Системи за контролу верзија. Гит. Конфигурисање Гита. Основе Гита.

Веб програмирање

Доц. др Слободан Јелић

Грађевински факултет, Универзитет у Београду
Основне академске студије
смер: Геоинформатика
2. предавање

Београд, 9. октобар 2024.

Увод

- Шта је контрола верзије?
- Примитивни систем за контролу верзија
- Централизовани системи за контролу верзија (ЦСКВ)
- Дистрибуирани системи за контролу верзија (ДСКВ)

Гит

- Историјски развој Гита
- Делта принцип
- Ток снимака
- Локалност
- Интегритет
- Опозивост
- Три стања
- Ток рада с Гитом

Конфигурисање Гита

- Подешавања
- Дефинисање идентитета
- Дефинисање подразумеваног уређивача текста
- Дефинисање подразумеване гране
- Приказ подешавања
- Помоћ

Основе Гита

- Прављење Гит репозиторијума
- Иницијализација репозиторијума у постојећем директоријуму
- Клонирање постојећег директоријума
- Снимање промена у репозиторијуму
- Провера статус фајлова
- Праћење нових фајлова
- Стејџовање измењених фајлова
- Кратак статус
- Игнорисање фајлова
- Пример игнорисања
- Детаљан преглед припремљених и неприпремљених промена
- Комитовање промена

Увод

Шта је контрола верзије?

- **контрола верзије**[1] (енг. Version Control) подразумева праћење промена на фајлу или скупу фајлова кроз одређени временски период
- углавном се примењује на програмски код

Пример

Радите у програмерском тиму који се бави израдом веб апликације (клијентске и серверске стране). Сваки од чланова тима ради на неком задатку као што је: кориснички интерфејс и форме за унос података, прегледи и табеле података, контролери и руте, база података, итд... Поставља се неколико питања?

[1] понекад кажемо и контрола верзије програмског кода

Увод

Шта је контрола верзије?

Пример

1. Кад свако заврши на појединачном задатку, како ћете спојити појединачне резултате?
2. Како ће пројектни менаџер да прати рад сваког од чланова тима?
3. Уколико ваше последње решење не може да буде прихваћено, како ћете се вратити на неко од претходних?
4. Уколико коначно решење свих чланова тиме не може да буде прихваћено, како ћете се вратити на претходно продукцијско решење?
5. Ако изгубите код или направите значајне грешке, како ћете се вратити на претходно стање кода?

Увод

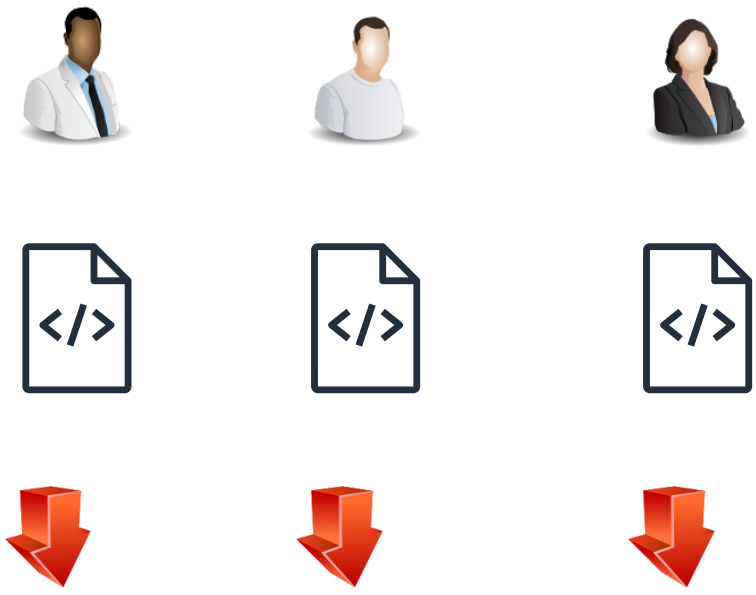
Шта је контрола верзије?

Увод

Шта је контрола верзије?

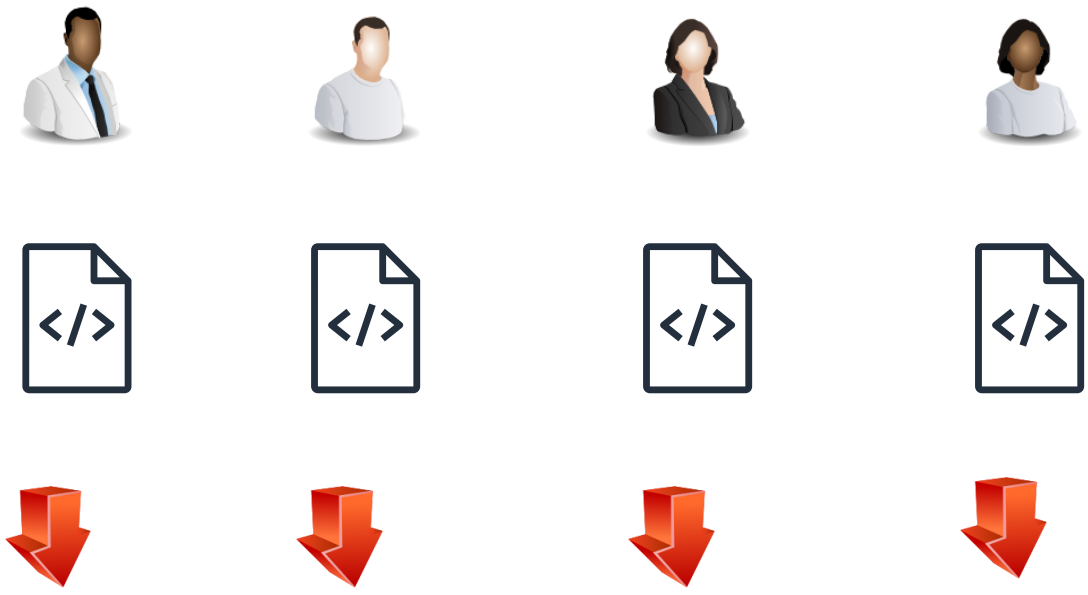
Увод

Шта је контрола верзије?



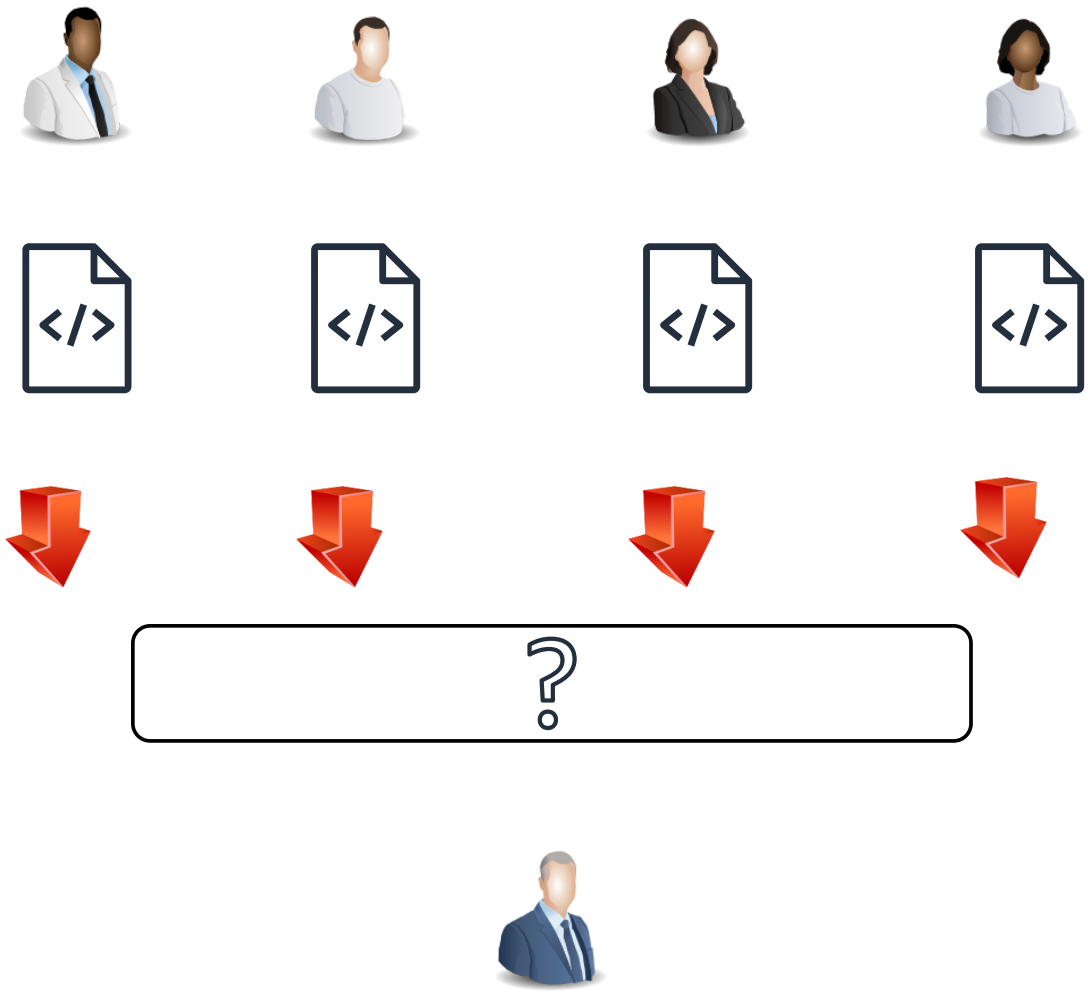
Увод

Шта је контрола верзије?



Увод

Шта је контрола верзије?



Увод

Шта је контрола верзије?

- решење претходно наведених проблема даје **систем за контролу верзија (СКВ)**[2].
- овај систем може да изведе следеће операције:
  - врати одабране фајлове на неко од претходних стања,
  - врати цео пројекат на претходно стање,
  - пореди промене на програмском коду кроз време,
  - утврди ко је направио промене на програмском коду које су изазвале проблеме, итд.

