu (više o tome kasnije)
 - Promene uključuju sve **commit-ove** do izvršenja komande

```
$ git push
```

\$ git pull

- Povlači promene sa udaljenog repozitorijuma i spaja ih na lokalnom repozitorijumu
 - Prilikom pull-a može doći do **konflikata**

```
$ git pull
```

- Git pull zapravo zamenjuje dve komande:

```
$ git fetch  
$ git merge origin/<ime_grane>
```


\$ git stash

- Komanda koja se koristi kada hoćemo da sačuvamo izmene u **radnom direktorijumu (stablu)** i **index-u**, ali želimo da se vratimo na „čisto“ stanje
- „Čisto“ stanje je recimo stanje repozitorijuma u nekom trenutku (**commit-u**), ali bez **lokalnih izmena**
- Kada bismo hteli da koristimo ovu komandu?
 - Pre nego što odradimo komandu pull, želimo da promene na našoj „backupujemo“, kako bi nam radno stablo bilo čisto.
 - Nakon pull-a možemo da povratimo naše promene
 - Razrešimo konflikte ukoliko postoje i commit-ujemo novonastale promene
 - Odradimo push ka udaljenom repozitorijumu

\$ git stash

- Stash-ovanje lokalnih promena:

```
$ git stash
```

- Listanje stash-a:

```
$ git stash list  
stash@{0}: WIP on <ime_grane>: <hash_commit-a> Ovo je neka poruka na trenutnom commit-u
```

- Vraćanje promene iz stash-a u **radni direktorijum**:
 - **pop** uklanja promene iz memorije stash-a
 - **apply** ne uklanja promene iz memorije stash-a

```
$ git stash pop stash@{0}  
$ git stash apply stash@{0}
```

\$ git log

- Istorija se može pregledati sa komandom:

```
$ git log
commit frJXBOSqkc3AxZZTYwOW7YjyCEYLq0q5fPNcVjLp (HEAD -> master, origin/master)
Author: author <author@gmail.com>
Date:   Sun Oct 27 16:16:16 2019 +0100

    My commit message that includes changes for the commit

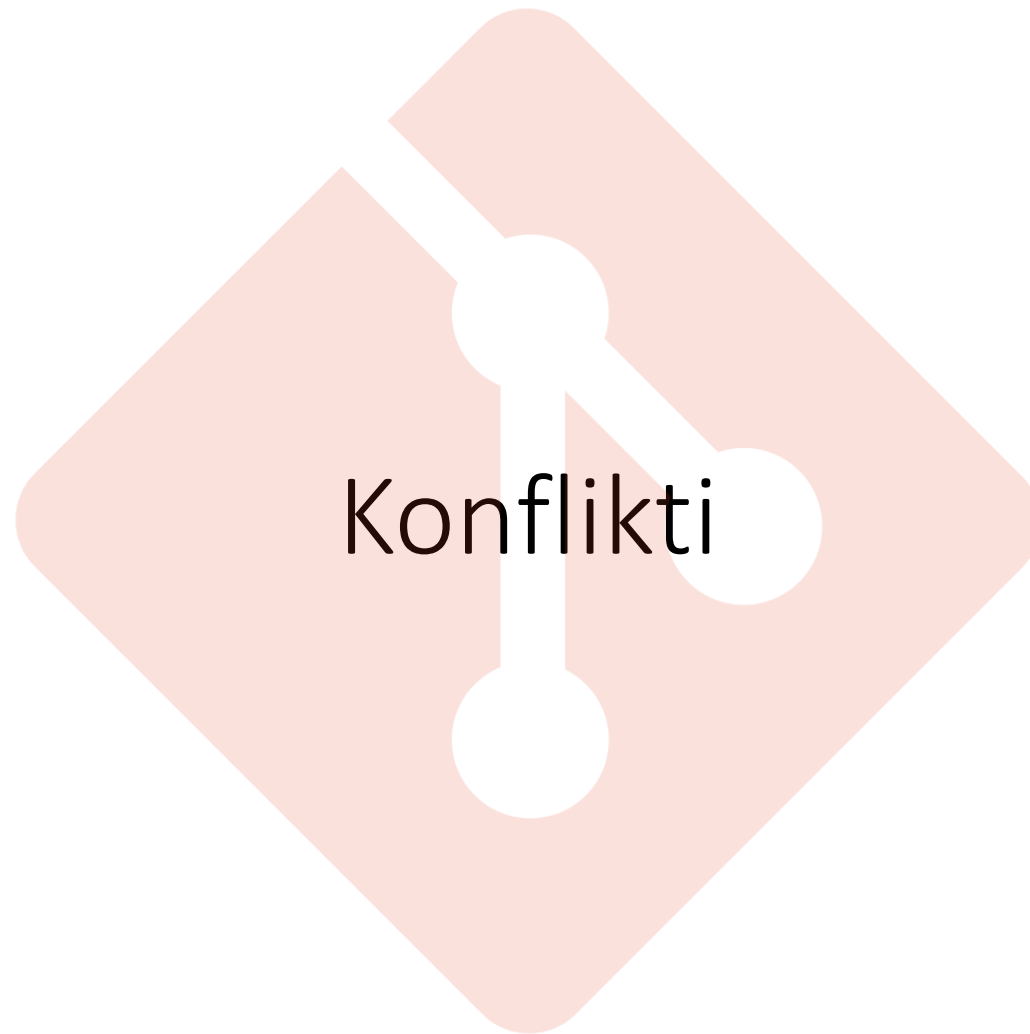
...
```

