Практикум из рачунарских алата

Вежбе 5

Исидора Грујић
isidora@uni.kg.ac.rs

Лазар Илић
lazar@uni.kg.ac.rs

Филип Милић
milicf@uni.kg.ac.rs

Катедра за електротехнику и рачунарство
Факултет инжењерских наука Универзитета у Крагујевцу

Крагујевац, 12. новембар 2024.

Садржај

Садржај

- Верзионисање је пракса у софтверском инжењерству која укључује контролисање, организовање и праћење различитих верзија фајлова, најчешће фајлова изворног кода.

Системи за верзионисање

- Верзионисање је пракса у софтверском инжењерству која укључује контролисање, организовање и праћење различитих верзија фајлова, најчешће фајлова изворног кода.
- Систем за верзионисање (en. *VCS*) је алат који аутоматизује верзионисање.

Зашто "VCS"

- Пружа могућност враћања фајла на неко претходно сачувано стање.

Зашто "VCS"

- Пружа могућност враћања фајла на неко претходно сачувано стање.
- Пружа могућност враћања читавог пројекта на неко претходно сачувано стање.

Зашто "VCS"

- Пружа могућност враћања фајла на неко претходно сачувано стање.
- Пружа могућност враћања читавог пројекта на неко претходно сачувано стање.
- Пружа увид у историју измена фајлова, што омогућује:

Зашто "VCS"

- Пружа могућност враћања фајла на неко претходно сачувано стање.
- Пружа могућност враћања читавог пројекта на неко претходно сачувано стање.
- Пружа увид у историју измена фајлова, што омогућује:
  - Увид у то ко је извршио измену која је проузроковала грешку,

Зашто "VCS"

- Пружа могућност враћања фајла на неко претходно сачувано стање.
- Пружа могућност враћања читавог пројекта на неко претходно сачувано стање.
- Пружа увид у историју измена фајлова, што омогућује:
  - Увид у то ко је извршио измену која је проузроковала грешку,
  - Увид у то када је измена извршена,

Зашто "VCS"

- Пружа могућност враћања фајла на неко претходно сачувано стање.
- Пружа могућност враћања читавог пројекта на неко претходно сачувано стање.
- Пружа увид у историју измена фајлова, што омогућује:
  - Увид у то ко је извршио измену која је проузроковала грешку,
  - Увид у то када је измена извршена,
  - Увид у то зашто је измена извршена.

Зашто "VCS"

- Пружа могућност враћања фајла на неко претходно сачувано стање.
- Пружа могућност враћања читавог пројекта на неко претходно сачувано стање.
- Пружа увид у историју измена фајлова, што омогућује:
  - Увид у то ко је извршио измену која је проузроковала грешку,
  - Увид у то када је измена извршена,
  - Увид у то зашто је измена извршена.
- Нешто опширнија листа разлога

Садржај

- *De facto* стандард за верзионисање,

- *De facto* стандард за верзионисање,
- Слободан софтвер, коришћен у развоју и одржавању Линукс језгра.

Објекти

- Git рукује са три типа објекта:

Објекти

- Git рукује са три типа објекта:
  - Blob (Фајл)
  - Tree (Директоријум)
  - Commit (Снимак)

Објекти

- Git рукује са три типа објекта:
  - Blob (Фајл)
  - Tree (Директоријум)
  - Commit (Снимак)
- Пре него што наставимо са моделом података осврнућемо се на хеш вредности.

Хеш

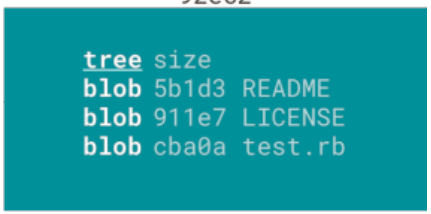
- Механизам који git користи за означавање објеката.
- Вредност од 40 хексадецималних цифара која се израчунава на основу садржаја фајлова и директоријума.
- Git све складишти у бази података користећи хеш вредности, не називе фајлова.
- Git користи SHA-1 алгоритам за израчунавање хеш вредности.

Blob

- У терминологији Git-а Blob представља фајл, низ бајтова. Blob је најосновнија јединица за складиштење у Git бази података.