[2] (енг. Version Control System (VCS))

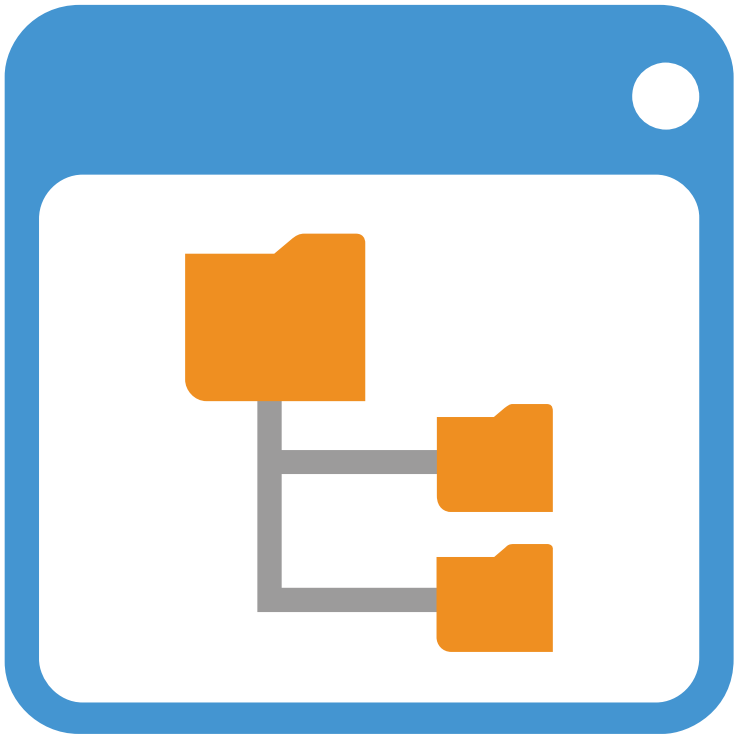
Увод

Примитивни систем за контролу верзија

- први (примитивни) систем за контролу верзија састојао се од ручног спремања сваке верзије на локални фајл систем
- подложан грешкама и губитку кода
- тешко даје одговоре на неке од наведених проблема