- Grafički pregled istorije za tekuću granu:

```
$ gitk
```

- Grafički pregled istorije za sve grane:

```
$ gitk --all
```



Konflikti

- Nastaju kad spajanje izmena (npr. sa udaljenog repozitorijuma na lokalno) nije uspešno izvršeno
- Za fajlove čiji je **sadržaj** u konfliktu označene su **konfliktne linije**

Konflikti

```
<<<<<< HEAD:file.txt
Hello world
=====
Goodbye
>>>>>> 77976da35a11db4580b80ae27e8d65caf5208086:file.txt
```

- Šta se desilo u fajlu 'file.txt'? Odgovor: **Konflikt**.
- Između markera <<<<<< i ===== se nalaze naše lokalne izmene:

```
<<<<<< HEAD:file.txt
Hello world
=====
```

- Između markera ===== i >>>>>> se nalaze izmene povučene sa **commit-a** koji je opisan hash-om:

```
=====
Goodbye
>>>>>> 77976da35a11db4580b80ae27e8d65caf5208086:file.txt
```

Konflikti

- Konflikte možemo ručno razrešiti
- Razrešavanje konflikata predstavlja samo još jedan vid izmena nad fajlovima
 - Dalji workflow se ne razlikuje od klasičnog
- Nakon razrešavanja konflikta potrebno ih je povrditi sa **git add**

```
$ git add file.txt
```

- Ukoliko smo završili sa izmena i želimo da ih potvrdimo commit-om

```
$ git commit -m "poruka o commit-u"
```



Grananje

- Grane su alternativni tokovi razvoja
- Kreiranje grana kod Git-a je jednostavno i brzo
- Ohrabruje se njihovo često kreiranje
 - Za svaki **feature**, **bugfix**, itd.
- Osnovna grana se zove **master**
 - Obično uvek postoji (sem kod praznog repozitorijuma)

Primer workflow-a

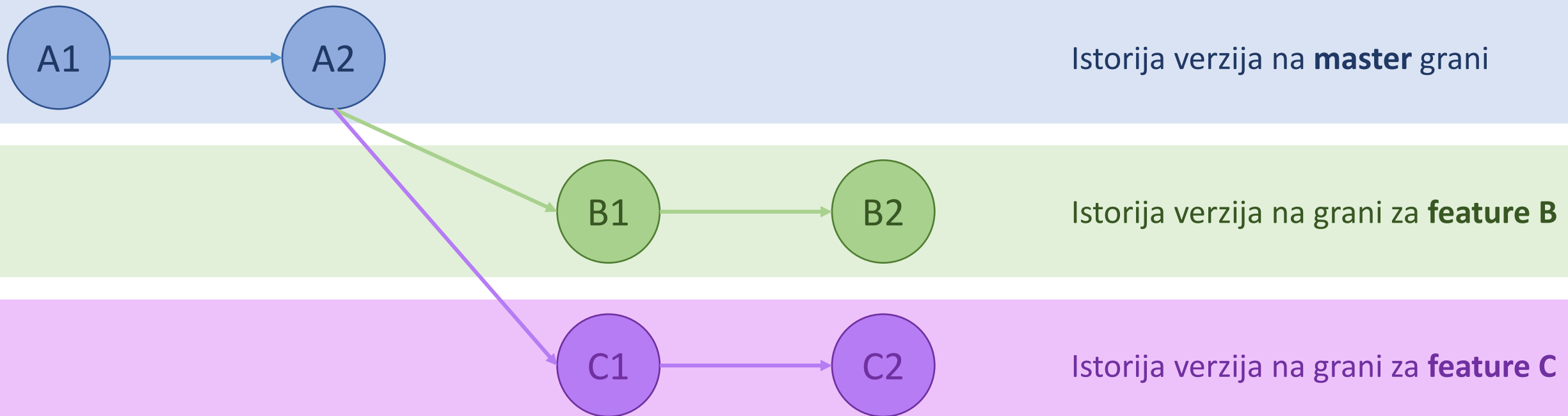
- Imamo naš projekat
- Za **svaki feature pravimo novu granu** gde taj feature implementiramo