Tree

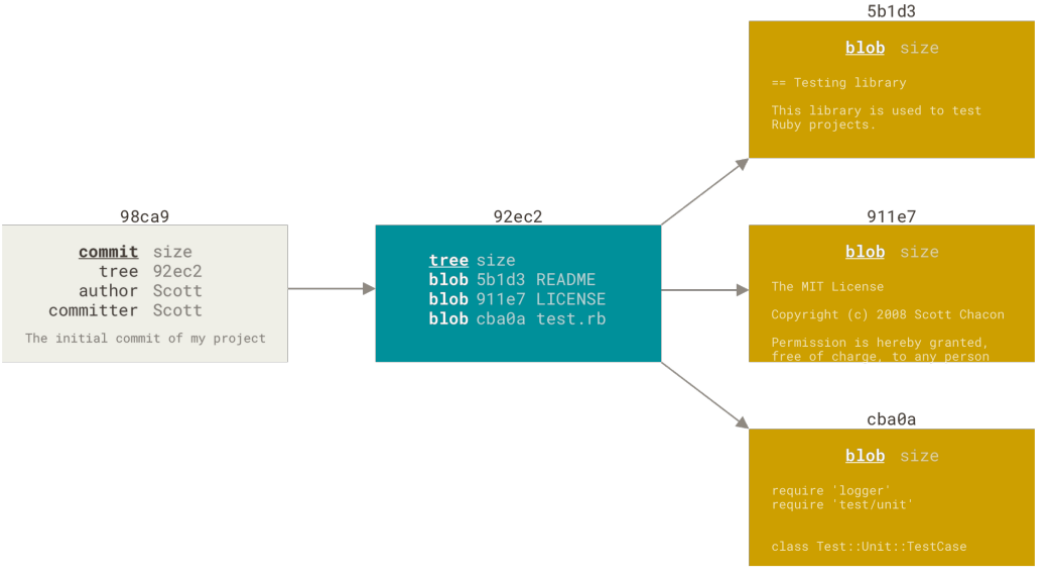
- Стабло представља директоријум. Стабло може да садржи друга стабла и Blob-ове. Корено стабло (root tree) представља структуру која ће бити сачувана када се направи снимак. Корено стабло садржи посебан скривени директоријум са називом *.git*.



Commit

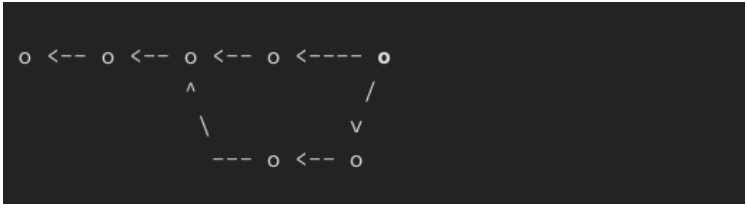
- Commit је снимак тренутног садржаја радног директоријума. Commit је дефинисан хеш вредношћу, поруком, именом аутора, временском ознаком, показивачем на корено стабло...

Commit пирмер



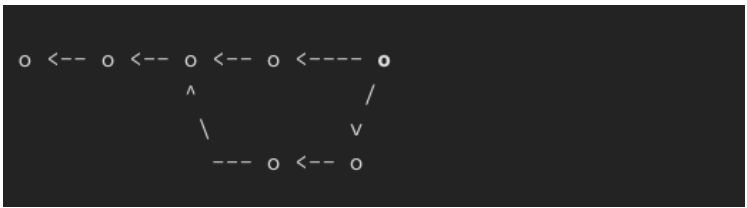
Структура историје

- Сваки commit чува хеш вредности родитељских commit-ова.



Структура историје

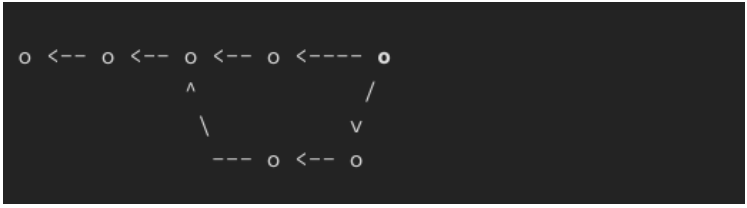
- Сваки commit чува хеш вредности родитељских commit-ова.



- Зашто не само једног родитеља?