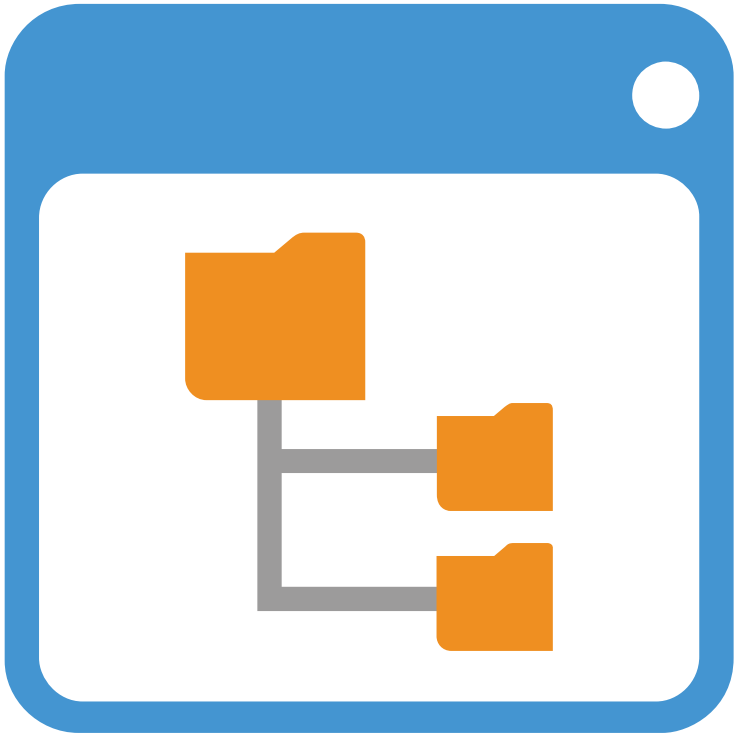
Увод

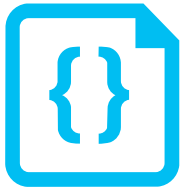
Примитивни систем за контролу верзија



Увод

Примитивни систем за контролу верзија

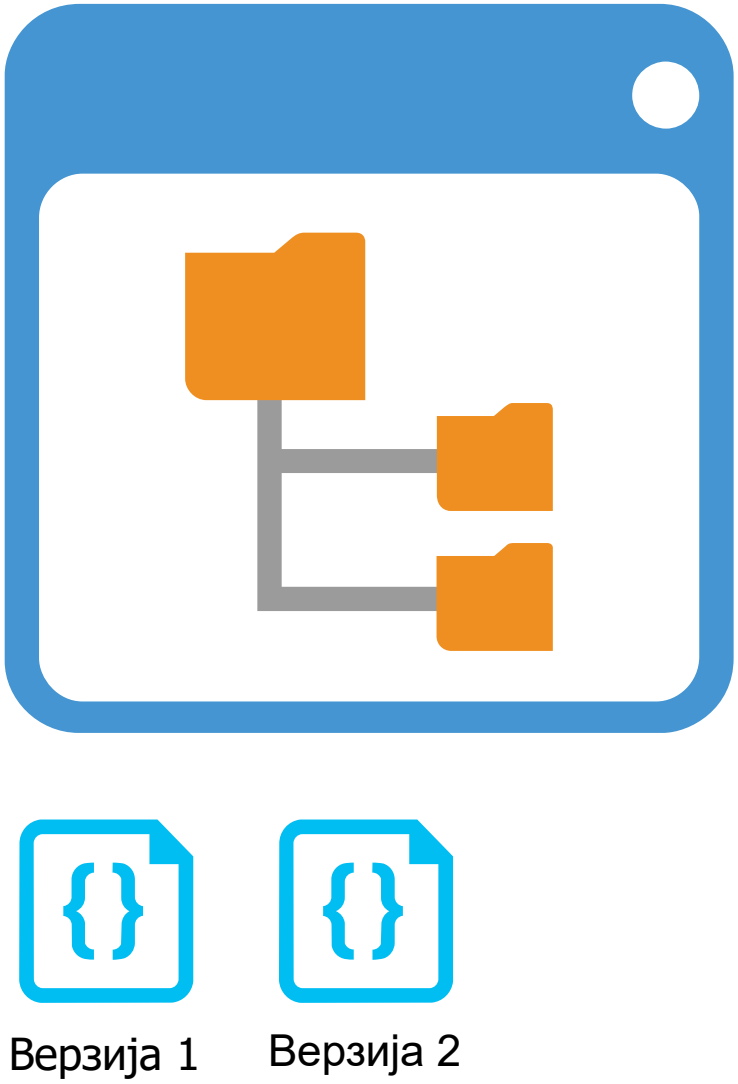




Верзија 1

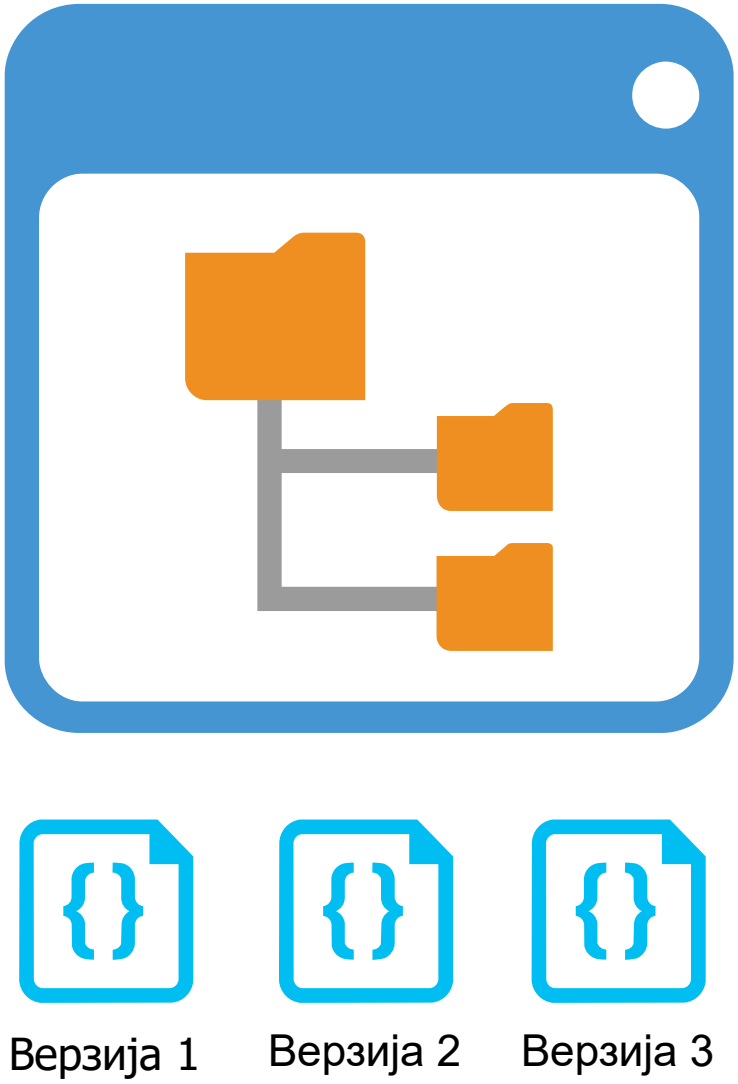
Увод

Примитивни систем за контролу верзија



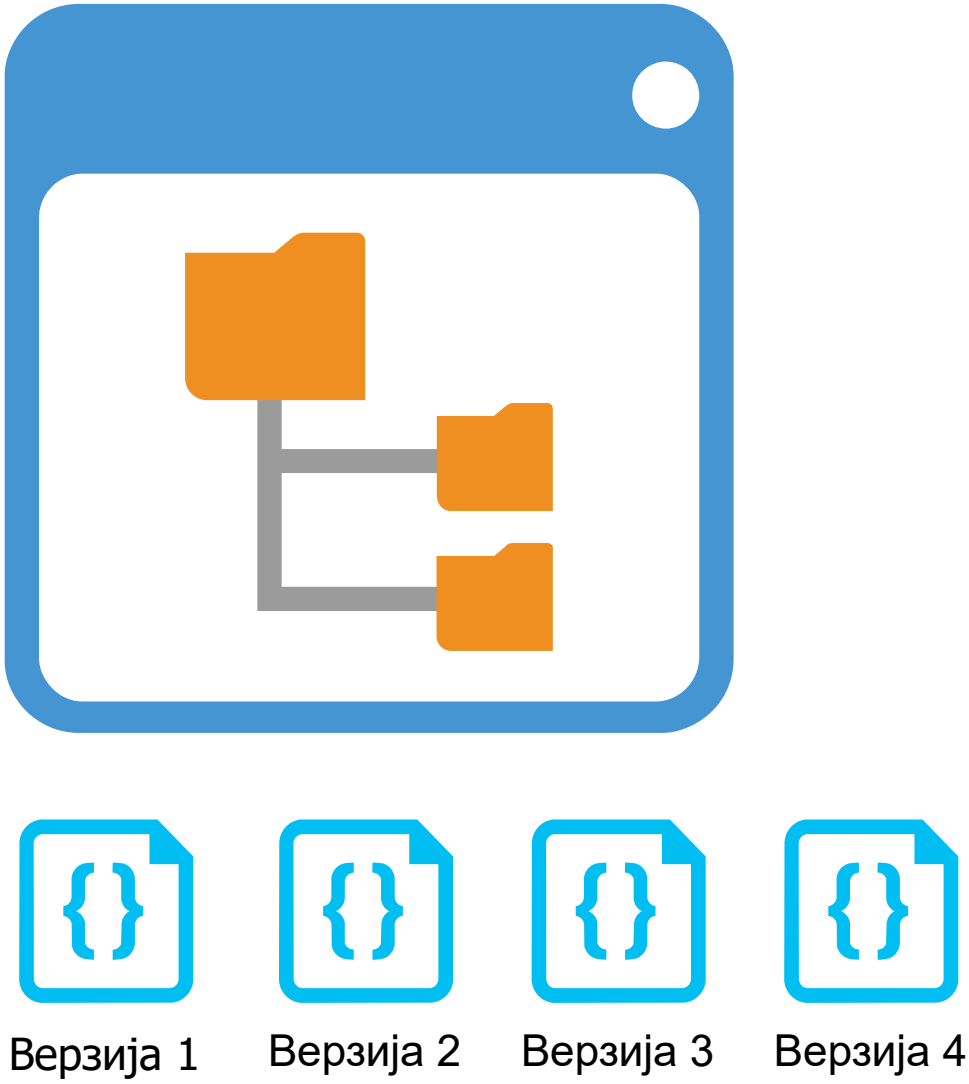
Увод

Примитивни систем за контролу верзија



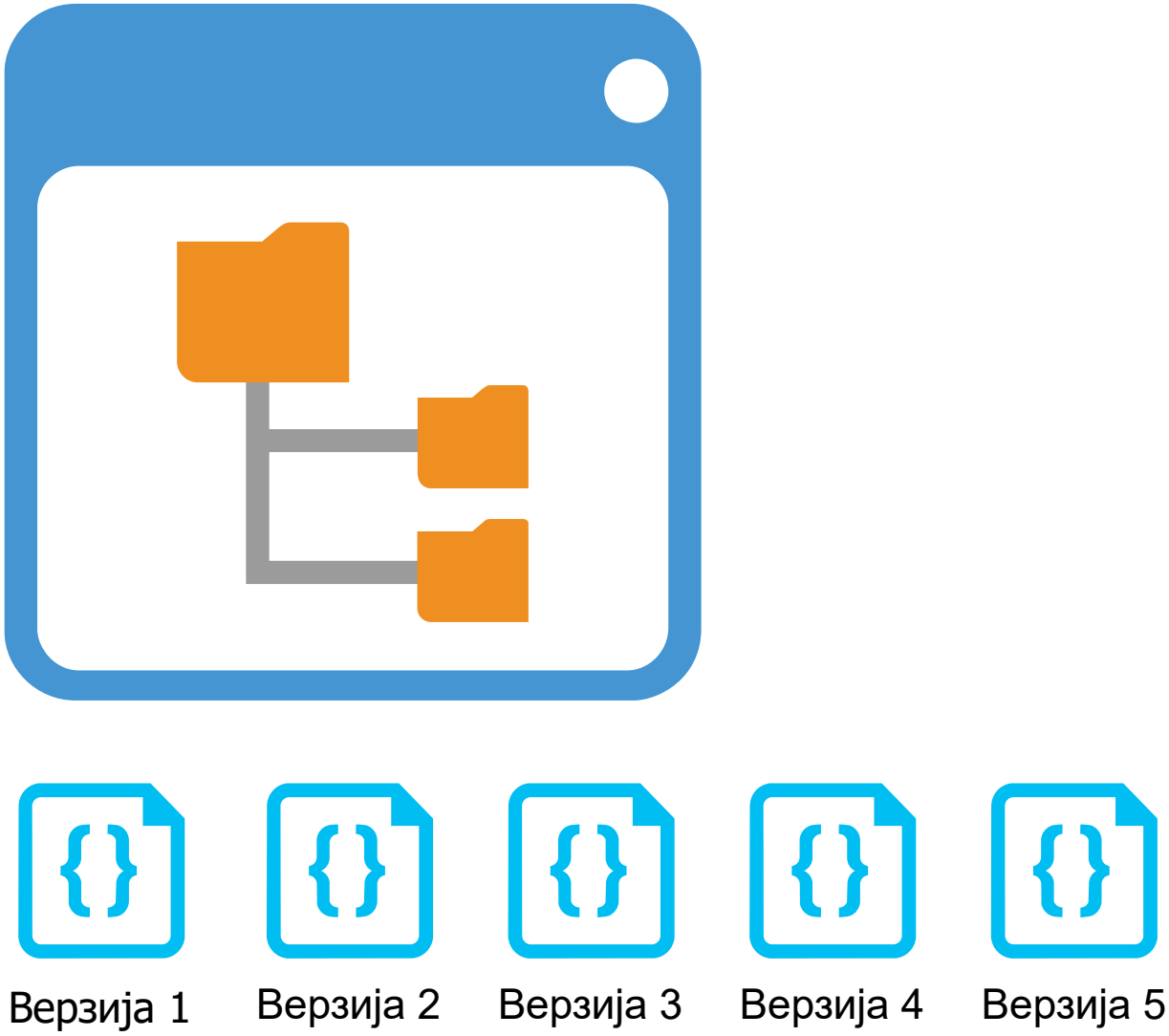
Увод

Примитивни систем за контролу верзија



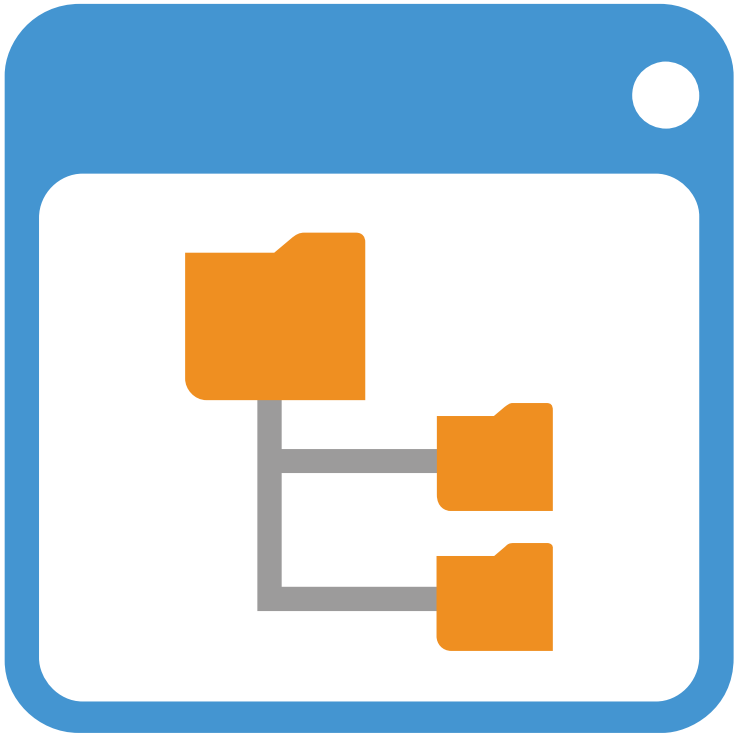
Увод

Примитивни систем за контролу верзија

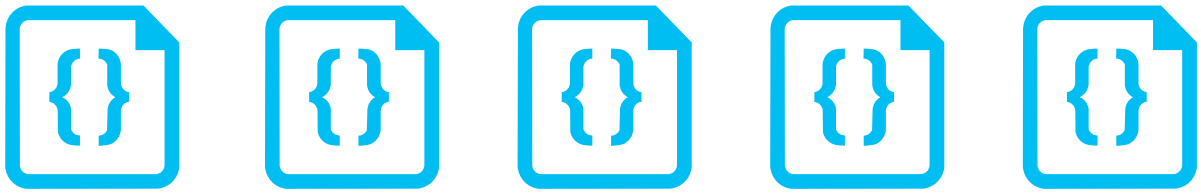


Увод

Примитивни систем за контролу верзија

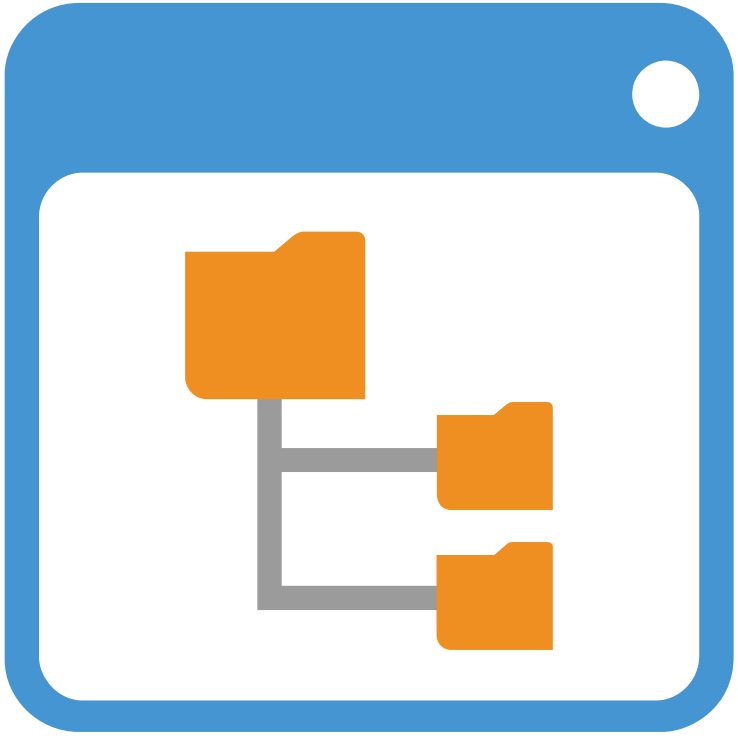


Свака верзија спрема се у посебан фајл

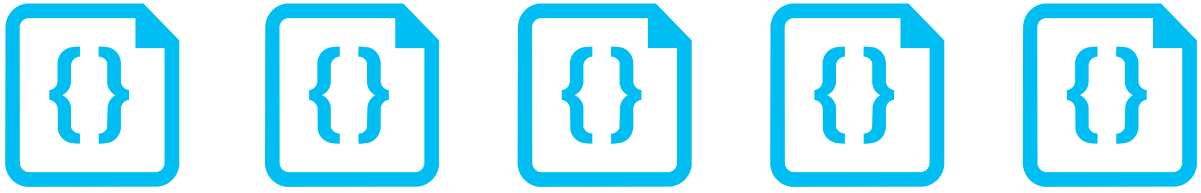


Увод

Примитивни систем за контролу верзија



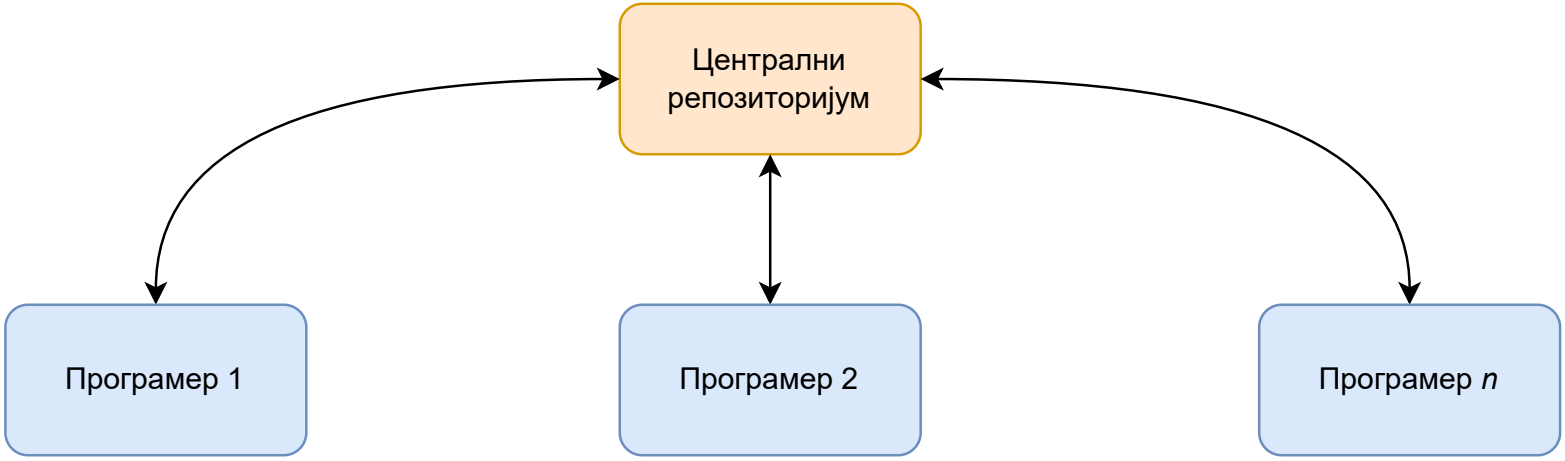
Свака верзија спрема се у посебан фајл



Увод

Централизовани системи за контролу верзија (ЦСКВ)