Međutim otkrio se bug na master-u

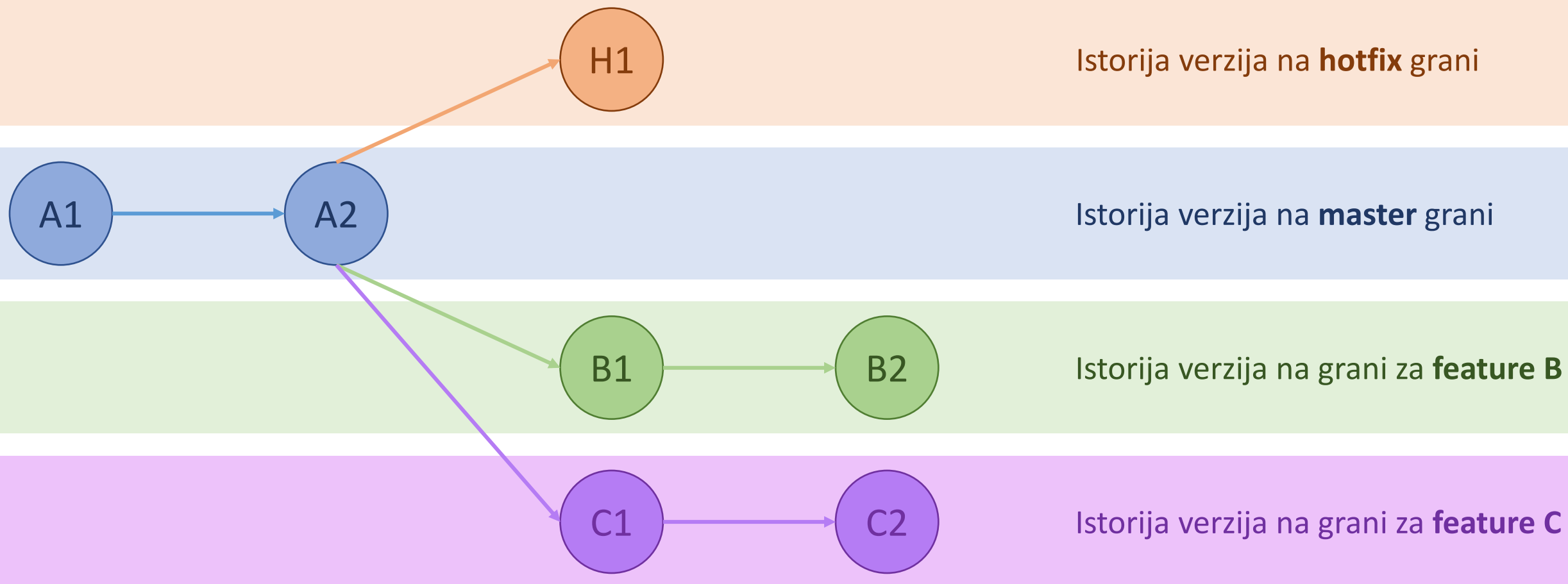
- Prebacujemo se na master
- Pravimo novu granu gde ćemo rešavati bug (npr. **hotfix grana**)
- Rešavamo bug na hotfix grani
- Kada smo zadovoljni rešenjem hotfix granu možemo spojiti sa master granom (**merge**)

Nastavljamo dalje sa radom...

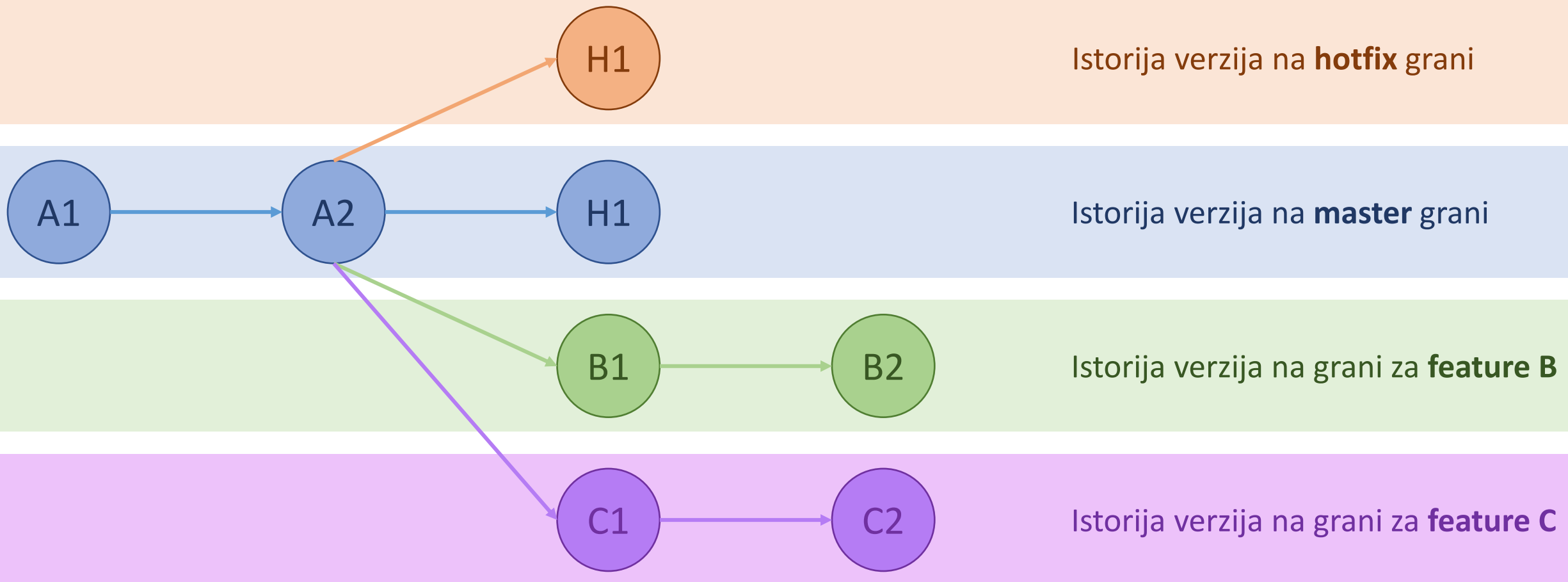
Kako grananje izgleda?



