Advanced Web Programming

Yuan Wang

2019 spring

Lecture 12

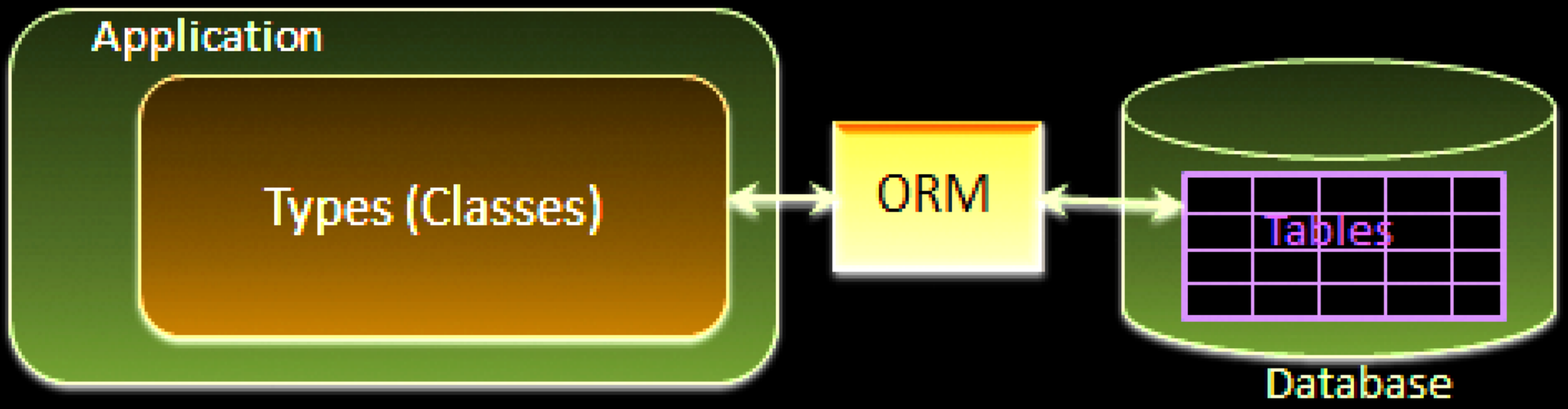
Ruby - Sinatra - RECAP

SQLite: command line utility: sqlite3

```
> sqlite3 database-file
sqlite> CREATE TABLE table-name (
   . . . >      col1 varchar(10) primary key,
   . . . >      col2 text,
   . . . >      col3 real );
sqlite> INSERT INTO table-name
    . . .> (col1, col2, col3) values
    . . .> (“10”, “yuan”, “wang”);
sqlite> DELETE FROM table-name
    . . .> where col1==“10”;
sqlite> .tables
sqlite> .schema table-name
sqlite> .help
sqlite> .exit
```

Ruby - Sinatra - Database

ORM is mapping table to your Ruby class.



Ruby - Sinatra - RECAP

ORM - Object Relational Mapper

- DataMapper
define a class to represent table (model)

```
class Song
  include DataMapper::Resource   # include Resource mixin
  property :id, Serial
  property :title, String
  property :lyrics, Text
  property :length, Integer
  property :released, Date
end
DataMapper.finalize
DataMapper.setup(:default,“sqlite3://#{Dir.pwd}/song.db”)
Song.auto_migrate!
```

Ruby - Sinatra - RECAP

```
song = Song.new
song.save
song.title = “My Way”
song.lyrics = “this is my way”
song.length = “323”
song.released = Date.new(1969)
song.save
Song.create(title: “New York New York”,
            lyrics: “new york, new york”,
            length: 210,
            released: Date.new(1960))
Song.all
Song.get(id)    # get the object with id
Song.get(1)
Song.first      # first object
Song.last       # last record
song = Song.first(title: “My Way”)
Song.last.destroy
```

Ruby - Sinatra - RECAP

Folder structure

app/

main.rb
song.rb
song.db

two more files

views/

layout.erb
home.erb
about.erb
contact.erb
not_found.erb
songs.erb
new_song.erb
show_song.erb
song_form.erb
edit_song.erb
style.scss

bunch of more views

public/

images/

fsinatra.jpg
logo.png

css/

style.css

Ruby - Sinatra

Today's topics

Configuration

Session and Logon

Deploying the application

Ruby - Sinatra

Environments

Are values used for specifying different stages of your application development.

there are 3 predefined environments:

- development (default)
- test
- production

Ruby - Sinatra

Environments - how to set

use “-e” flag when start the server

for example

```
$ ruby myapp.rb -e production
```

OR use “set” method in your program:

```
set :environment, :production
```

Ruby - Sinatra

Environments - how to check

to check what environment is in, use these methods:

development? (default)
production?
test?

for example, you can do this:

require ‘sinatra/reloader’ if development?

this means that at development stage, do not need to restart server for any code modifications

Ruby - Sinatra

Configure block

```
configure do
    # configuration options
end
```

using configure block does not make any difference. it is just a way to make your code better readable.

- this will only run once.
- use as many configure blocks as you like.
- can be placed at any spot in the code.
- by convention, use one block and placed at the start of the file.

To set environment:

```
configure do
    set :environment :development
end
```

Ruby - Sinatra

Example:

```
configure do
    set :public_folder, ‘/static’
    set :views, “/templates”
    set :static, false    # not check public folder for static file
    set :root, “main_application_folder”
    set :port, 1234      # change the default 4567
    set :show_exceptions,  false
    set :username, “yuan”              # custom value
end
```

note:
- “set :static, false”

is equivalent to:

“disable :static”

- To access setting: settings.username

example:

```
puts settings.username    # => yuan
puts settings.environment
```

Ruby - Sinatra

Configuration block with environment parameters

```
configure :development do
    DataMapper.setup(:default,
    “sqlite3://#{Dir.pwd}/development.db”)
end
```

Run this block if environment was set to “development”

Ruby - Sinatra

Configuration block with environment parameters

```
configure :development, :test do
    DataMapper.setup(:default,
      "sqlite3://#{Dir.pwd}/development.db")
end
```

for both development and testing

```
configure :production do
    DataMapper.setup(:default,
        ENV['DATABASE_URL'])
end
```

environment variable on deployment server

Ruby - Sinatra

Sessions

Ruby - Sinatra

Sessions

Sinatra uses session hash to save session information.

Session is disabled by default.

To enable it:

```
enable :sessions
```

stores all data in a cookie actually

To remember session values, use session[ ]:

```
session[:keyname] = “value”
```

Ruby - Sinatra - Session

Example:

```
enable :sessions

get '/start' do
  session[:message] = 'Hello World!'
  redirect to('/another')
end

get '/another' do
  session[:message]                    # => 'Hello World!'
end
```

save “message” information in session variable

“message” information is available on other page (view)
note：it is another “unrelated” request!
(message information is supposed to be lost without session)

Ruby - Sinatra - Session

Example:

```
enable :sessions

get '/set/:name' do
  session[:name] = params[:name]
  redirect to('/another')
end

get '/another' do
  "hello, session[:name]"
end
```

save the "name" info sent from client into session variable

note: "to" is a method (alias to uri)
to('/another') = uri('/another')
will create complete URL,
then
redirect(URL) will use complete URL to redirect

so: redirect to('/another') is actually:
redirect(to('/another'))

"name" info is available on other page through session variable

Ruby - Sinatra

Sessions -
add login for “Song” application

```
configure do
    enable :session
    set :username, “yuan”
    set :password, “newnew”
end
```

Ruby - Sinatra

Sessions - add login for “Song” application

```
get ‘/login’ do
    slim :login
end

post ‘/login’ do
    if params[:username] == settings.username &&
     params[:password] == settings.password

      session[:admin] = true
      redirect to (‘/songs’)
    else
      slim :login
    end
end
```

login request, return login page (view)

check if login data match the values in settings

mark as logged in

show song list page

if not match, send back the login page

Ruby - Sinatra

Sessions - add login for “Song” application

add login page (view):

login.slim

```
form action=“/login” method=“post”
  input#username name=“username”
  input#password type=“password” name = “password”
  input type=“submit” value=“log in”
```

Ruby - Sinatra

Sessions - add login for “Song” application

use session[:admin] value to protect some of the routes:
for example:

```
get ‘/songs/new’ do
    halt(401, ‘Not Authorized’) unless session[:admin]
    @song = Song.new
    slim :new_song
end
```

halt - stop handling request in the route handler. send special response back.
example:

```
halt “can you see this?”

halt erb(:errorpage)
```

Some other routes also need to be protected like:
edit, delete, create.