- потреба сарадње с програмерима довела је до развијања **централизованог система за контролу верзије (ЦСКВ)** (енг. Centralized Version Control System)
- у овом приступу постоји један централизовани сервер који садржи све верзије фајлова кода
- програмери (клијенти) спајају се на тај централни сервер и преузимају фајлове с тог централног места



Увод

Централизовани системи за контролу верзија (ЦСКВ)

Предности

- сваки програмер у сваком моменту зна шта ради други програмер
- администратори имају потпуну контролу над системом јер се конфигурише на централном месту

Недостаци

- изложеност централног сервера кваровима, падовима система, оштећењу или трајном квару дискова, итд.
- у случају недоступности централног сервера, програмерски тим не може да ради на пројекту
- у случају трајног квара диска, долази до потпуног губитка пројектних података, кода и документације[3]

[3] осим радних локалних копија кода појединих програмера

Увод

Централизовани системи за контролу верзија (ЦСКВ)

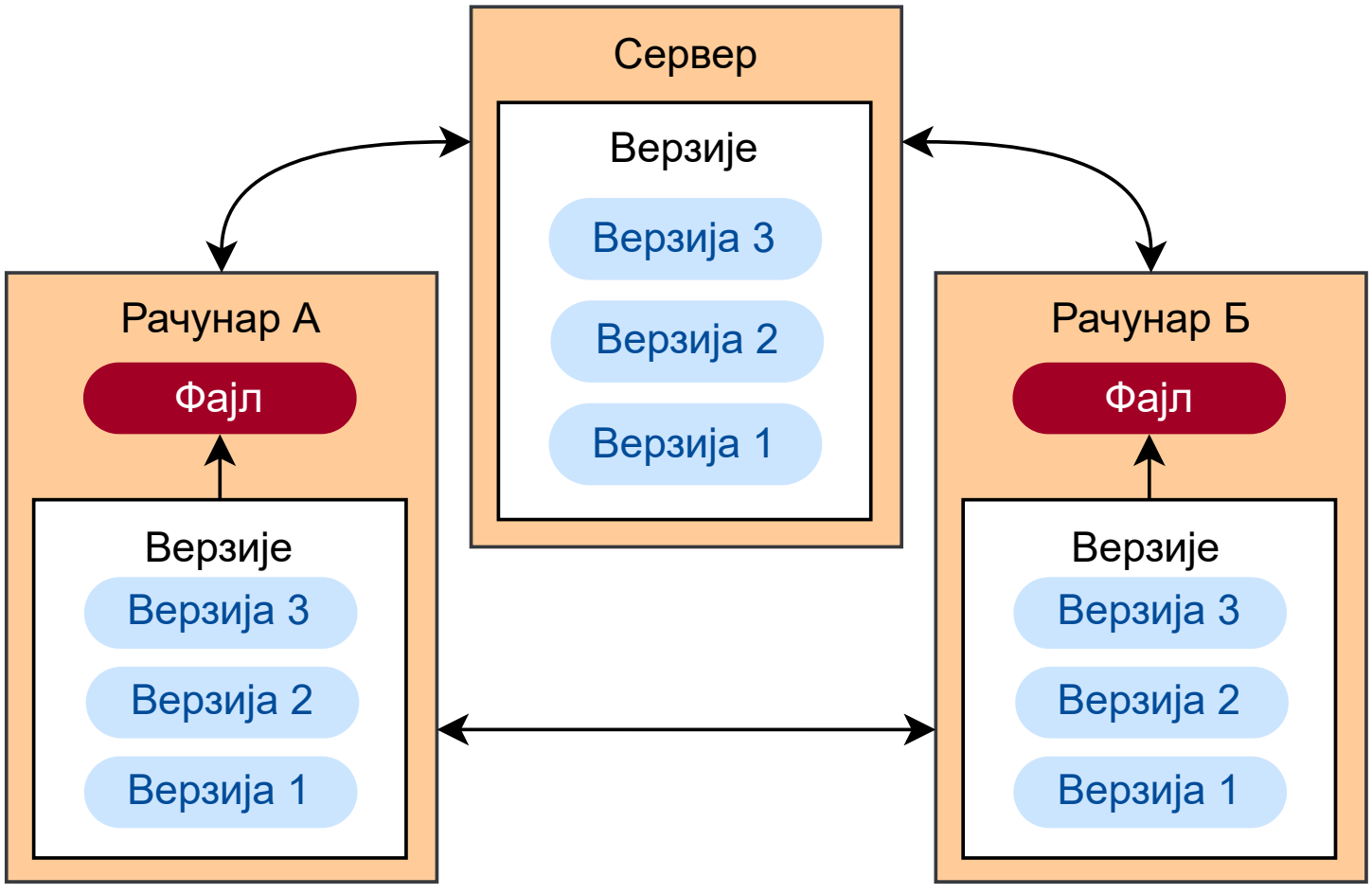
Увод

Дистрибуирани системи за контролу верзија (ДСКВ)

- код **дистрибуираних система за контролу верзија (ДСКВ)** (енг. Distributed Version control System (DVCS)) клијенти преузимају целовиту слику репозиторијума, а не само тренутан изглед фајлова
- ако неки од сервера падне, сваки клијент може да га опорави јер има целовиту верзију репозиторијума
- сваки ”клон” репозиторијума је резервна копија свих података
- омогућава рад на даљину са више репозиторијума

Увод

Дистрибуирани системи за контролу верзија (ДСКВ)



Гит

Историјски развој Гита

- идеја ДСКВ први пут је имплементирана као потреба у развоју Линукс[4] пројекта отвореног кода где се јављао велики број закрпа и имплементација нових својстава па је било тешко управљати верзијама
- 2002. године Линукс почиње да користи комерцијално решење који се зове Биткипер[5]
- 2005. године долази до разлаза између фирме која је одржавала Биткипер и Линукса па је тиме Биткипер престао да буде бесплатан софтвер

[4] (енг. Linux)
[5] (енг. BitKeeper)

Гит

Историјски развој Гита

- Линукс заједница је била приморана да развије своје ново решење отвореног кода које ће да унапреди Биткипер и отклони све његове недостатке у следећем смислу:
  - брзина,
  - једноставан дизајн,
  - снажна подршка за нелинеарни развој (на хиљаде паралелних грана)
  - потпуно дистрибуирани принцип
  - могућност руковања великим пројектима (нпр. Линукс језгро)
- тако настаје данашњи Гит који је изузетно брз и ефикасан у развоју великих софтверских пројеката кроз велики број паралелних грана

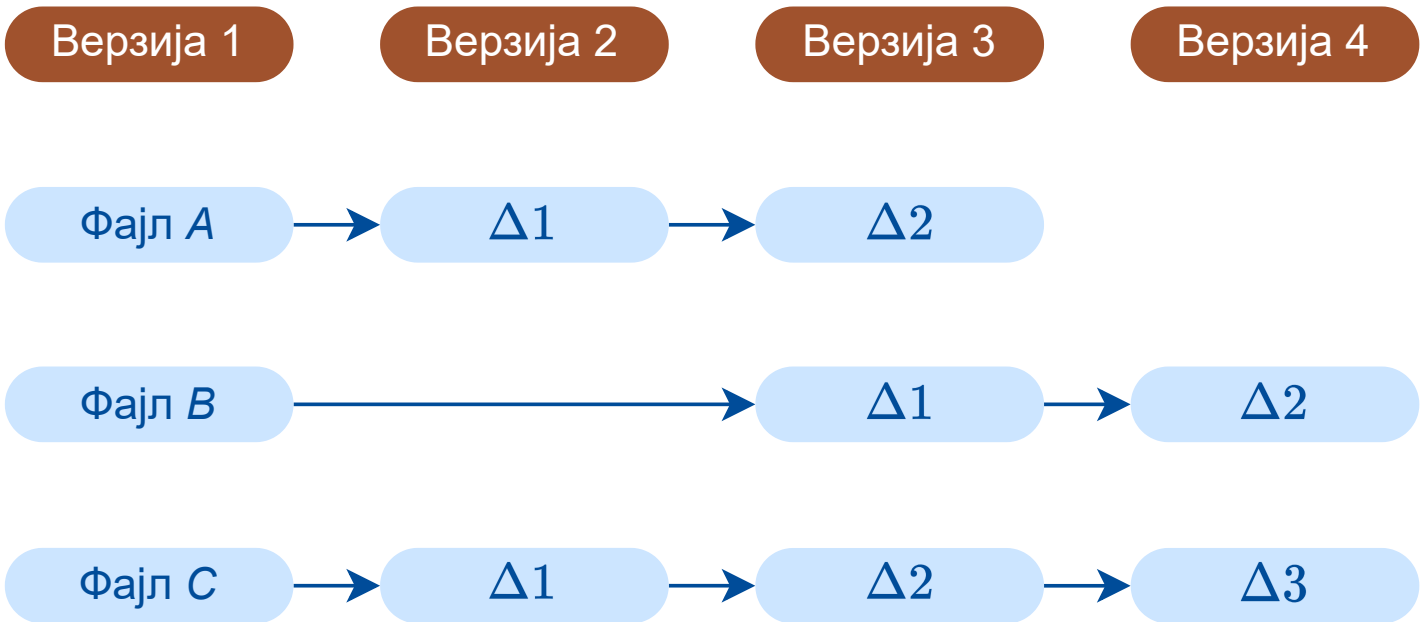
Гит

Историјски развој Гита

- главна разлика између Гита и СКВ система је начин на који Гит посматра податке
- СКВ системи контролишу верзије кода по **делта принципу**
- **делта принцип** подразумева да се током времена спремају фајлови са променама које су направљене на почетној верзији кода
- за разлику од СКВ-а, Гит користи принцип **тока снимака**