Kako grananje izgleda?



Kako grananje izgleda?



Vrste grana

- **Lokalne**
 - Nastale u lokalnom repozitorijumu i ne prate druge grane
- **Tracking**
 - Lokalne grane koje prate druge grane
 - Podešena im je **upstream** grana
 - Najčešće prate remote tracking grane
- **Remote Tracking**
 - Grane nastale u udaljenom (remote) repozitorijumu koje se kloniraju u lokalni

\$ git branch

- Komanda za pravljenje novih grana i dobijanja informacija o granama
- Pravljenje nove nezavisne grane na lokalnu:

```
$ git branch <ime_grane>
```

- Spisak lokalnih grana (* označava trenutnu):

```
$ git branch
* master
feature_grana_1
feauture_grana_2
Hotfix_grana_1
```

- Spisak lokalnih i remote tracking grana:

```
$ git branch -a
```

\$ git checkout

- Sa git checkout se možemo sa trenutne grane prebaciti na drugu:

```
$ git checkout <ime_grane>  
Switched to branch '<ime_grane>'  
$ git checkout master  
Switched to branch 'master'
```

- Takođe možemo napraviti novu granu i odmah se prebaciti na nju:

```
$ git checkout -b <ime_grane>  
Switched to a new branch '<ime_grane>'
```


\$ git push

- Da bi promene nastale na lokalnoj grani prebacili i na udaljeni (remote) repozitorijum git push pozivamo sa flagom **--set-upstream:**
 - Ime udaljenog repozitorijuma je u većini slučajeva **origin**

```
$ git push --set-upstream <ime_udaljenog_repozitorijuma> <ime_grane>
```



Spajanje grana

\$ git merge

Postoje dve strategije spajanja grana sa git merge komandom:

- Fast-forward
- No-fast-forward

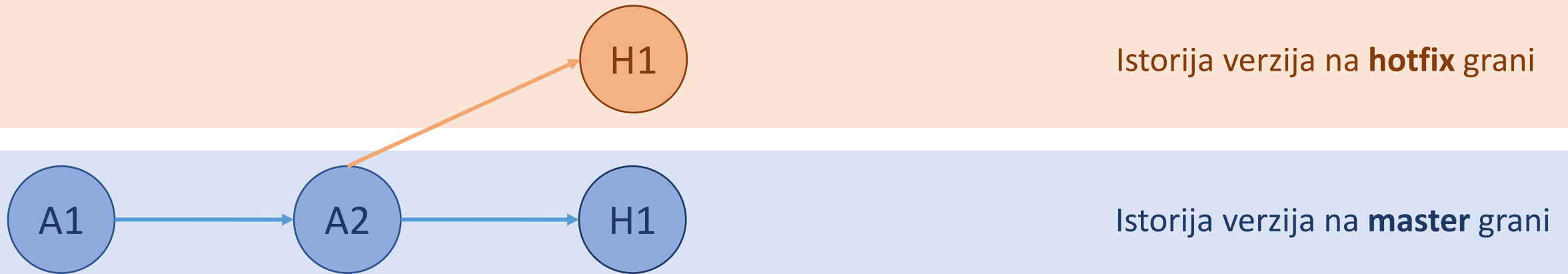
\$ git merge: *fast-forward*



- U ovom primeru stanje (istorija) **master grane** se nije menjalo dok je rađeno na **hotfix grani**, odnosno na masteru **nema** novih **commit-ova**
 - Samo u ovom slučaju je moguć **fast-forward** merge