Ruby - Sinatra

Sessions - add login for “Song” application

To destroy session:

session.clear

for example

```
get ‘/logout’ do
   session.clear
   redirect to (‘/login’)
end
```

Ruby - Sinatra

Deploying the application

Ruby - Sinatra - Deploy

Deploying the application:

- To actually put our application on real sever to be accessed by the world

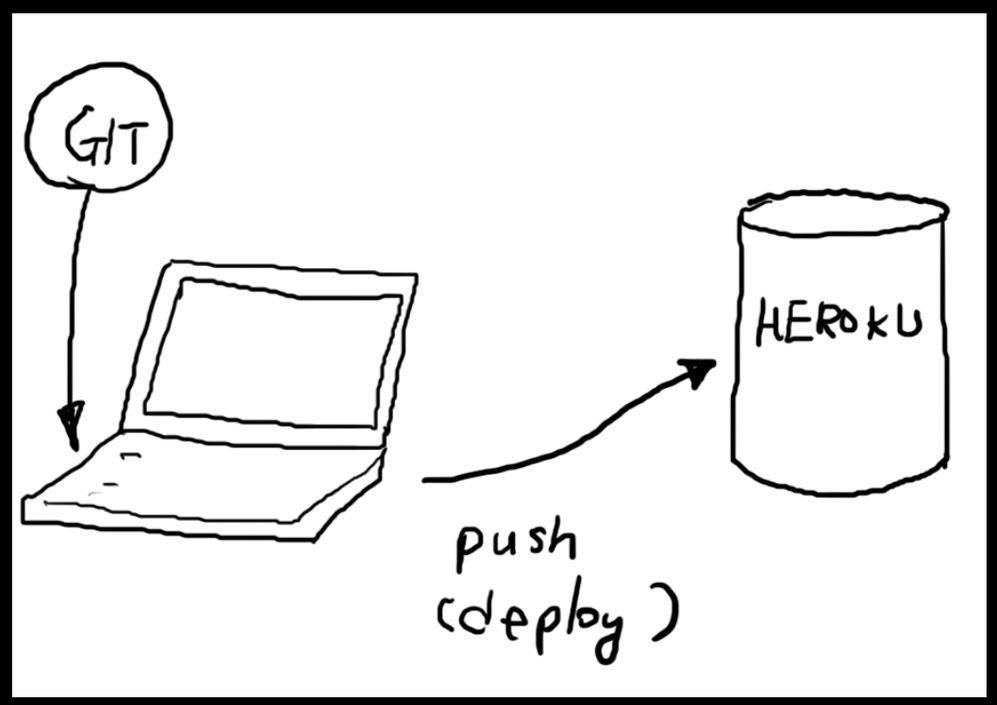
We will be using Heroku cloud service to host the application

Ruby - Sinatra - Deploy

Deploying the application:

First, create Git repository on our machine for our application

Then, push our application on to Heroku server



Ruby - Sinatra - Deploy

Heroku
a cloud PaaS (Platform as a Service)

Yukihiro "Matz" Matsumoto (the Ruby creator) is the chef designer of the company

Support PostgreSQL database as standard.

website: heroku.com

Ruby - Sinatra - Deploy

Heroku

$brew tap heroku/brew && brew install heroku
(install homebrew first)

Or

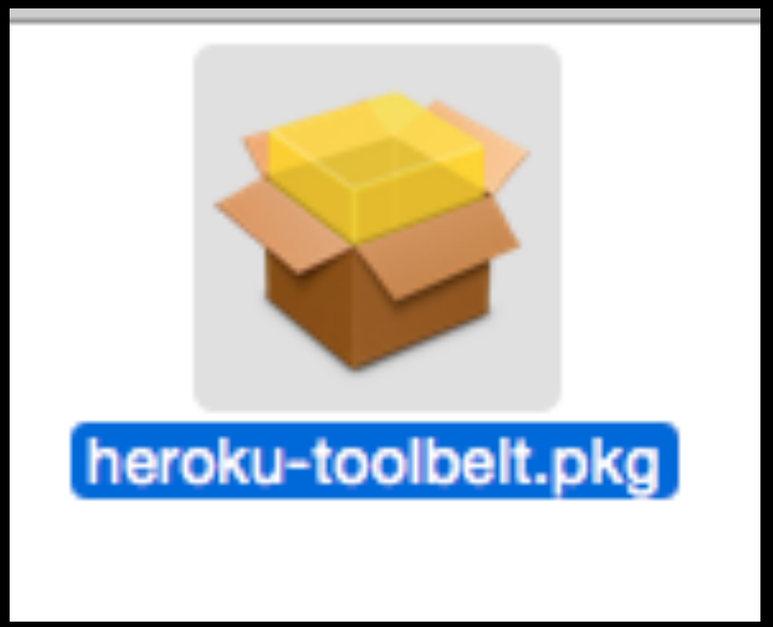
Download Heroku Toolbelt

which include:

- command line interface to communication with Heroku server

- Foreman: run application locally

- Git



heroku

Ruby - Sinatra - Deploy

Git

A version control system
developed by Linus Torvalds
(the guy who developed Linux)

Git: a slang in Scotland.
is another form of “get”, means offspring, illegitimate offspring, illegitimate child, silly, stupid, childish