Гит

Делта принцип



Гит

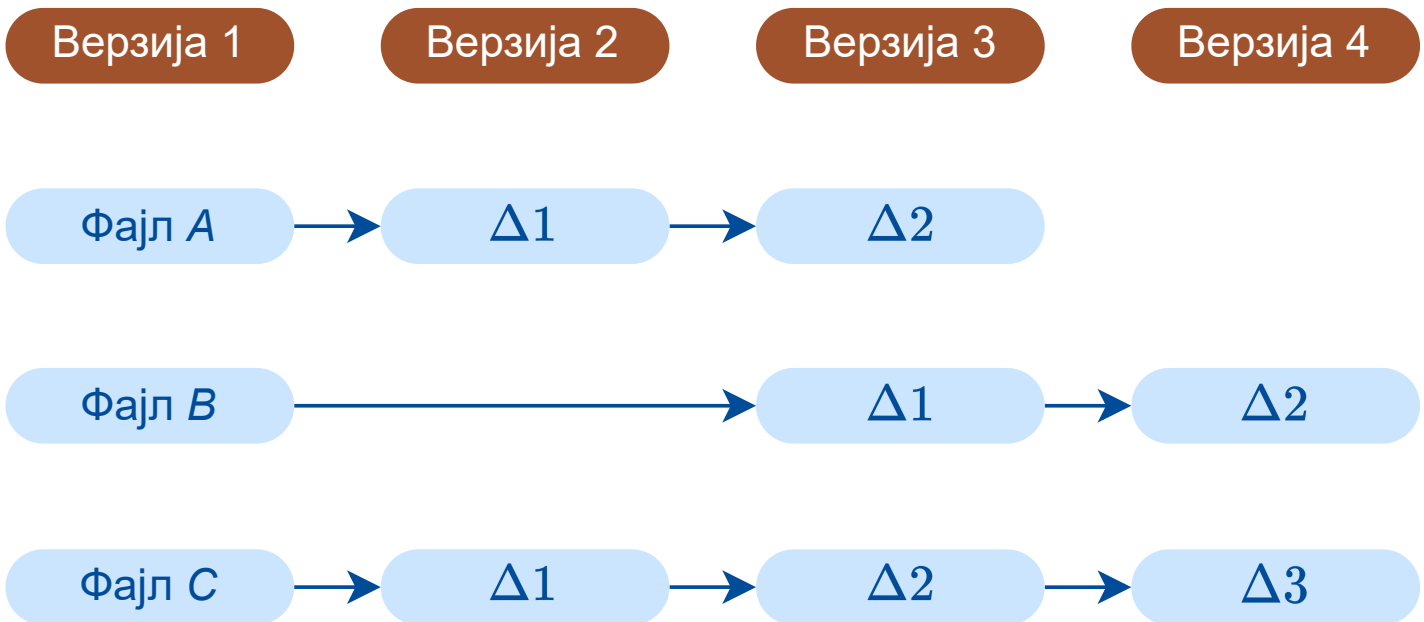
Ток снимака

- Гит посматра податке као **скуп снимака** (енг. snapshots) фајл система
- када се промена на фајлу потврди[6], односно сачува стање пројекта у Гит репозиторијуму, Гит узима слику **свих стања** ваших фајлова
- Гит **памти референце** на снимке
- због ефикасности, ако се фајл не промени, Гит не чува фајл поново, већ само везу на идентични фајл из претходног снимка
- због тога кажемо да је Гит користи принцип тока снимака

[6] комитује, (енг. commit)

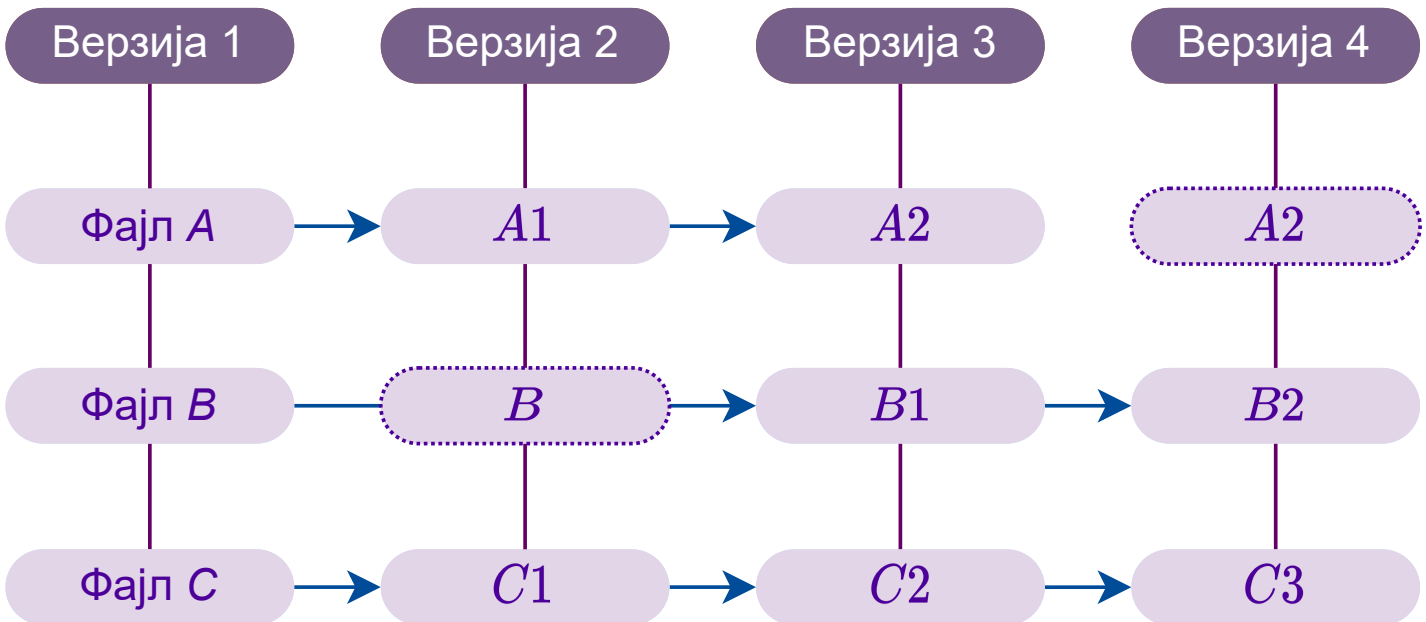
Гит

Ток снимака



Гит

Ток снимака



Гит

Локалност

- већина Гит операција изводе се на локалним фајловима без потребе ресурса на другим удаљеним рачунарима
- због те чињенице операције су изузетно брзе и флуидне
- Гит може да уради локално израчунавање историје без комуникације са сервером
- велика предност Гита је у томе што омогућује рад на пројекту локално без потребе везе са сервером
- направљене и потврђене[7] промене могу на крају да се пошаљу[8] на сервер

[7] (енг. commit)
[8] (енг. push)

Гит

Интегритет

- Гит одржава интегритет фајлова рачунањем **контролне суме** (енг. checksum) пре него што се они сачувају
- интегритет у овом смислу значи да је немогуће променити садржај било којег фајла ”без знања” Гита
- због интегритета је немогуће да се изгубе подаци у току транзита или да се они оштете
- механизам за имплементацију контролне суме је SHA-1 хеширање (енг. SHA-1 hash) у којем се рачуна стринг од 40 хексадецималних знакова (цифре 0-9 и слова a-f) на сонову садржаја фајла и структуре директоријума у Гиту (нпр. 31б9да65ффф52987аа493б52ф8123цд6даба0373)

Гит

Опозивост

- Гит само додаје податке, никад их не брише и не ажурира
- због тога не постоји операција која се не може опозвати
- промене је готово немогуће изгубити, поготово оног часа када су оне потврђене (енг. commit)

Гит

Три стања

- ово је најважније за разумевање како ради Гит
- Гит има три стања у којем могу да се нађу фајлови:
  - **измењено** - фајл је промењен али та промена није потврђена, односно, комитована, и спремљена у базу података коју Гит користи,
  - **припремљено** - фајл је промењен и означен да ће бити укључен у следеће комитовање,
  - **комитовано** или **потврђено** - промене су смештене у локалну базу података.

Гит

Три стања

- због овога имамо три главне секције или директоријума:
  - **радни директоријум** - одјављивање (енг. checkout) неке верзије пројекта, односно, достављање фајлова из Гит базе података (Гит директоријума) и смештање на диск где се они користе и мењају
  - **припремно подручје** - чине га фајлови који се налазе у Гит директоријуму и чувају информацију о томе шта ће бити укључено у следећем потврђивању[9]
  - **гит директоријум** - најважнији део Гит програма, место где Гит чува метаподатке пројекта и своју базу података, копира се када клонирамо репозиторијум са другог рачунара

[9] технички, ово је index

Гит

Ток рада с Гитом

- **измена фајлова** у радном директоријуму
- одабирамо само измене за које желимо да буду потврђене, другим речима, **припремамо измене**
- **потврђивање** верзије фајлова, која се налази у припреми, и трајно спремање њиховог снимка у Гит директоријум