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
...
```

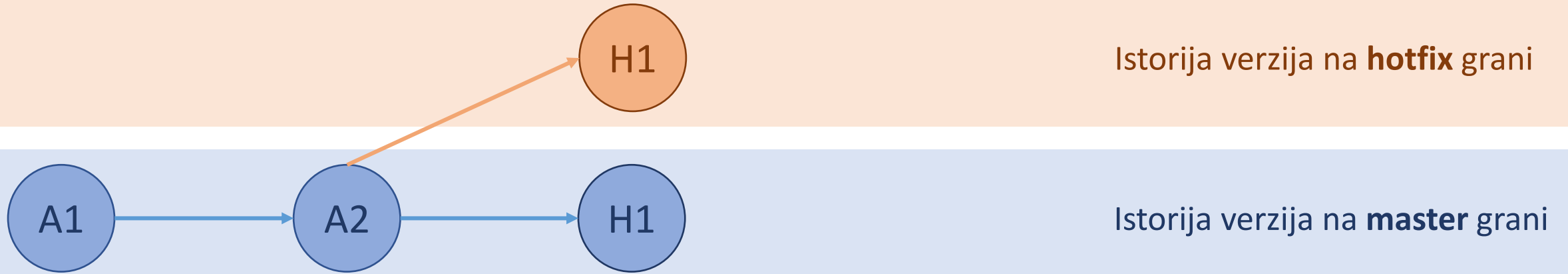
\$ git merge: *fast-forward*



- U ovom primeru stanje (istorija) **master grane** se nije menjalo dok je rađeno na **hotfix grani**, odnosno na masteru **nema** novih **commit-ova**
 - Samo u ovom slučaju je moguć **fast-forward** merge

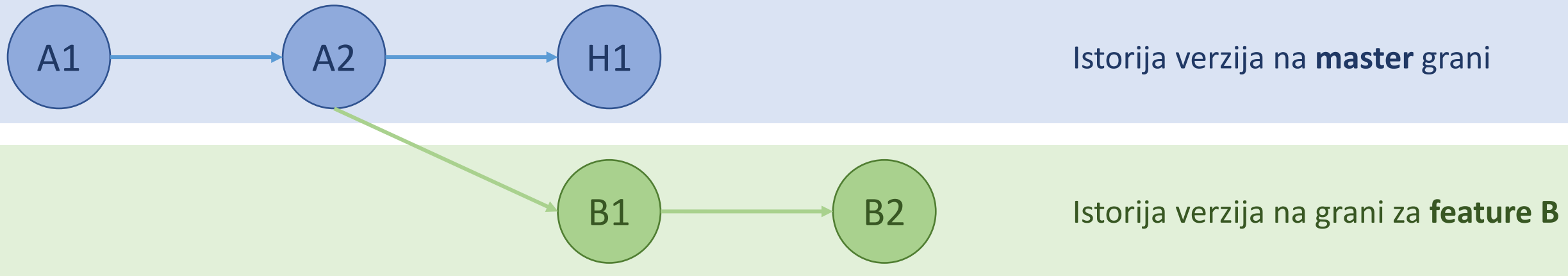
```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
...
```

\$ git **merge**: *fast-forward*



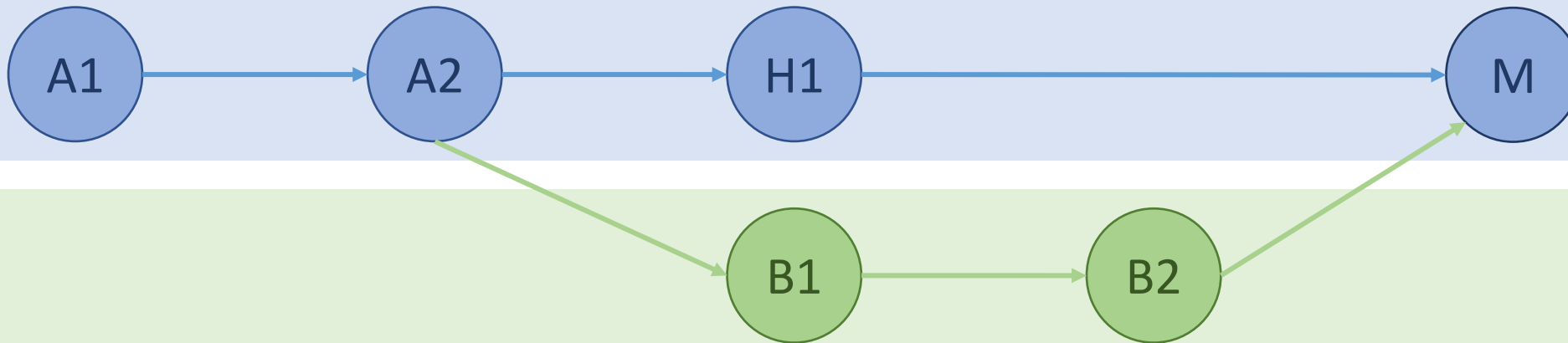
- Posle **merge-a**:
 - **master** grana pokazuje na commit - H1
 - **hotfix** grana pokazuje na commit - H1

\$ git merge: *no-fast-forward*



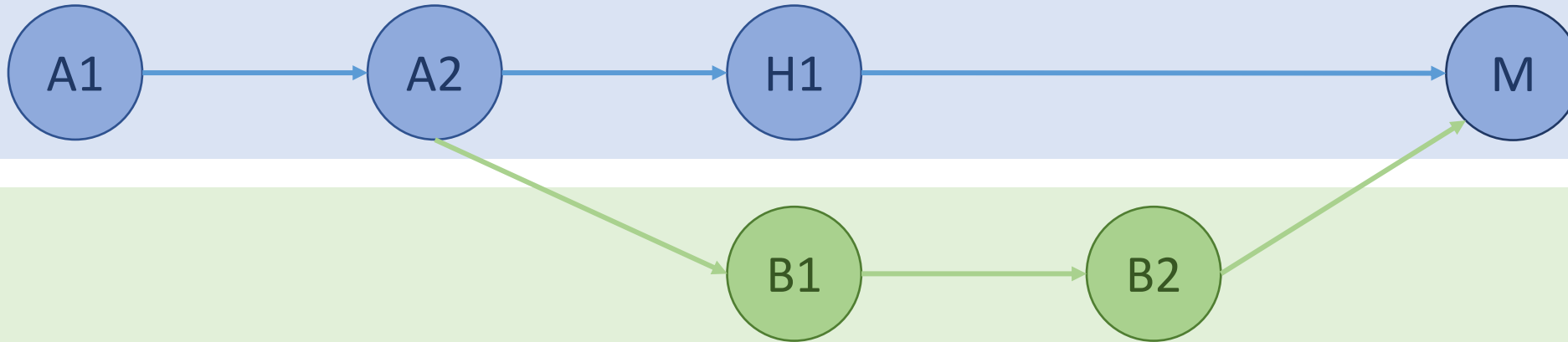
- U ovom primeru stanje (istorija) **master grane** se menjala dok je rađeno na **feature B grani**, odnosno na masteru **ima** novih **commit-ova**
- U ovom primeru novi commit je baš onaj sa **hotfix grane**
- U ovom slučaju nije moguć **fast-forward** merge

\$ git **merge**: *no-fast-forward*



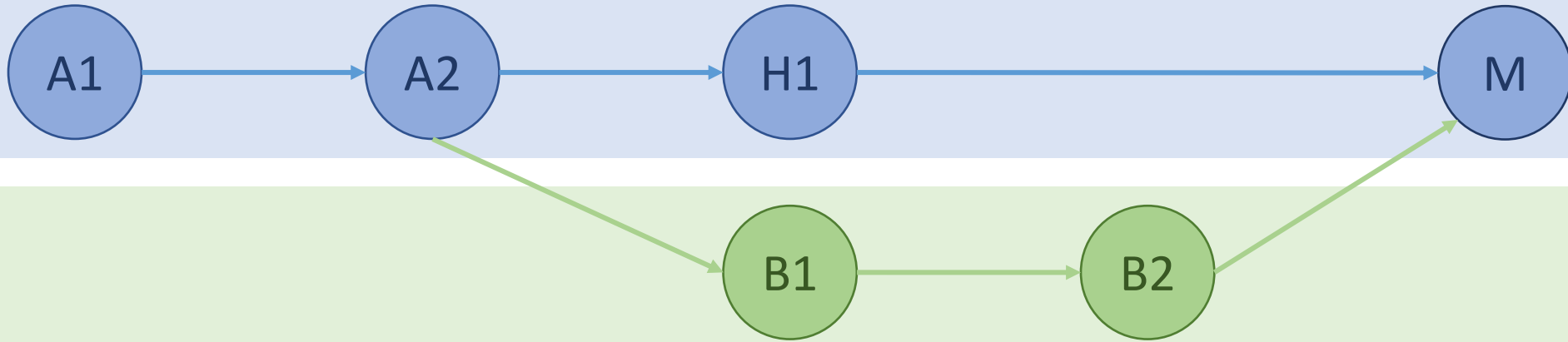
- U slučaju kad nije moguć fast-forward merge git pravi novi commit
- Novi commit se zove **merge commit**
- Merge commit (**M**) pokazuje na:
 - Prethodni commit na grani **na** kojoj je urađen merge (**H1**)
 - Prethodni commit na grani **sa** koje je urađen merge (**B2**)

\$ git merge: *no-fast-forward*



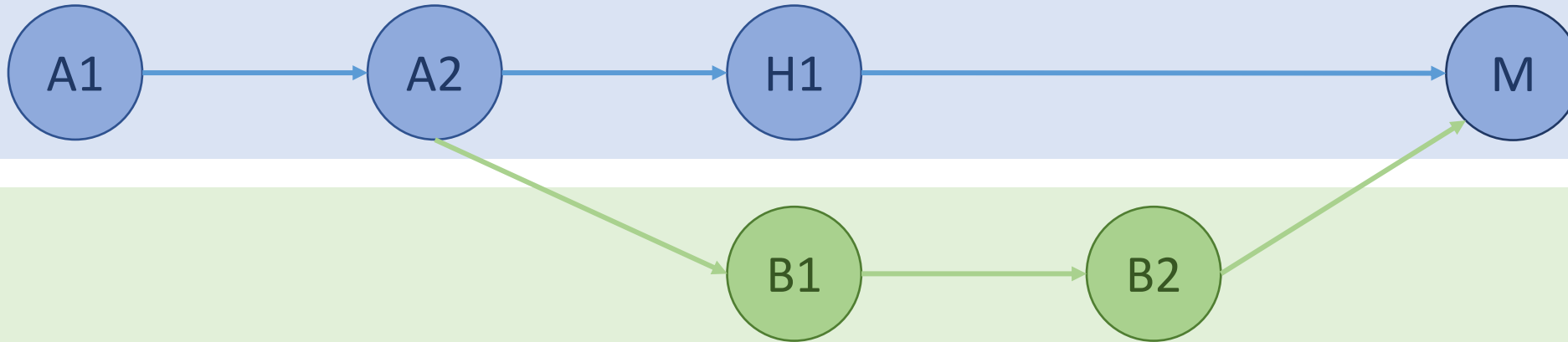
- Mogu se desiti dva slučaja:
 - Bez konflikata
 - odmah se pravi merge commit
 - Sa konfliktima
 - pravljenje merge commit-a se **pauzira**
 - tada ih je potrebno ručno **razrešiti** i **dodati** mergovane izmene
 - nastaviti sa pravljenjem commit-a (poziva se **git commit**)

\$ git **merge**: *no-fast-forward*