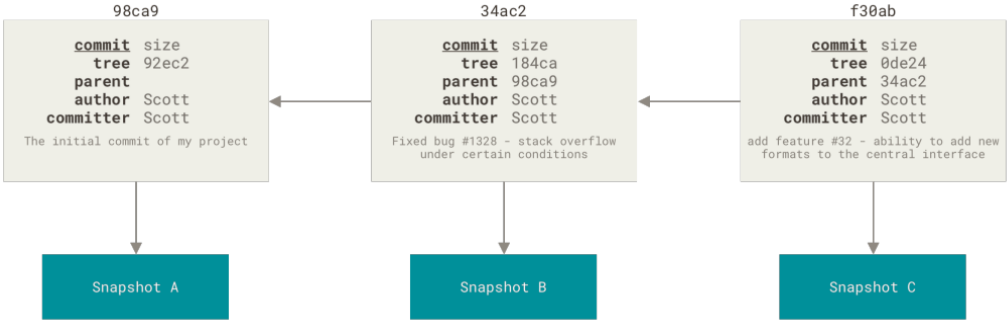
Структура историје

- Сваки commit чува хеш вредности родитељских commit-ова.



- Зашто не само једног родитеља?
- Сазнаћемо када се упознамо са гранама.

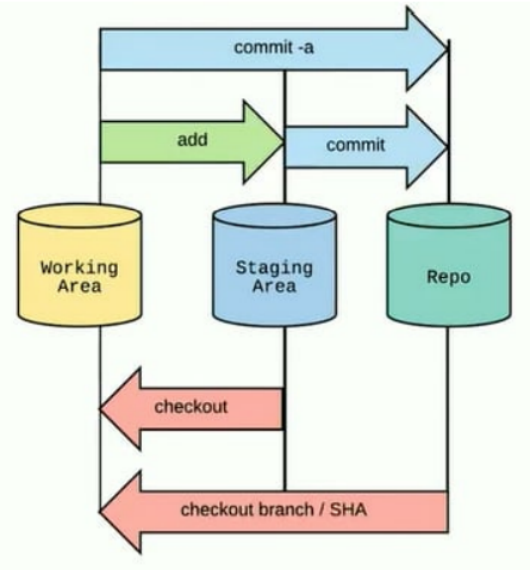
Пример историје



Staging Area

- Индекс, или Staging Area, предстаља део интерфејса Git-а. Индекс омогућује селекцију промена које ће бити део следећег commit-а.

Типичан ток рада

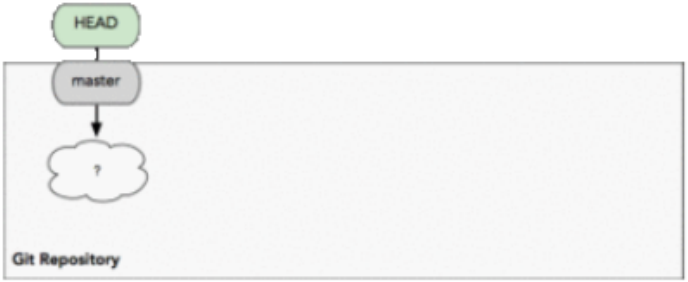


Садржај

Корак 1

- Креирати директоријум "демо" и у њему иницијализовати репозиторијум.
- Корисне команде: *git init*, *git status*.
- Направити фајл са називом file.txt.

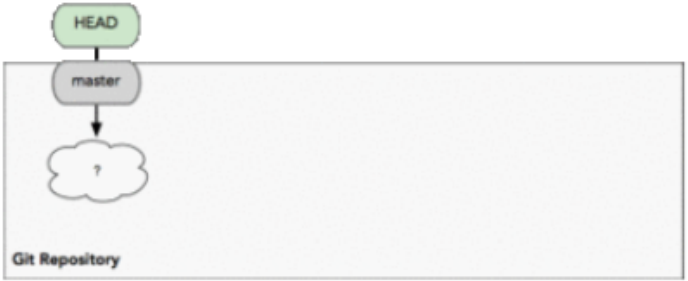
Стање 1


HEAD
master
?
Git Repository
HEAD
Index
Working Directory
v1
file.txt

Корак 2

- Додати фајл у индекс.
- Корисне команде: *git add*, *git status*, *git help*

Стање 2

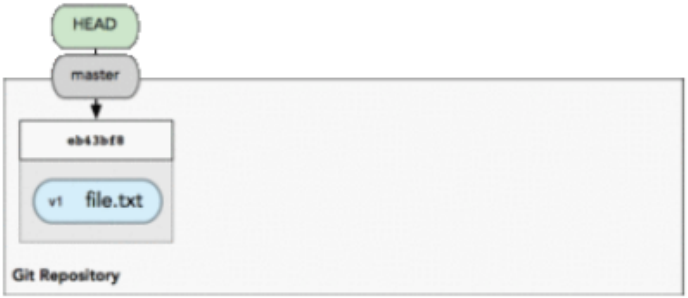


git add

Корак 3

- Направити снимак репозиторијума.
- Корисне команде: *git commit*, *git log*, *git status*