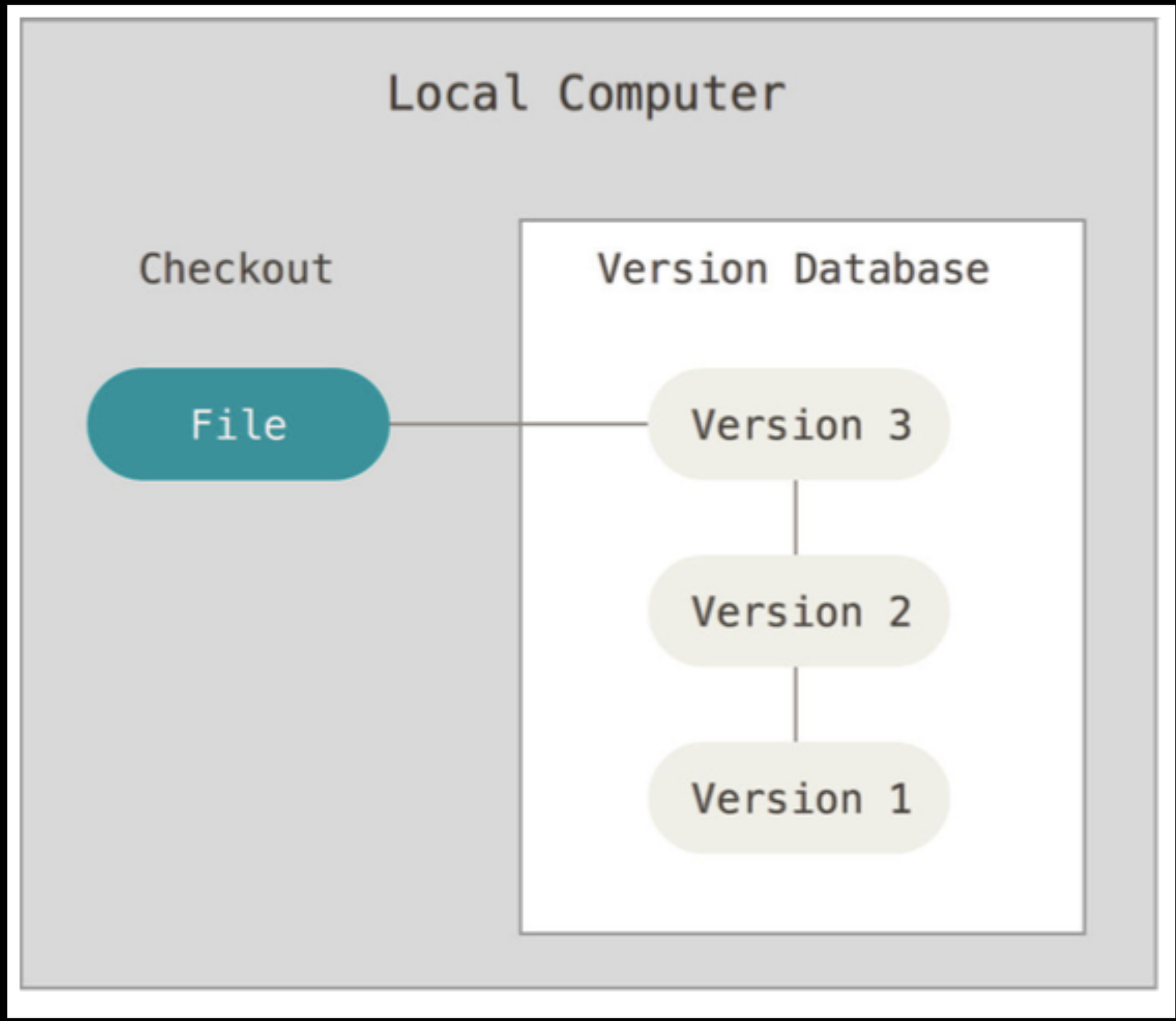
Ruby - Sinatra - Deploy

Version Control System (VCS) allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

Using a VCS also generally means that if you screw things up or lose files, you can easily recover.