Гит

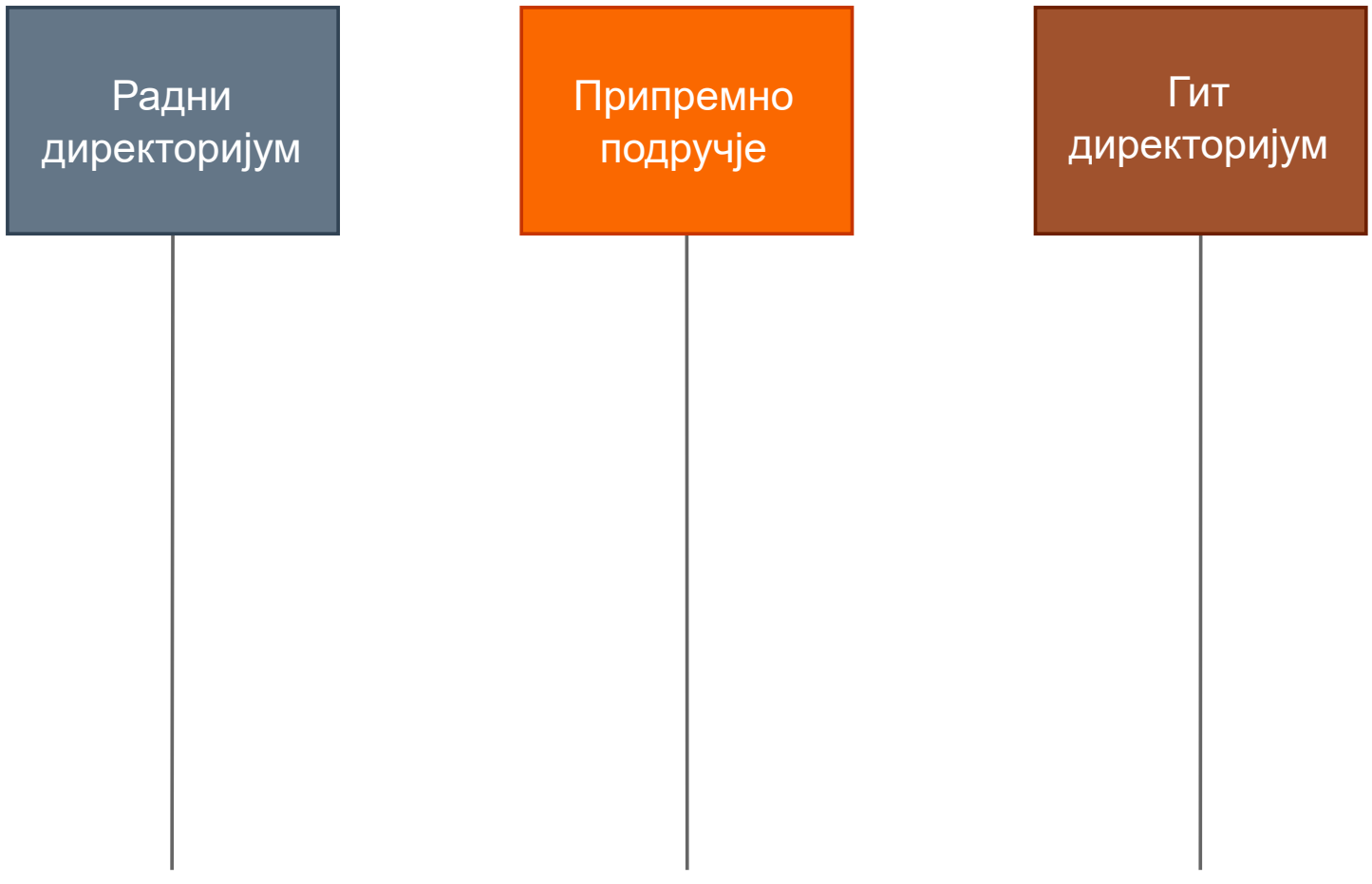
Ток рада с Гитом

- да резимирамо:
  - ако се нека верзија фајла налази у Гит директоријуму сматра се **потврђеном**[10],
  - ако је нека верзија фајла измењена и додата у припрему, кажемо да је **припремљена**,
  - ако је нека верзија промењена након одјављивања са Гит директоријума (односно након checkout -а), а није додата у припрему, кажемо да је **измењена**.

[10] (енг. commited)