Стање 3

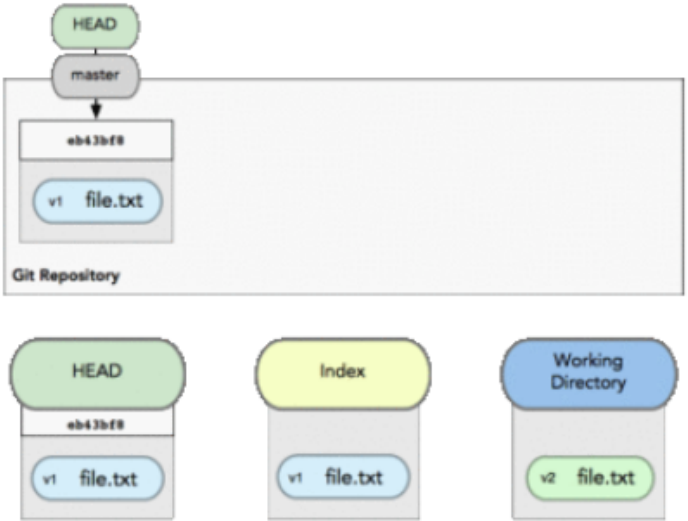


git commit

Корак 4

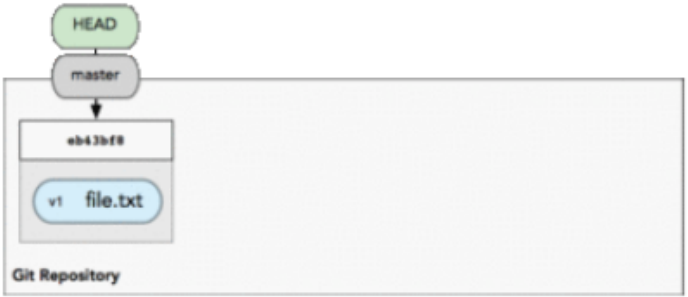
- Променити садржај фајла.

Стање 4


HEAD
master
eb43bf8
v1 file.txt
Git Repository
HEAD
eb43bf8
v1 file.txt
Index
v1 file.txt
Working Directory
v2 file.txt
edit file

Корак 5

- Додати фајл у индекс.
- Корисне команде: *git add*, *git status*, *git help*

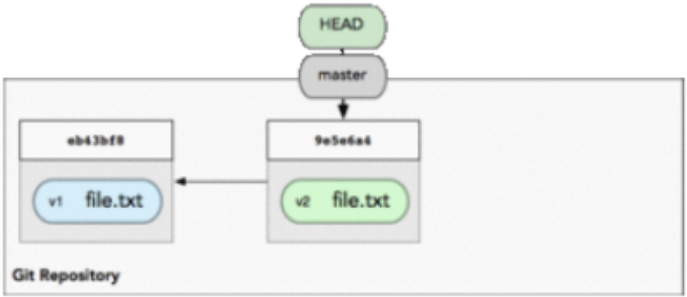

HEAD
master
eb43bf8
v1 file.txt
Git Repository


HEAD
eb43bf8
v1 file.txt
Index
v2 file.txt
Working Directory
v2 file.txt

git add

Корак 6

- Направити снимак репозиторијума.
- Корисне команде: *git commit*, *git log*, *git status*
- *git log - -all - -graph - -decorate* за лепши приказ.


HEAD
master
eb43bf8
9e5e6a4
v1 file.txt
v2 file.txt
Git Repository


HEAD
9e5e6a4
v2 file.txt
Index
v2 file.txt
Working Directory
v2 file.txt

git commit

Садржај

Референце

- Git омогућава додељивање назива објектима, како је људима згоднији рад са називима него са хексадецималним записом. Референца мапира назив на неку хеш вредност.

Референце

- Git омогућава додељивање назива објектима, како је људима згоднији рад са називима него са хексадецималним записом. Референца мапира назив на неку хеш вредност.
- Битне референце у git-у су гране, remote-ови и HEAD.