Ruby - Sinatra - Deploy

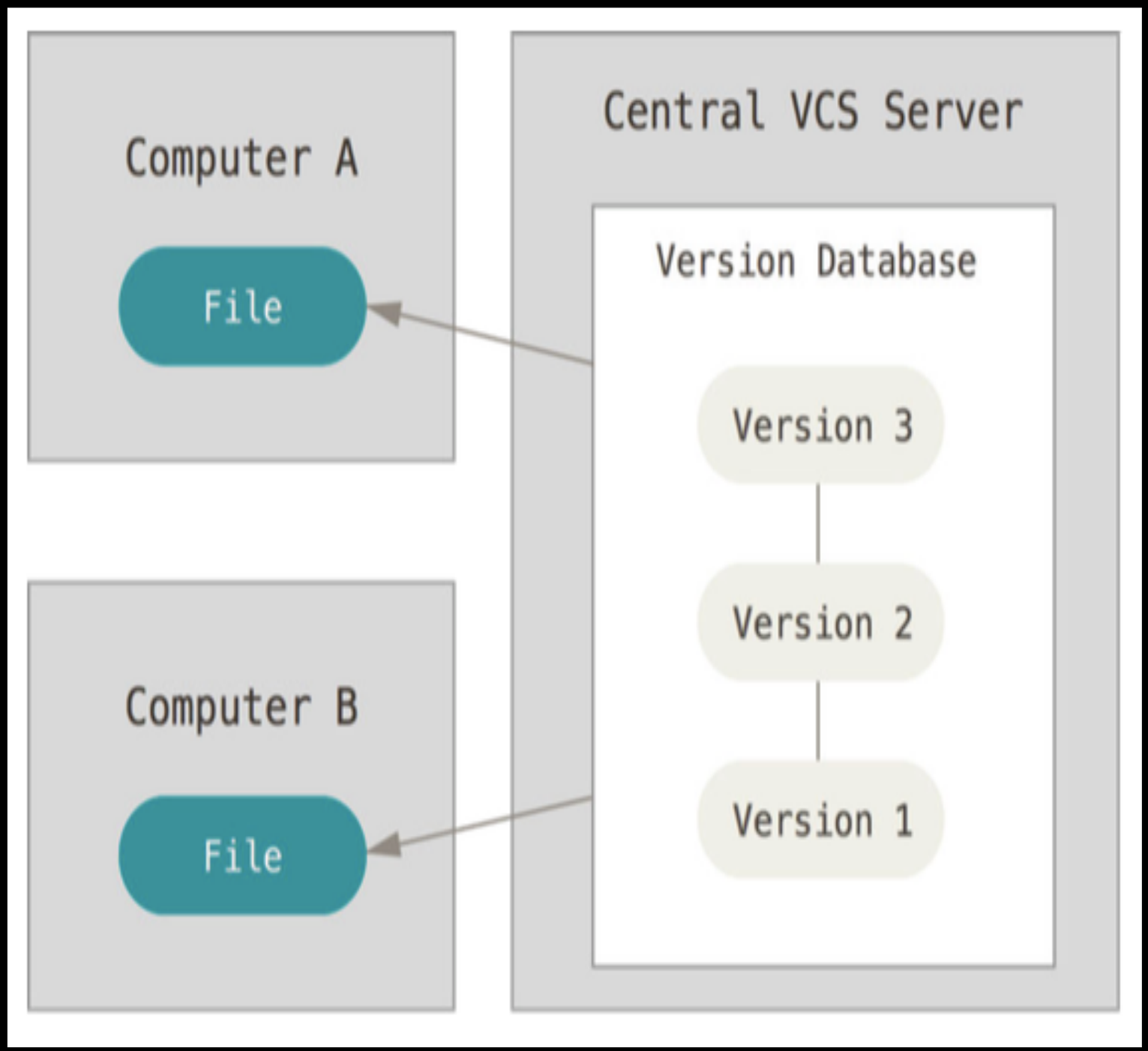
Local VCS



Ruby - Sinatra - Deploy

Centralized VCS

if server goes down,
nobody can collaborate and save changes of the project

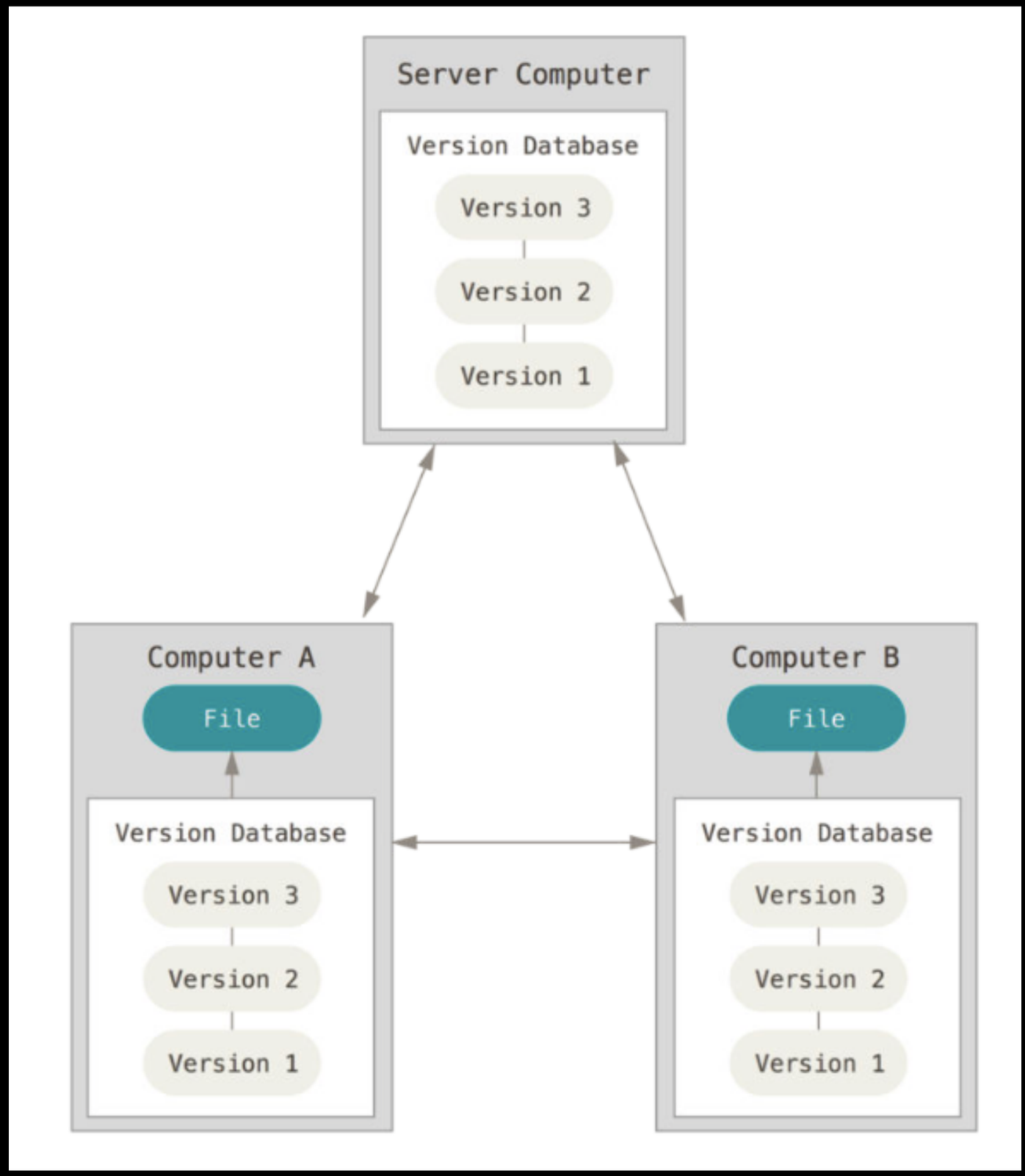


Ruby - Sinatra - Deploy

Distributed VCS - Git

Client keep full mirror of repository (not just the latest)

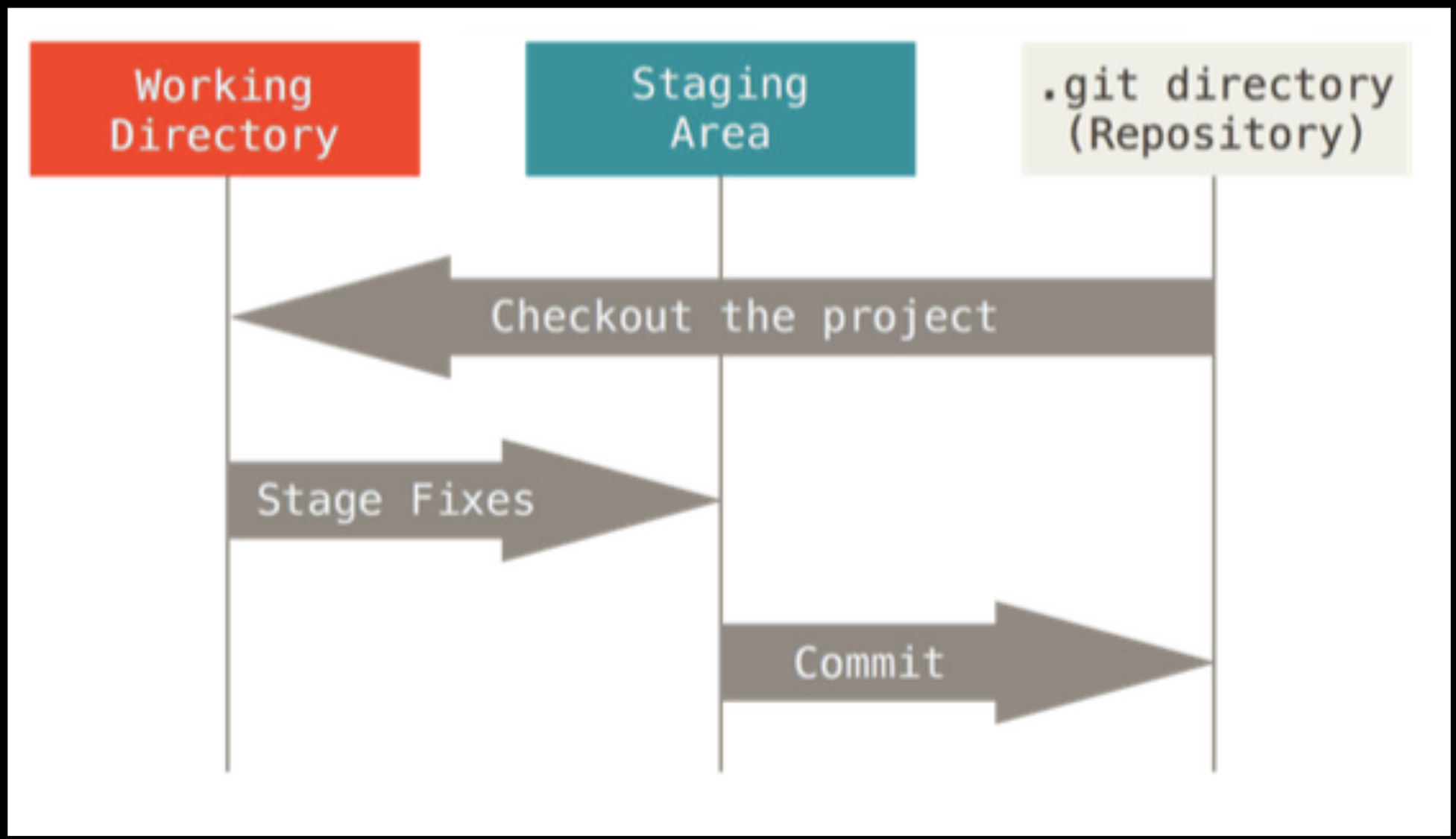
if server dies, any of client repository can be copied back up to the server to restore it



Ruby - Sinatra - Deploy

Git

- Nearly all operation is local.
- 3 Stages:



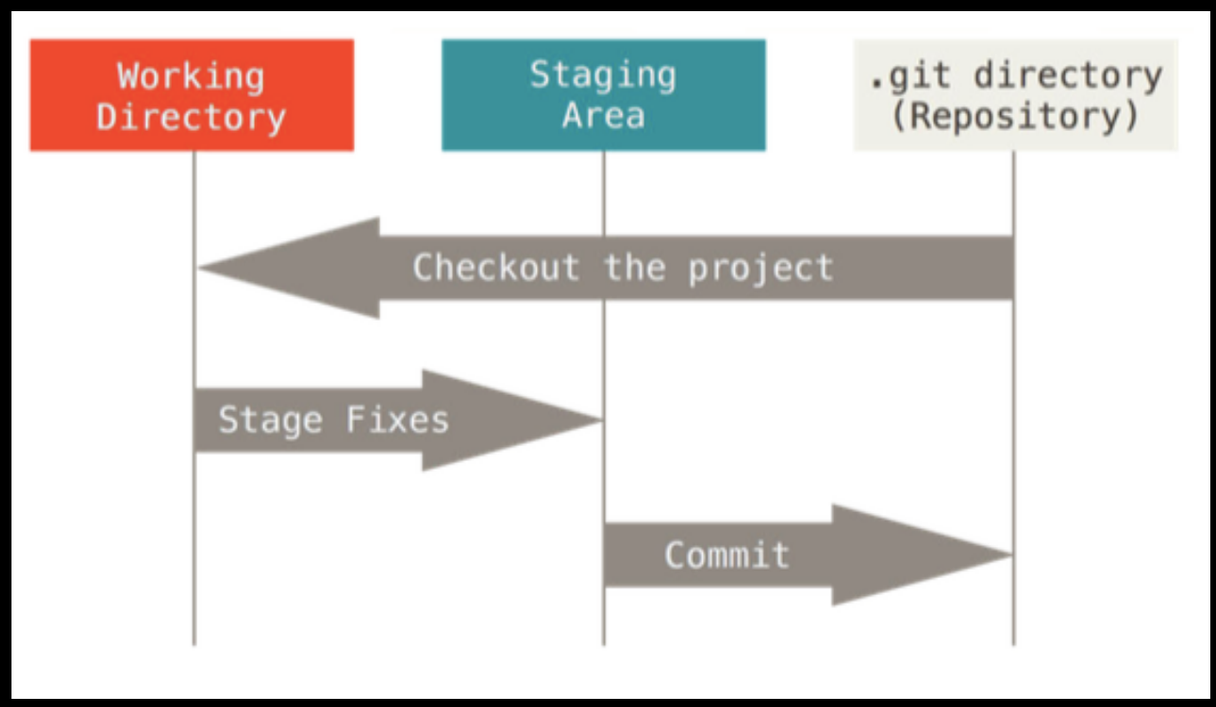
Ruby - Sinatra - Deploy

Git

You make changes to your files in “working directory”

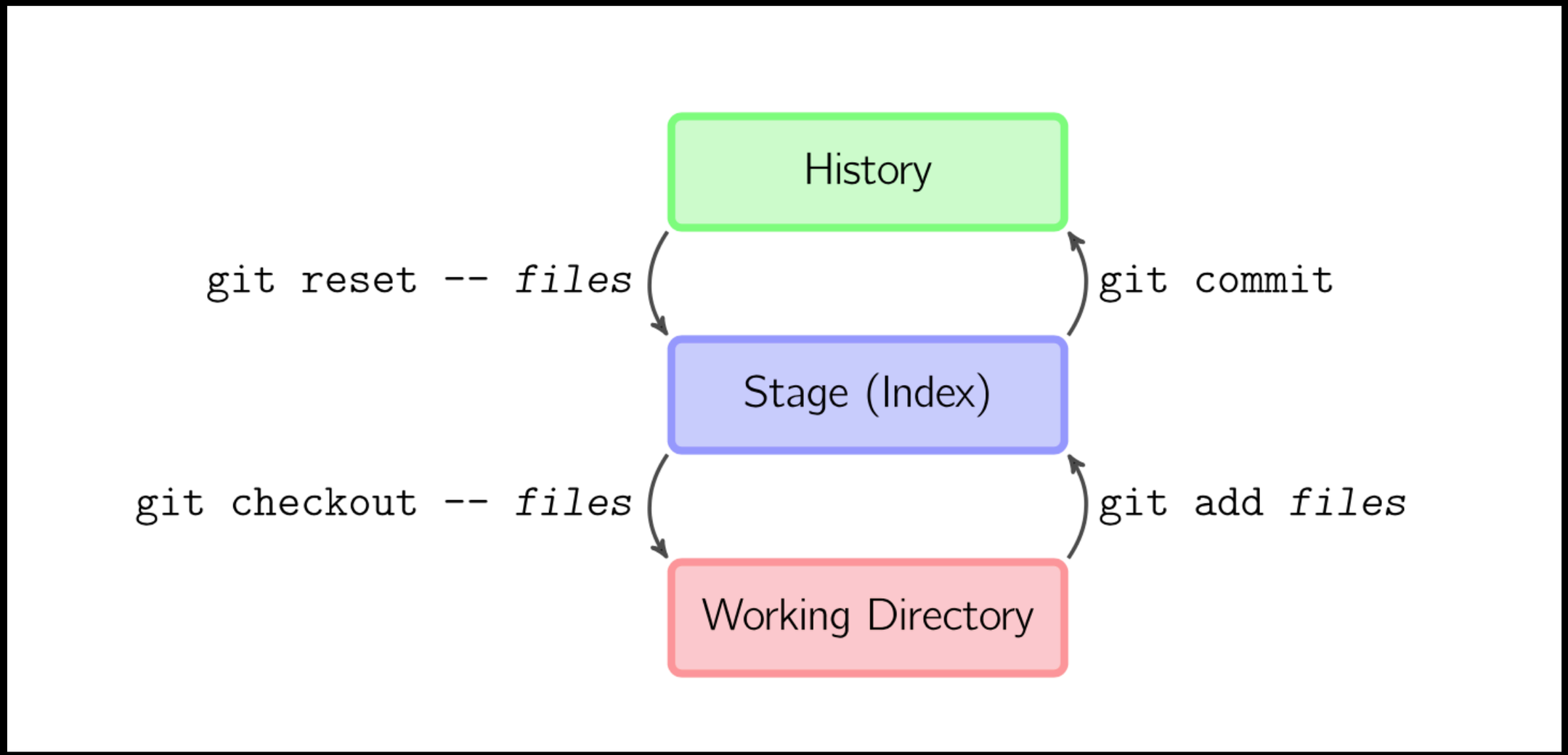
add them to staging area (snapshot of changes in a file in your git directory)

then commit
(store permanently in database)