```
$ git checkout master
$ git merge feature_b
Auto-merging <ime_fajla>
CONFLICT (content): Merge conflict in <ime_fajla>
Automatic merge failed; fix conflicts and then commit the result.
```

\$ git **merge**: *no-fast-forward*



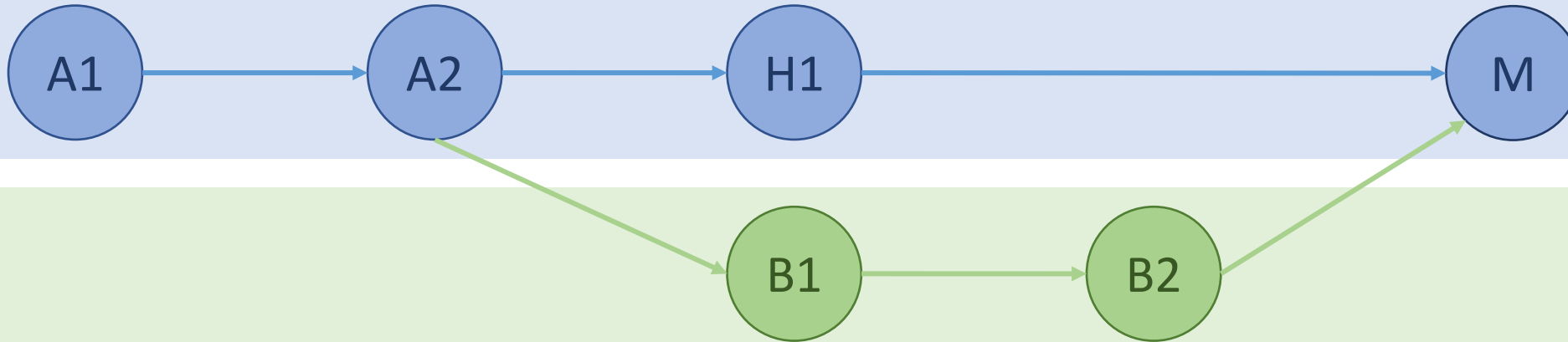
```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:    <ime_fajla>