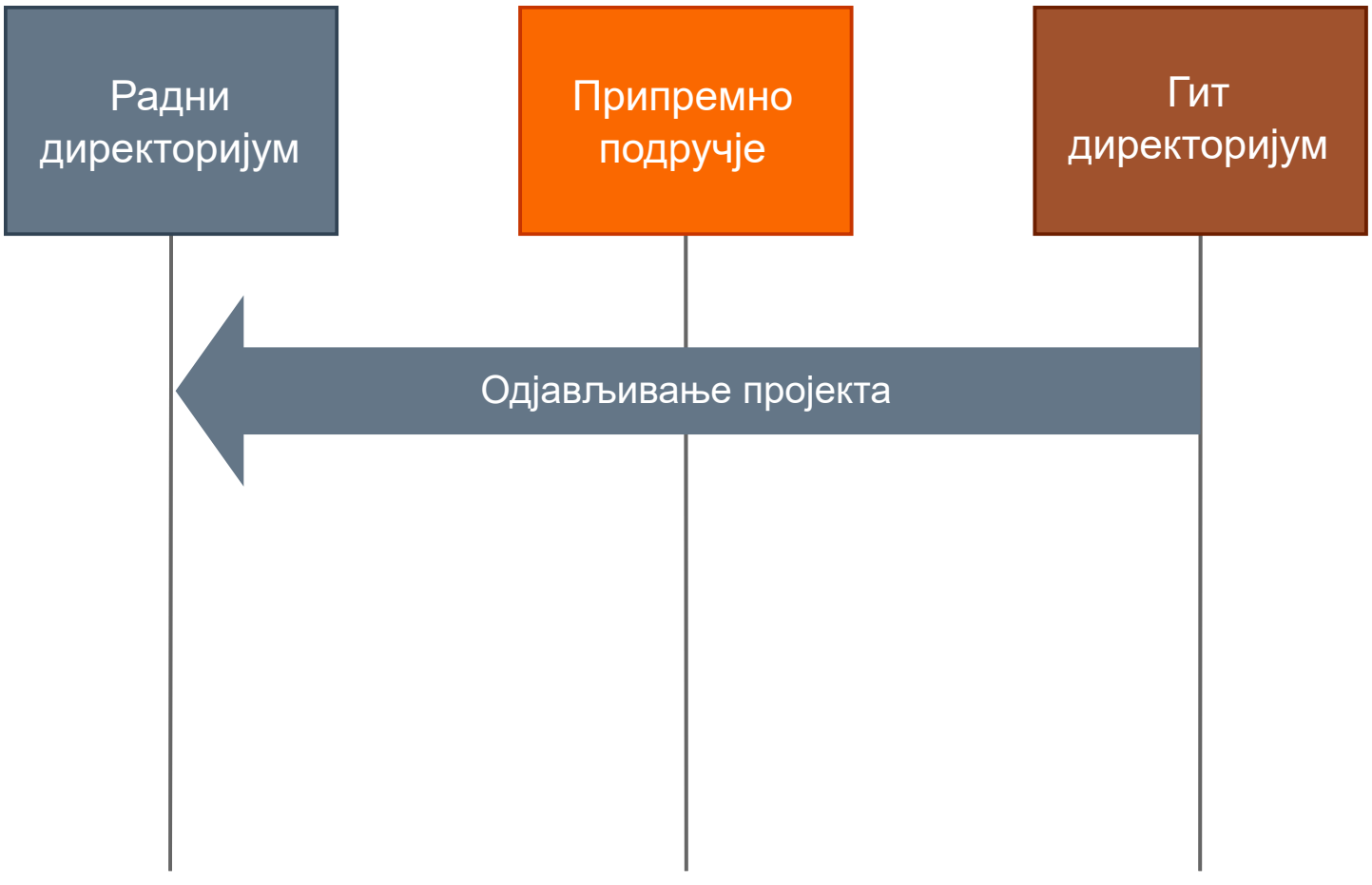
Гит

Ток рада с Гитом



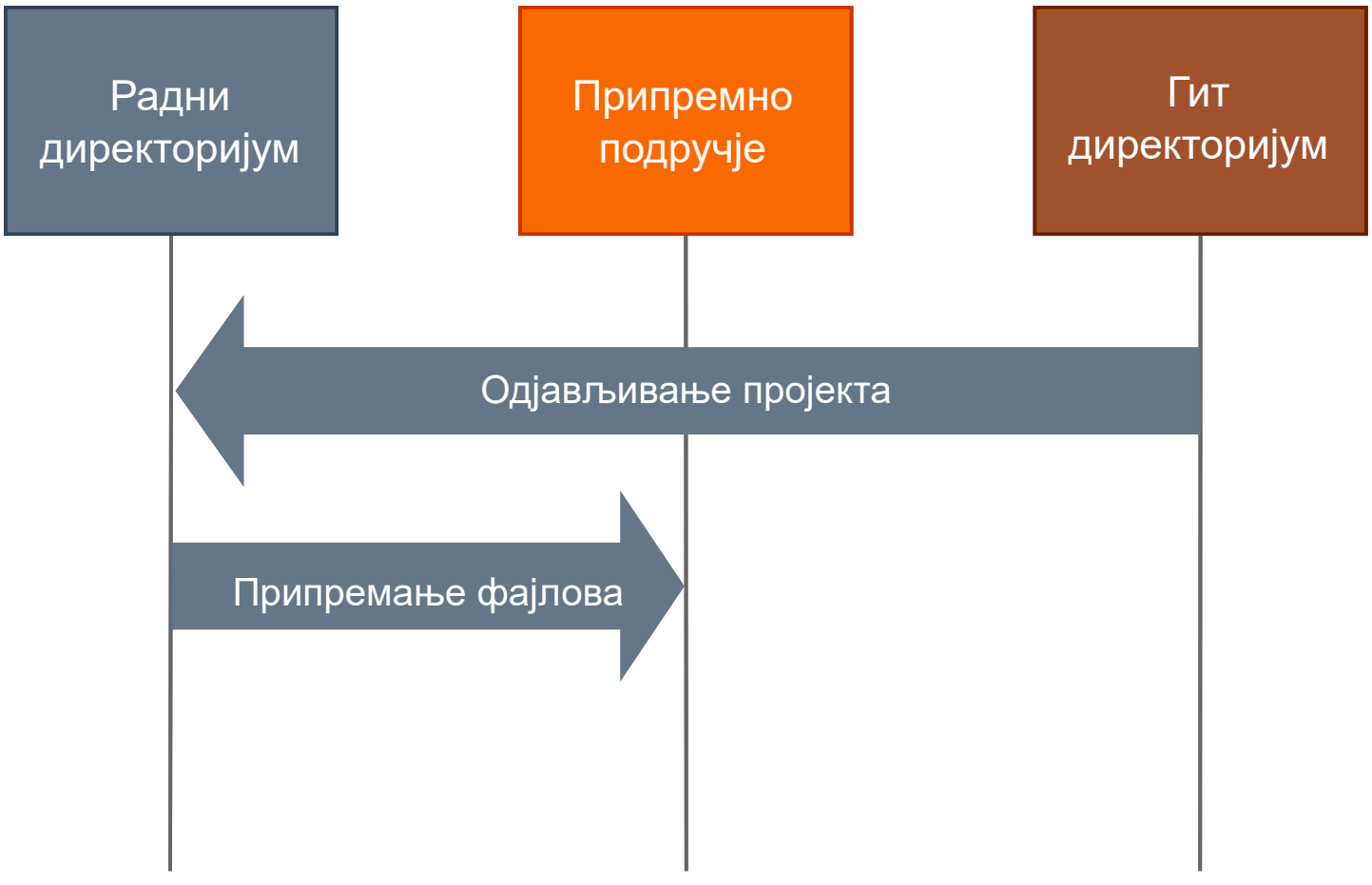
Гит

Ток рада с Гитом



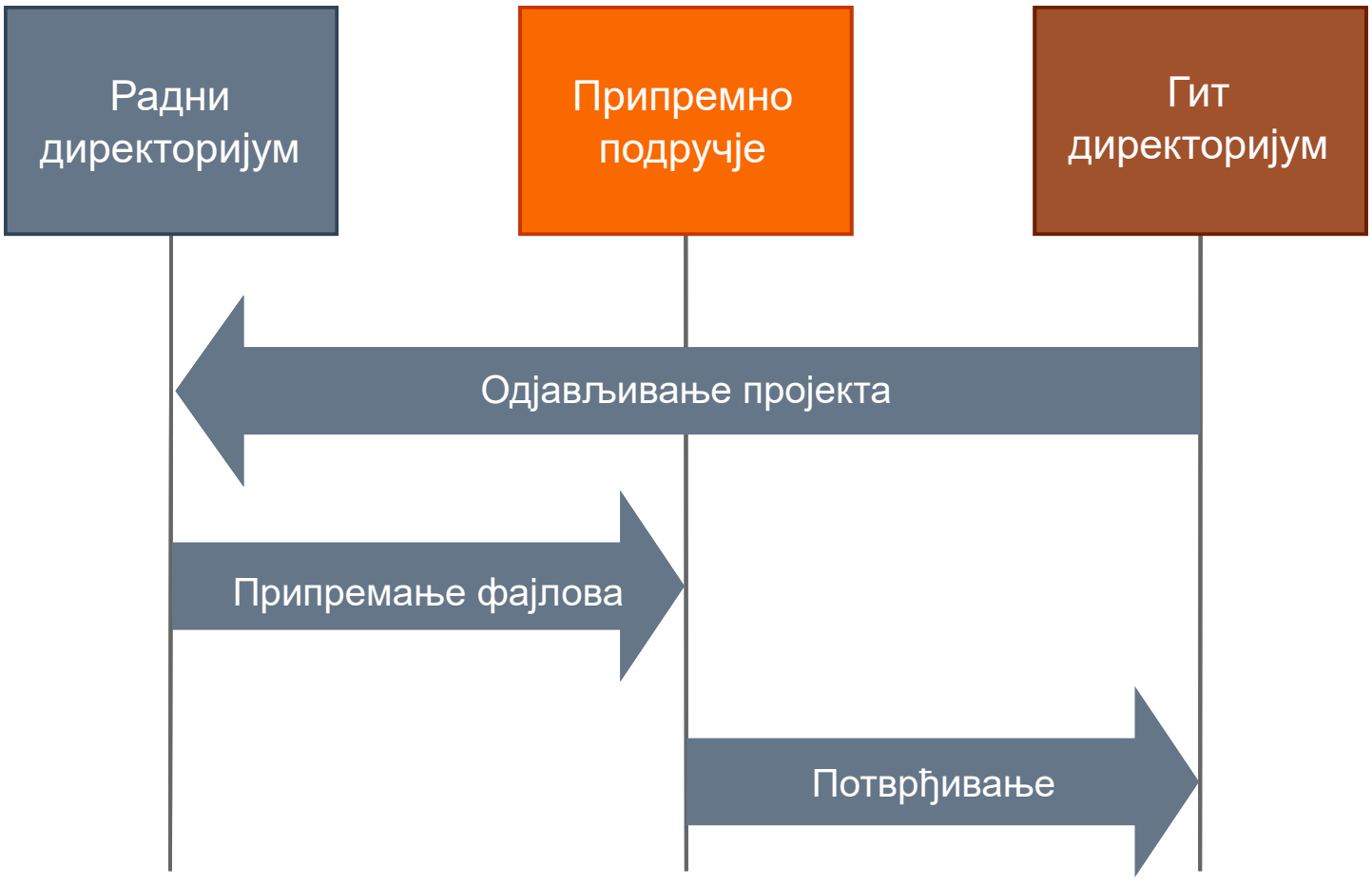
Гит

Ток рада с Гитом



Гит

Ток рада с Гитом



Конфигурисање Гита

Подешавања

- нека подешавања раде се само први пут,
- односе се на подешавање програма Гит на локалном рачунару,
- радимо на командној линији
- користимо наредбу `git config` на три нивоа:
  - **системском** - мења се садржај системског фајла за **све кориснике** и **све њихове репозиторијуме** наредбом `git config --system`
  - **корисничком** - мења се садржај фајлова `~/.gitconfig` или `~.config/git/config` само за **специфичног корисника**[11] и **све његове репозиторијуме** наредбом `git config --global`
  - **репозиторијумском** - мења се садржај фајла `.git/config` у Гит репозиторијуму **специфичног корисника** и **његовог специфичног репозиторијума** наредбом `git config --local`

[11] приметите да се овде мењају фајлови у корисничком директоријуму а не системском

Конфигурисање Гита

Подешавања

- репозиторијумски ниво је подразумеван и за његову примену морате да се налазите у том специфичном репозиторијуму
- сваки следећи ниво преклапа претходни
- на пример, кориснички ниво има већи приоритет од системског, а репозиторијумски има већи приоритет од корисничког

Конфигурисање Гита

Подешавања

- на оперативном систему Виндовс кориснички конфигурациони фајлови налазе се унутар `$HOME` директоријума који је углавном `C:\Users\$USER`
- локација системских конфигурационих фајлова зависи од верзије Виндовс оперативног система коју користите
- на новијим верзијама то је углавном `C:\ProgramData\Git\config`
- измене системског конфигурационог фајла врши се само покретањем наредбе `git config -f <naziv_fajla>` уз администраторске привилегије
- преглед тренутних подешавања види се покретањем наредбе `git config --list --show-origin`

Конфигурисање Гита

Дефинисање идентитета

- дефинисање идентитета подразумева постављање имејл адресе и имена корисника/програмера који ради на репозиторијуму
- ове конфигурације могу да се проведу на корисничком нивоу користећи опцију `--global`

```
git config --global user.name "Marko Markovic"
git config --global user.email
↪  marko.markovic@grf.bg.ac.rs
```

Конфигурисање Гита

Дефинисање подразумеваног уређивача текста

- односи се на подешавање подразумеваног уређивача[12] текста којег ће гит користити
- на оперативном систему Виндовс потребно је навести пуну путању до уређивача текста
- на настави користимо Visual Studio Code уређивача текста
- пример подешавања је следећи:

```
git config --global core.editor "'C:/Program
↪  Files/Microsoft VS Cod/bin/code' --wait"
```

[12] (енг. editor)

Конфигурисање Гита

Дефинисање подразумеване гране

- користећи наредбу `git init` приликом иницијализације новог Гит репозиторијума, ствара се подразумевана грана `master`
- подразумевана грана може да се промени, на пример, у `main`

```
git config --global init.defaultBranch main
```

Конфигурисање Гита

Приказ подешавања

- подешавања могу да се прикажу извршавањем наредбе која враћа листу **кључ-вредност** парова:

```
git config --list

init.defaultbranch=master
user.email=sjelic@grf.bg.ac.rs
user.name=Slobodan Jelic
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.origin.url=https://github.com/sjelic/wp.git
```

Конфигурисање Гита

Приказ подешавања

- могуће је видети и појединачна подешавања наводећи кључеве у наредби:

```
git config user.name

Marko Markovic
```

- уколико желимо да одредимо порекло вредности за поједини кључ наводимо опцију `--show-origin`

Конфигурисање Гита

Помоћ

- постоје три еквивалентна начина за тражење помоћи

```
git help <glagol>
git <glagol> --help
man git-<glagol>
```

- на пример, `git help config` или `git add -h`

Основе Гита

- ово поглавље даје најважније команде у рад с Гитом које су у највећој мери и довољне за успешно управљање кодом на програмерском пројекту
- садржи следеће аспекте:
  - конфигурација и иницијализација репозиторијум,а
  - почетак и престанак праћења фајлова,
  - припремање и потврђивање верзија, односно, промена,
  - игнорисање неких фајлова или типова фајлова,
  - поништавање грешака,
  - претрага историје пројекта и промене између потврђених верзија,
  - слање и повлачење фајлова на/са удаљени/удаљеног репозиторијума.

Основе Гита

Прављење Гит репозиторијума

- разликујемо два начина:
  - претварањем постојећег директоријума на локалном фајл систему у Гит репозиторијум,
  - клонирање постојећег Гит репозиторијума са неког другог места.
- исход обе опције је локални Гит репозиторијум, односно, Гит репозиторијум на локалној машини

Основе Гита

Иницијализација репозиторијума у постојећем директоријуму

- ово се односи на први начин претварања постојећег директоријума (који није под контролом верзија) у Гит репозиторијум
- потребно је да одете у тај директоријум користећи наредбу `cd` на оперативном систему Виндовс и Линукс

```
cd /home/user/app
```

- када се налазите унутар директоријума извршите наредбу:

```
git init
```

- овом наредбом нисмо започели праћење фајлова, него смо иницијализовали Гит директоријум стварањем поддиректоријума `.git` у директоријуму нашег пројекта
- праћење фајлова започињемо наредбом `git add`
- потврђивање фајлова извршавамо наредбом `git commit`
- више о овим наредбама када дођу на ред

Основе Гита

Клонирање постојећег директоријума

- у случају да желимо да допринесемо неком пројекту који се налази на удаљеном репозиторијуму, могуће је да користимо наредбу `git clone`
- исход ове наредбе показује концептуалну разлику у односу на ЦСКВ и даје основни допринос Гита који проистиче из принципа ДСКВ
- `git clone` наредба повлачи комплетну историју пројекта са удаљеног рачунара, а не само последњу верзију на којој ћемо радити
- овом наредбом се, дакле, клонира пројекат са целом својом историјом и локална копија може да послужи као резервна копија у случају пада Гит сервера

Основе Гита

Клонирање постојећег директоријума

- приликом покретања ове команде потребно је дати аргумент који представља URL удаљеног репозиторијума којег желимо да клонирамо
- на пример:

```
git clone https://github.com/libgit2/libgit2
```

- видимо да је исход клонирања стварање поддиректоријума `.git` у директоријуму пројекта који се створио извршавањем наредбе `git clone` где се сада налазе сви фајлови задње верзије који су спремни за рад

Основе Гита

Снимање промена у репозиторијуму

- иницијализацијом добијамо радни директоријум фајлова које Гит види
- често кажемо да их Гит ”прати”
- сваки фајл у радном директоријуму може да буде праћен[13] или непраћен[14]
- **праћени** фајлови су они фајлови који су били у последњем снимку и о којима Гит води рачуна
- праћени фајлови могу да буду: *неизмењени*, *припремљени* и *потврђени*

[13] (енг. tracked)
[14] (енг. untracked)