Ruby - Sinatra - Deploy

Git



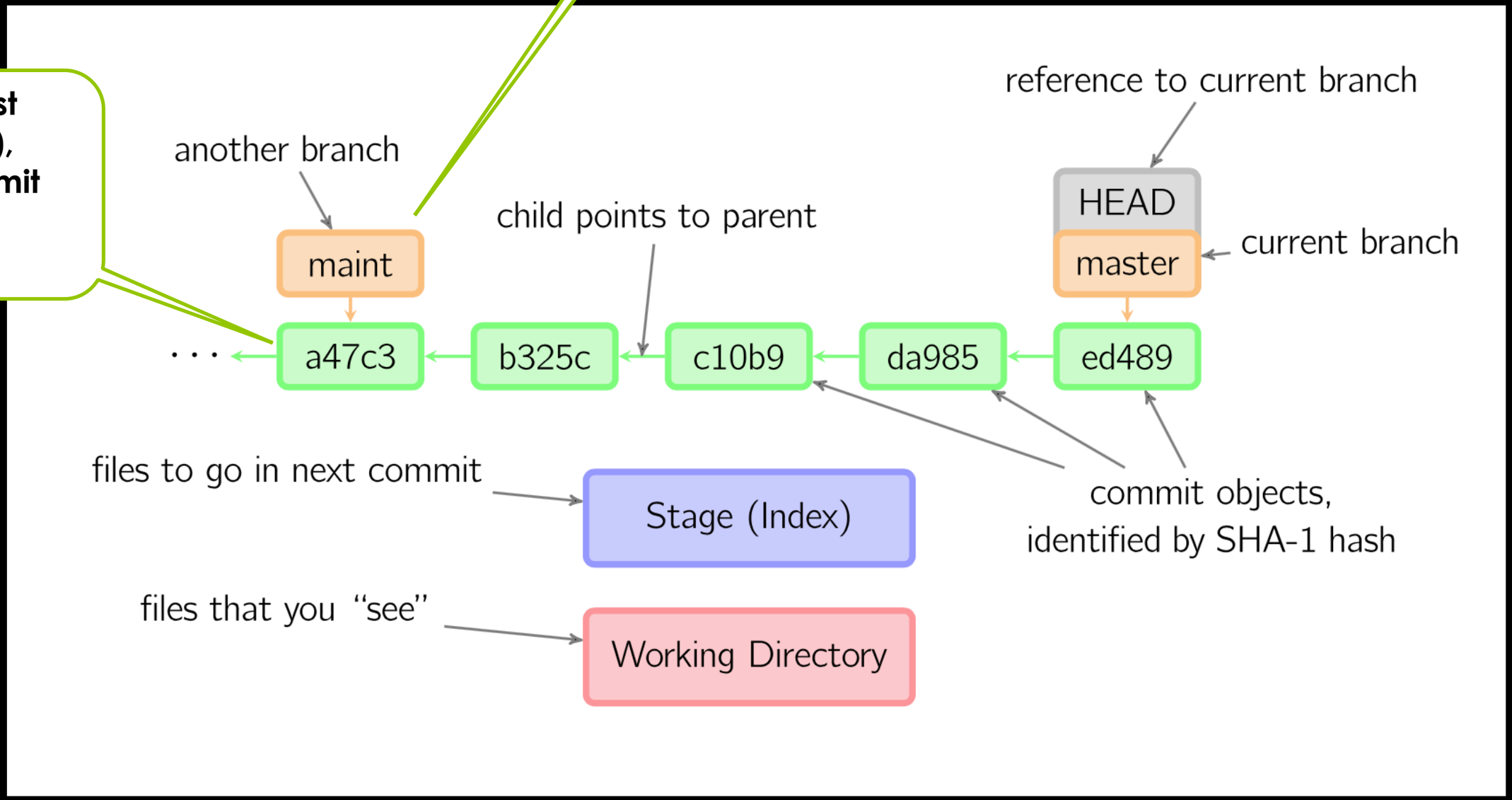
- **`git add` *`files`* copies *files* (at their current state) to the stage.**
- **`git commit` saves a snapshot of the stage as a commit.**
- **`git reset --` *`files`* unstages files; that is, it copies *files* from the latest commit to the stage. Use this command to "undo" a `git add` *`files`*. You can also `git reset` to unstage everything.**
- **`git checkout --` *`files`* copies *files* from the stage to the working directory. Use this to throw away local changes.**

Ruby - Sinatra - Deploy

Git

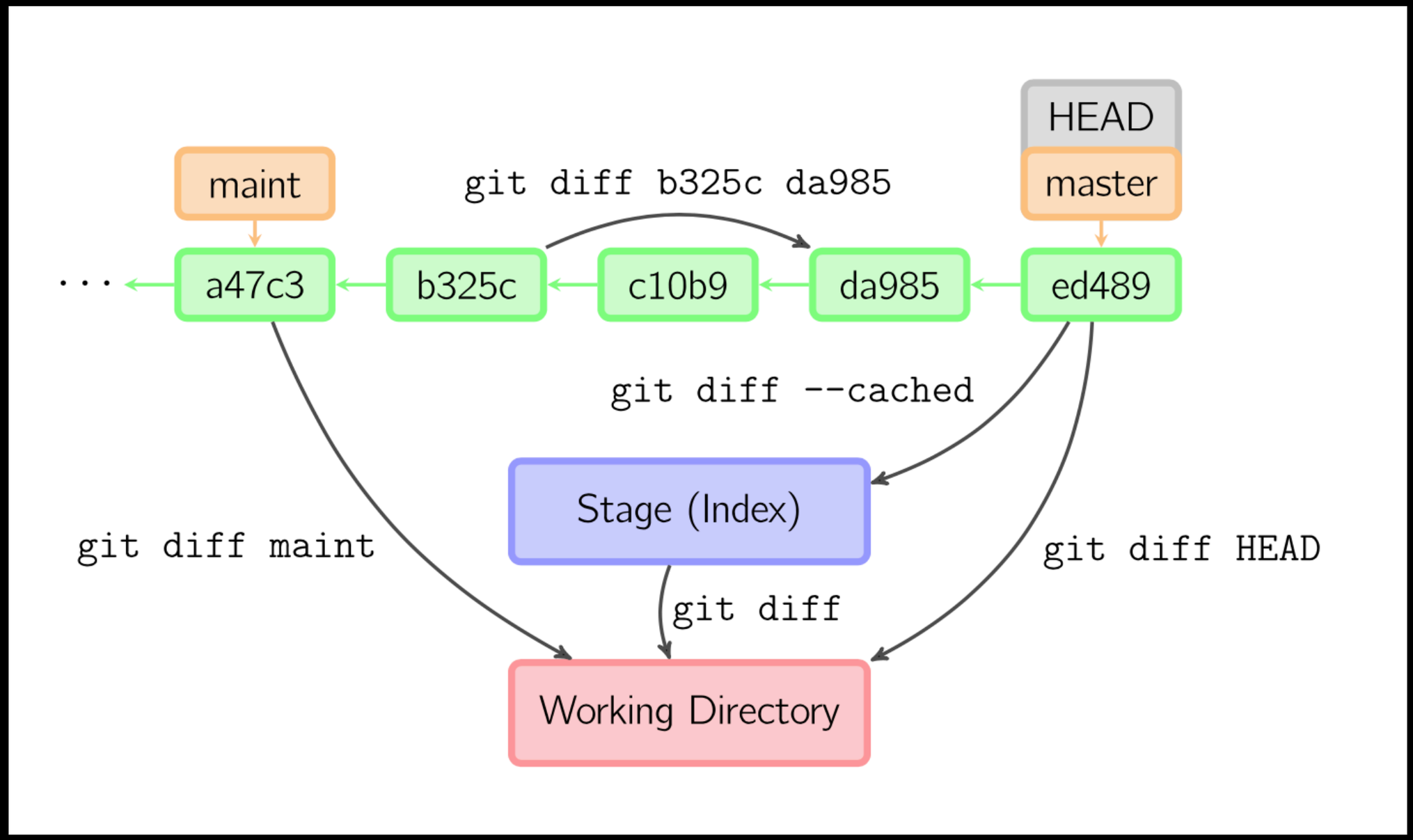
a branch is name of the END of a commit list, you can commit to any branch, the child will grow from there