no changes added to commit (use "git add" and/or "git commit -a")
```

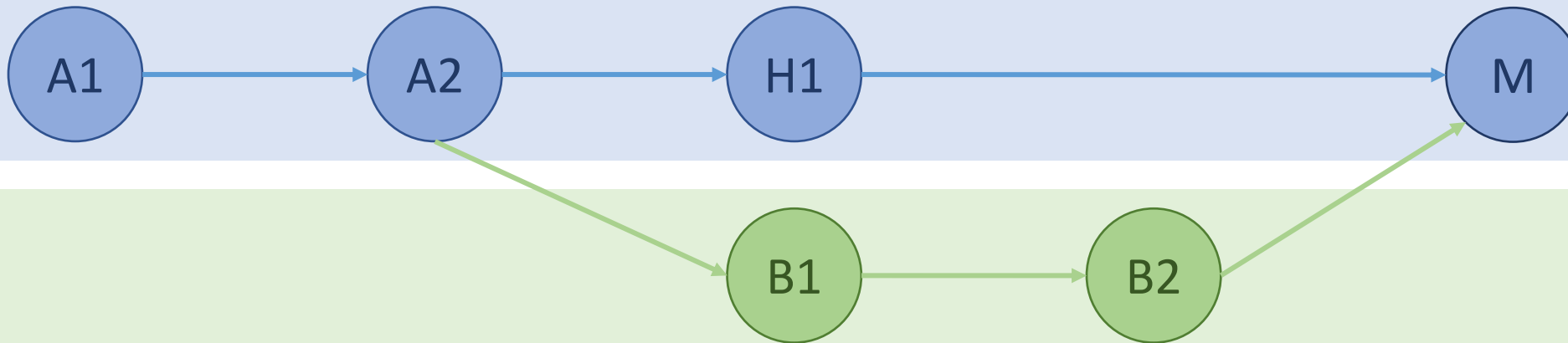
\$ git **merge**: *no-fast-forward*



Ukoliko smo zadovoljni sa izmenama:

```
$ git commit
```

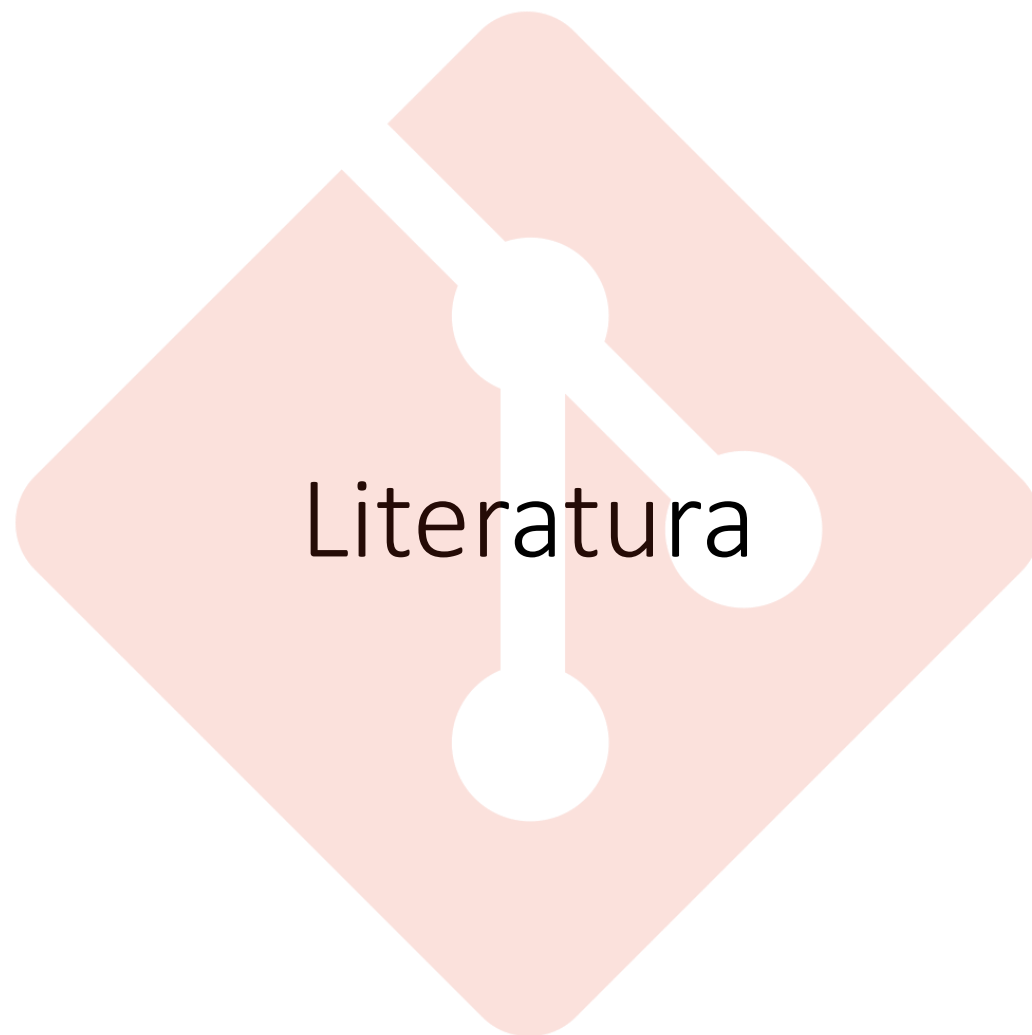
\$ git **merge**: *no-fast-forward*



Pri čemu će se automatski izgenerisati poruka u podrazumevanom text editoru:

```
Merge branch 'feature_b'

Conflicts:
  <ime_fajla>
#
# It looks like you may be committing a merge.
# ...
```



Literatura i dalje čitanje

- Slajdovi prof. Igora Dejanovića:
 - <http://www.igordejanovic.net/courses/tech/git/>
- Atlassian Git tutorijali:
 - <https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud>
- Git dokumentacija:
 - <https://git-scm.com/doc>