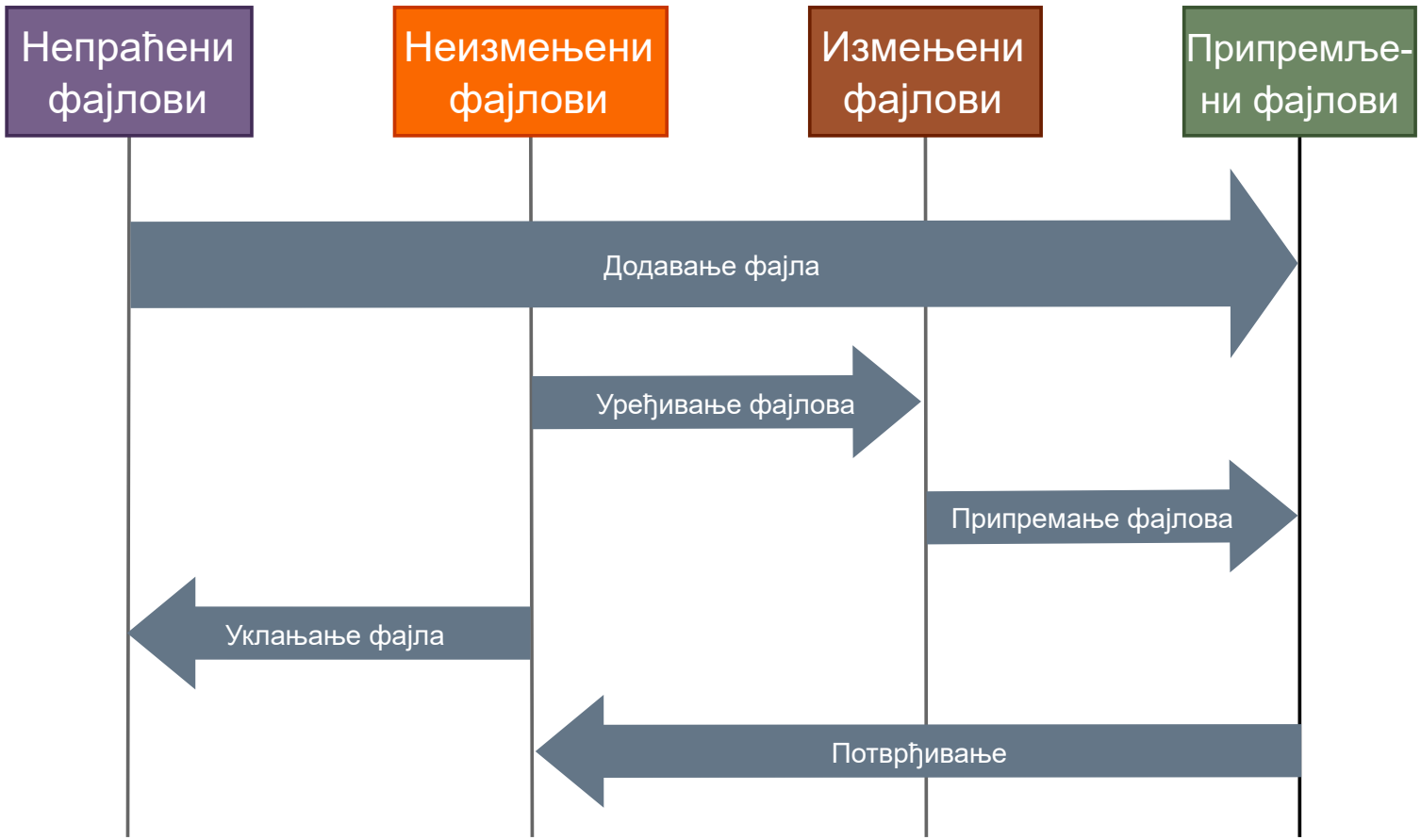
Основе Гита

Снимање промена у репозиторијуму

- **непраћени** фајлови су сви остали фајлови у радном директоријуму о којима Гит не води рачуна
- непраћени фајлови нису били у последњем снимку и нису у припреми
- након клонирања сви фајлови су иницијално праћени и неизмењени
- уређивањем фајлови постају измењени
- измењени фајлови се припремају, што представља одабир за потврђивање
- потврђивањем се спремају промене

Основе Гита

Снимање промена у репозиторијуму



Основе Гита

Провера статус фајлова

- статус фајлова се проверава наредбом `git status`
- на пример:

```
git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

- из резултата видимо да Гит пријављује како нема неизмењених фајлова
- сви фајлови су потврђени
- видимо да нема и непраћених фајлова јер би иначе они били пријављени

Основе Гита

Провера статус фајлова

- додајемо нови фајл README

```
echo 'My Project' > README
```

- провером статуса добијамо следећи резултат:

```
git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
(use "git add <file>..." to include in what will be
↪  committed)

README

nothing added to commit but untracked files present (use
↪  "git add" to track)
```

Основе Гита

Провера статус фајлова

- у претходном примеру видимо да је фајл `README` пријављен као непраћен јер га Гит није пронашао у претходној снимци репозиторијума
- Гит га неће аутоматски пратити или потврдити осим ако то експлицитно не нагласимо
- на тај начин се спречава аутоматско додавање нежељних или генерисаних фајлова у радном директоријуму

Основе Гита

Праћење нових фајлова

- уколико желимо да пратимо нови фајл потребно је да извршимо наредбу

```
git add <naziv_fajla>
```

- на пример, ако желимо да пратимо нови створени фајл `README` који је до сад био непраћен, извршавамо следећу наредбу

```
git add README
```

Основе Гита

Праћење нових фајлова

- након провере статуса репозиторијума добијамо следећи резултат:

```
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: README
```

- приметимо да порука `Changes to be committed:` каже како је фајл већ и припремљен јер је спреман за комитовање
- `git add` команда узима за аргумент путању до фајла или директоријума[15]

[15] ако додајемо директоријум, онда се рекурзивно додају и сви фајлови у том директоријуму

Основе Гита

Стејџовање измењених фајлова

- уколико променимо фајл који је већ праћен потребно је да га најпре припремимо ако желимо касније да потврђујемо промене
- претпоставимо да постоји фајл `CONTRIBUTING.md` којег смо већ раније пратили
- ако изменимо тај фајл и покренемо команду `git status` добијамо следећи резултат:

Основе Гита

Стејџовање измењених фајлова

```
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: README
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: CONTRIBUTING.md
```

Основе Гита

Стејџовање измењених фајлова

- припремање измењених фајлова који нису припремљени такође се покреће командом `git add`
- потребно је нагласити да се приликом извршавања команде `git add` припрема **тачно ону верзију фајла која је затечена у том тренутку** у радном директоријуму
- ако припремамо фајл наредбом `git add`, не потврдимо га, него га изменимо додатно (јер смо се сетили да смо нешто заборавили) и проверимо статус видећемо да ће се он појавити **и као припремљен и као неприпремљен**
- тиме се открива "права природа" наредбе `git add` која заправо не припрема фајл, него његове верзије
- претпоставимо а смо након припремања направили измене на фајлу `CONTRIBUTING.md` те погледајмо статус репозиторијума

Основе Гита

Стејџовање измењених фајлова

```
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: README
  modified: CONTRIBUTING.md

  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: CONTRIBUTING.md
```

Основе Гита

Стејџовање измењених фајлова

- видимо да се `CONTRIBUTING.md` појављује и као припремљен и као неприпремљен
- то је због тога што смо направили промене на фајлу након што смо извршили команду `git add`
- како бисмо припремили и последњу измену/верзију, потребно је опет да извршимо наредбу `git add`

Основе Гита

Кратак статус

- уколико је потребан сажети и прегледан статус Гит репозиторијума, могуће је користити команду за кратак статус додавањем опције `-s`, односно `--short`

```
git status -s
M README
MM Makefile
A lib/git.rb
M lib/simplegit.rb
?? LICENSE.txt
```

Основе Гита

Кратак статус

- постоји две колоне: лева и десна
- лева колона представља статус припремања, десна колона представља статус фајла у радном директоријуму
- `??` - нови фајлови који није праћен (не зна се његов статус припремања а ни статус у радном директоријуму)
- `A` - нови фајлови који су припремљени
- `M` - фајл који измењен у радном директоријуму али та промена није у припреми
- `M` - фајл је припремљен, односно, нема промена у радном директоријуму које нису припремљене
- `MM` - фајл има припремљене промене, али има и промене у радном директоријуму које још нису припремљене

Основе Гита

Игнорисање фајлова

- ово је изузетно користан принцип када желимо да на нивоу репозиторијума дефинишемо фајлове, групу фајлова и директоријуме које Гит игнорише, тј. не прати
- то су често различити помоћни фајлови који се генеришу у току развоја и не желимо да их додајемо у коначну верзију програмског кода
- у том случају постоји образац за писање фајла `.gitignore` који садржи листу фајлова и директоријума који се игноришу
- користе се `glob` обрасци као поједностављени регуларни изрази

Основе Гита

Игнорисање фајлова

- ако се образац почне са косом цртом / онда се избегава рекурзивно игнорисање (игнорисање свих поддиректоријума)
- ако образац заврши косом цртом / онда се игнорише директоријум
- образац може и да се негира навођењем знака `!`
- `*` - хвата један или више карактера
- `[abc]` хвата један од карактера у загради
- `?` - хвата један карактер1
- `[0-9]` - хвата једну од цифара 0-9
- две звездице се користе за хватање угњеждених директоријума

Основе Гита

Пример игнорисања

```
# ignoriše sve .a fajlove
*.a
# ali prati sve lib.a fajlove, iako ignoriše .a fajlove
!lib.a
# ignoriše samo TODO fajlove u tekućem direktorijumu ali
↪  ne i TODO fajlove u nekom poddirektorijumu npr.
↪  src/TODO
/TODO
# ignoriše sve fajlove u build direktorijumu
build/
```

Основе Гита

Пример игнорисања

```
# ignoriše sve .txt fajlove u doc direktorijumu, npr.
↪  doc/notes.txt, ali ne i .txt fajlove u
↪  poddirektorijumima od doc, npr. doc/server/arch.txt
doc/*.txt
# ignoriše sve .pdf fajlove u doc direktorijumu
↪  (uključujući i .pdf fajlove u njegovim
↪  poddirektorijumima)
doc/**/*.pdf
```

Основе Гита

Детаљан преглед припремљених и неприпремљених промена

- `git status` даје информативни преглед стања фајлова у припреми и у радном директоријуму, иако не даје детаље о томе **шта** је промењено
- промене се наводе у виду додатих и одузетих линија текстуалног фајла
- користећи команду `git diff` добијамо детаљан преглед неприпремљених промена
- користећи команду `git diff --staged` или `git diff --cached` добијамо детаљан преглед припремљених промена, односно промена које улазе у следећи комит

Основе Гита

Комитовање промена

- присетимо се да све неприпремљене промене неће бити укључен приликом следећег потврђивања, те да командом `git add` припремамо све промене које су до тад направљене у радном директоријуму
- командом `git status` добијамо све промене које ће "бити укључене у следећем потврђивању"
- командом `git commit` потврђују се све промене на фајловима које су припремљене

Основе Гита

Комитовање промена

- ако се изврши команда `git commit` без аргумената, отвара се задати уређивача текста за гит[16] у којем је неопходно уписати **поруку потврђивања**
- порука потврђивања може да се проследи и у инлајн режиму користећи заставицу/опцију `-m`:

```
git commit -m "Inicijalna promena"
```

- након потврђивања видљива је грана у којој је извршена потврда са SHA-1 контролном сумом, број измењених фајлова и статистика о броју додатих и обрисаних линија

[16] видљив у конфигурацијама позивом команде `git config --global core.editor`