commit list (5-char id), each commit point to its parent



Ruby - Sinatra - Deploy

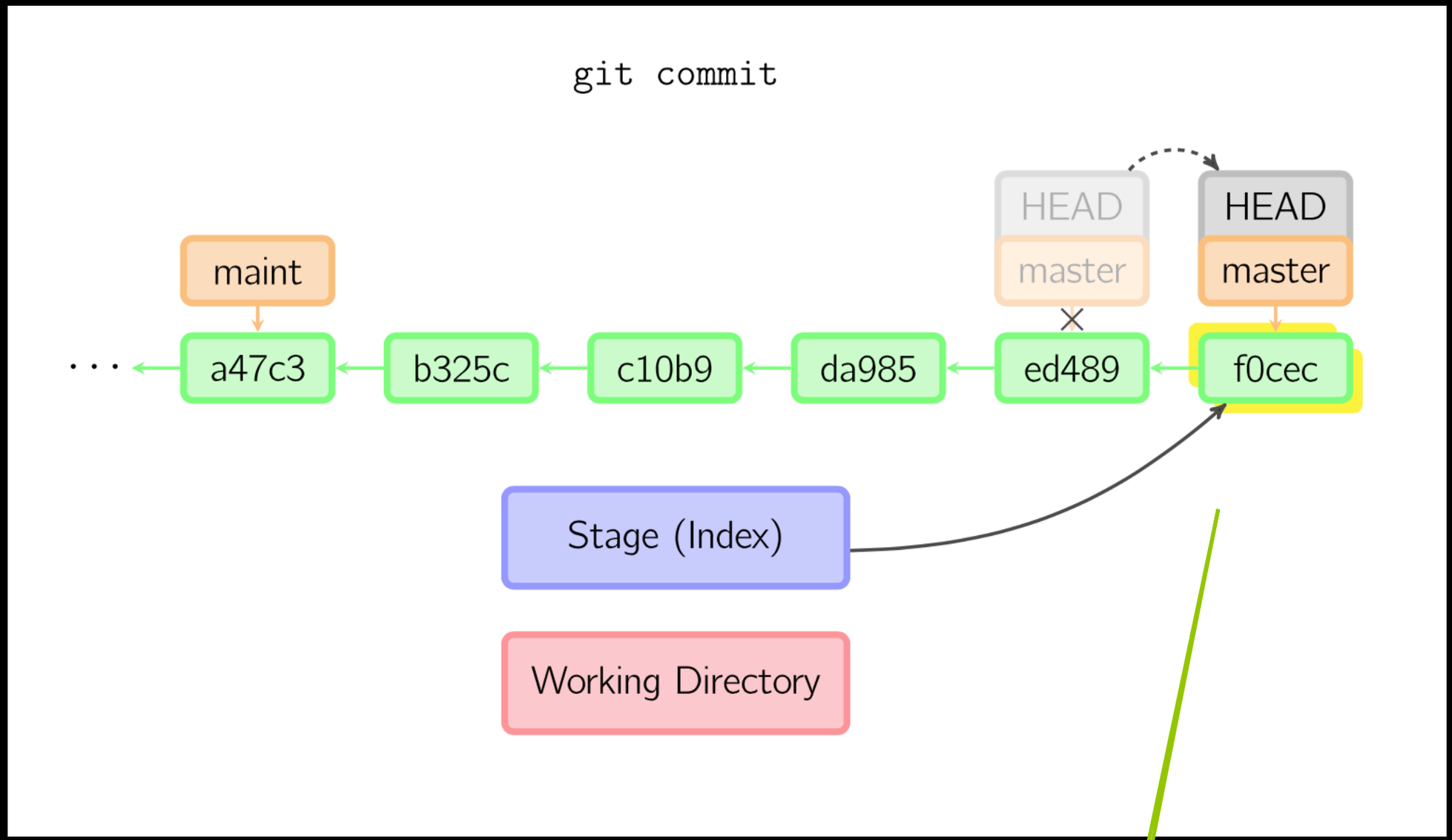
Git



compare difference between working directory and branch, or between stage and working directory, or between different commit

Ruby - Sinatra - Deploy

Git

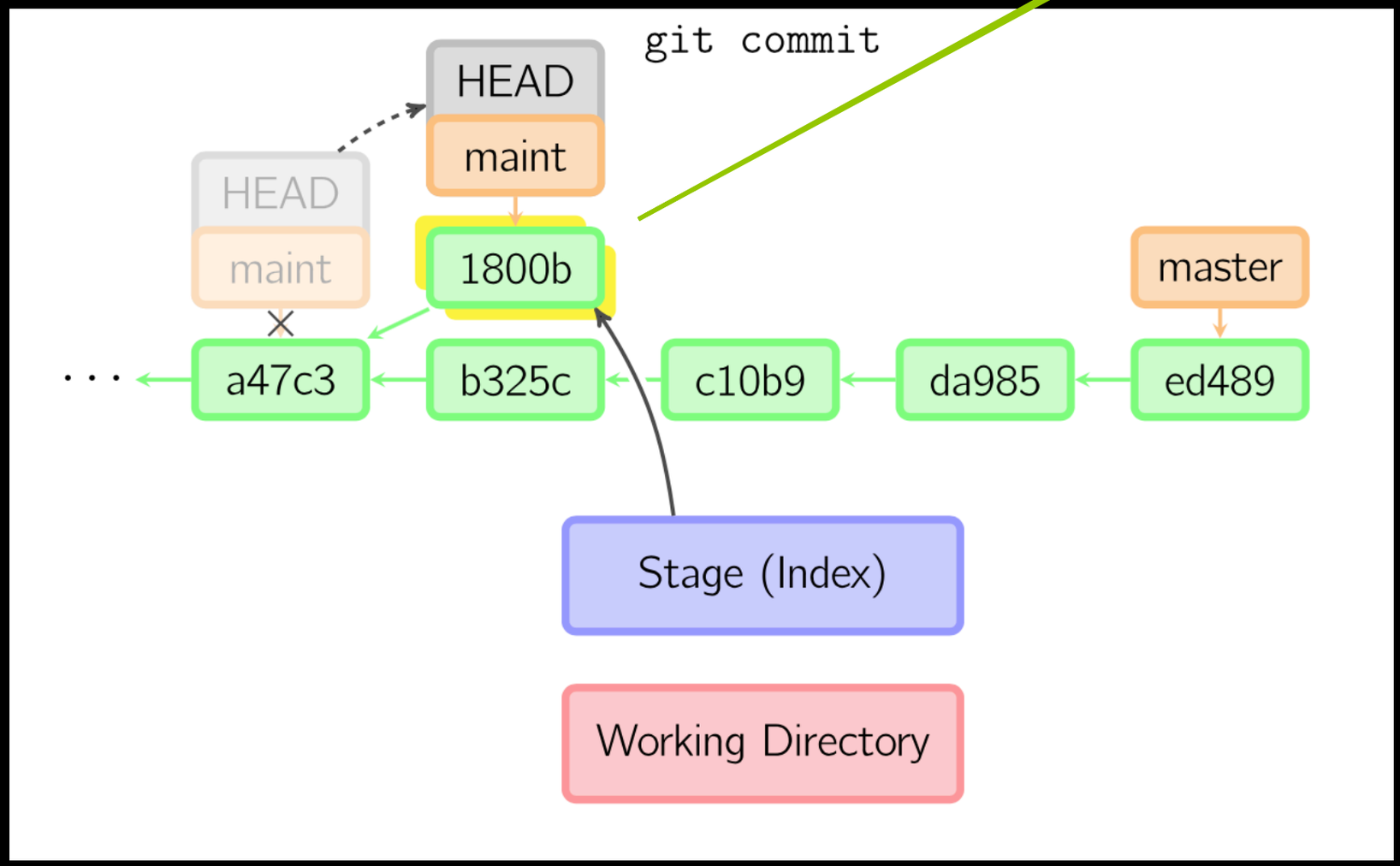


commit, will create a new commit object, current branch will point to the new commit