Референце

- Git омогућава додељивање назива објектима, како је људима згоднији рад са називима него са хексадецималним записом. Референца мапира назив на неку хеш вредност.
- Битне референце у git-у су гране, remote-ови и HEAD.
- Гране су тип референци које показују на commit-ове у историји.

Референце

- Git омогућава додељивање назива објектима, како је људима згоднији рад са називима него са хексадецималним записом. Референца мапира назив на неку хеш вредност.
- Битне референце у git-у су гране, remote-ови и HEAD.
- Гране су тип референци које показују на commit-ове у историји.
- HEAD је тип референце која показује на грану, уједно и commit, која представља тренутни радни директоријум.

Референце

- Git омогућава додељивање назива објектима, како је људима згоднији рад са називима него са хексадецималним записом. Референца мапира назив на неку хеш вредност.
- Битне референце у git-у су гране, remote-ови и HEAD.
- Гране су тип референци које показују на commit-ове у историји.
- HEAD је тип референце која показује на грану, уједно и commit, која представља тренутни радни директоријум.
- О remote-овима више у каснијој секцији.

Садржај

Корак 1

- Додати фајл *demoProg.py* у "демо" фолдер,
- Додати фајл у индекс,
- Снимити репозиторијум.

Корак 2

- Направити грану функционалност1,
- Корисне команде: *git branch*, *git checkout*, *git log*
- Пребацити се на грану функционалност1 (уколико до сада нисте),
- Преправити прву функцију у *demoProg.py*,
- Додати фајл у индекс,
- Снимити репозиторијум.

Корак 3

- Вратити се на master грану,
- Корисне команде: *git checkout, git log*
- Направити грану функционалност2,
- Пребацити се на грану функционалност2 (уколико до сада нисте),
- Преправити другу функцију у *demoProg.py*,
- Додати фајл у индекс,
- Снимити репозиторијум.

Корак 4

- Спојити мастер и функционалност2.
- Корисне команде: *git merge*,*git log*

Корак 5

- Спојити мастер и функционалност1.
- Корисне команде: *git merge*,*git log*

Корак 6

- Уклонити гране функционалност1 и функционалност2.
- Корисне команде: *git branch -d*

Корак 7

- Направити грану новаФункционалност,
- Пребацити се на грану новаФункционалност (уколико до сада нисте),
- Додати нову функцију у *demoProg.py*,
- Додати фајл у индекс,
- Снимити репозиторијум.

Корак 8

- Вратити се на master грану,
- Изменити неку од постојећих функција,
- Додати фајл у индекс,
- Снимити репозиторијум.

Корак 9

- Вратити се на новаФункционалност грану,
- Извршити rebase са мастер граном,
- Корисне команде: *git rebase*,*git log*
- Вратити се на мастер грану,
- Спојити мастер са новаФункционалност граном,
- Обрисати новаФункционалност грану.

Садржај

Remote

- Да би сте сарађивали на било ком Git пројекту морате баратати са git remote-овима. Удаљени (remote) репозиторијуми су верзије пројекта које су смештене на некој платформи или серверу.

Remote

- Да би сте сарађивали на било ком Git пројекту морате баратати са git remote-овима. Удаљени (remote) репозиторијуми су верзије пројекта које су смештене на некој платформи или серверу.
- Најпопуларнија интернет платформа за складиштење Git репозиторијума је GitHub.

Remote

- Да би сте сарађивали на било ком Git пројекту морате баратати са git remote-овима. Удаљени (remote) репозиторијуми су верзије пројекта које су смештене на некој платформи или серверу.
- Најпопуларнија интернет платформа за складиштење Git репозиторијума је GitHub.
- За потребе наредне вежбе потребно је направити GitHub налог и подесити SSH аутентификацију.

Садржај

Вежба 3

- Направити GitHub налог и подесити SSH аутентификацију (уколико већ нисте; потребно је направити кључ на вашој машини, а потом га повезати са вашим Github налогом).
- Форковати репозиторијум.
- Клонирати сопствену верзију репозиторијума (команда git clone).
- Провежбати, односно разумети садржај репозиторијума, закључно са целином - *Updating the Dev Environment with remote changes* (дакле станете код поднаслова *Cherry-picking*; наравно, амбициозни се охрабрују да "истерају" до краја).