Ruby - Sinatra - Deploy

Git

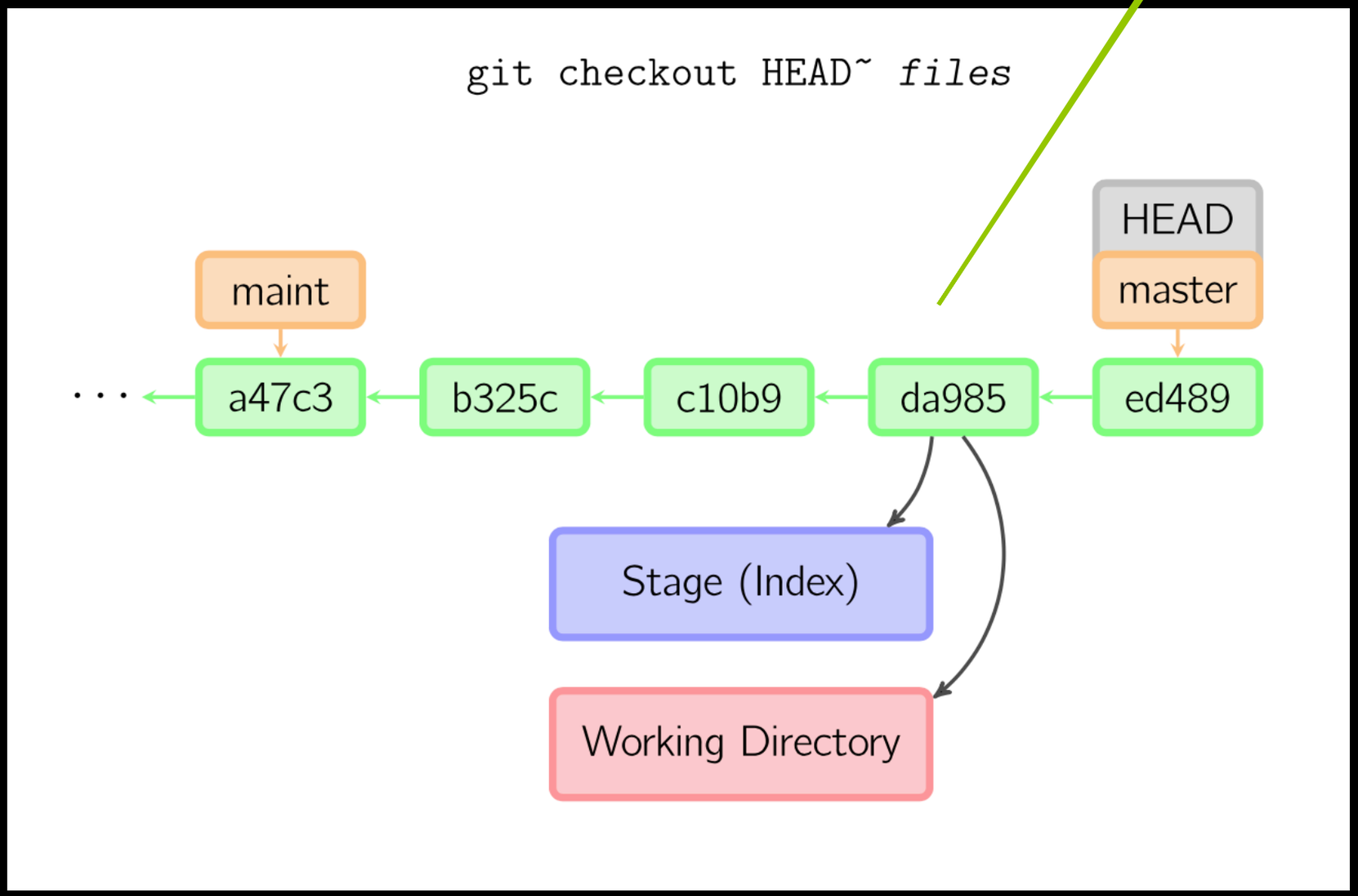
commit to another branch (maint), then new object will be created and become the end of that branch, branch name will point to the new end



Ruby - Sinatra - Deploy

Git

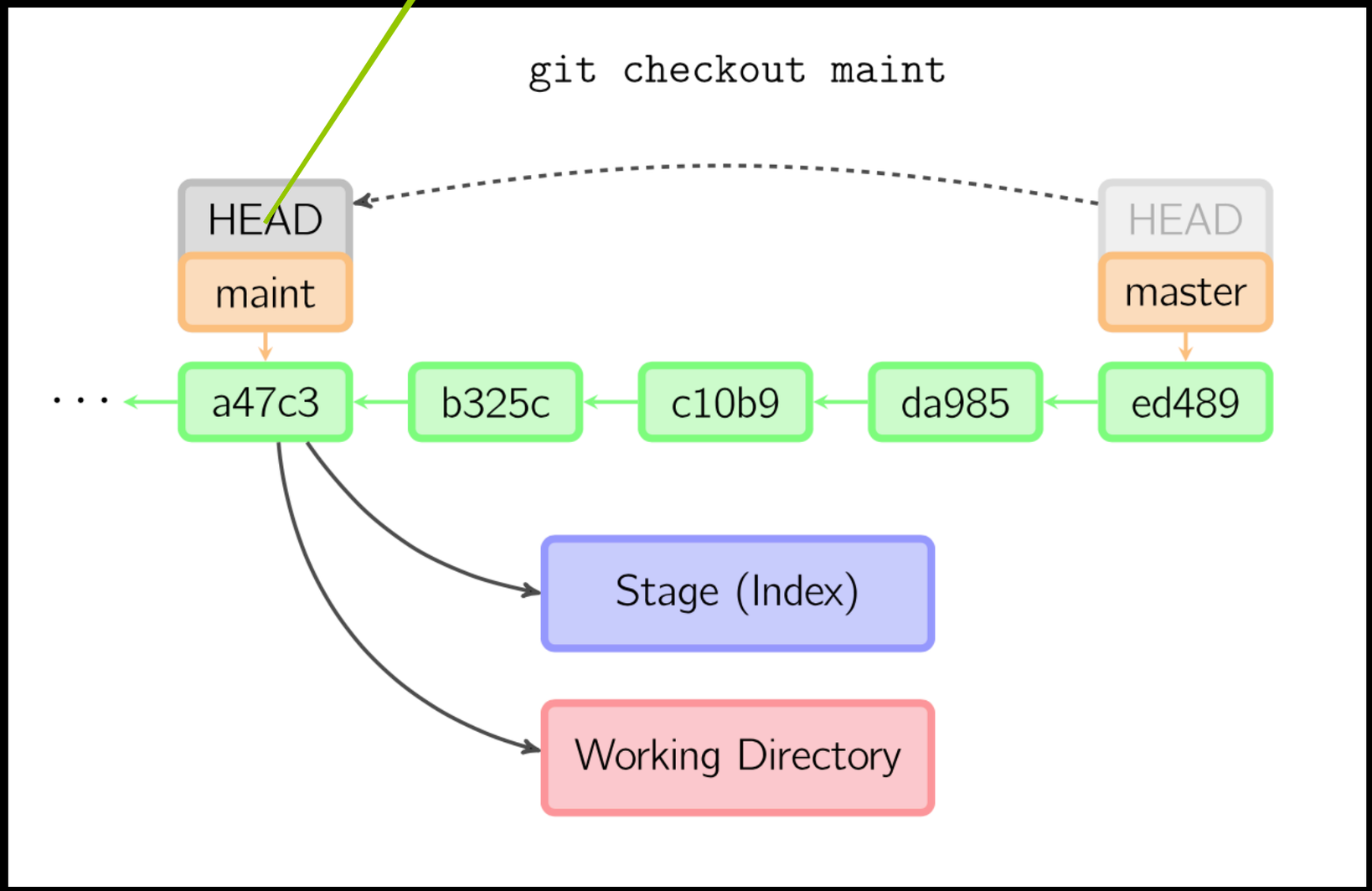
check out file from the parent of current commit (HEAD~)



Ruby - Sinatra - Deploy

Git

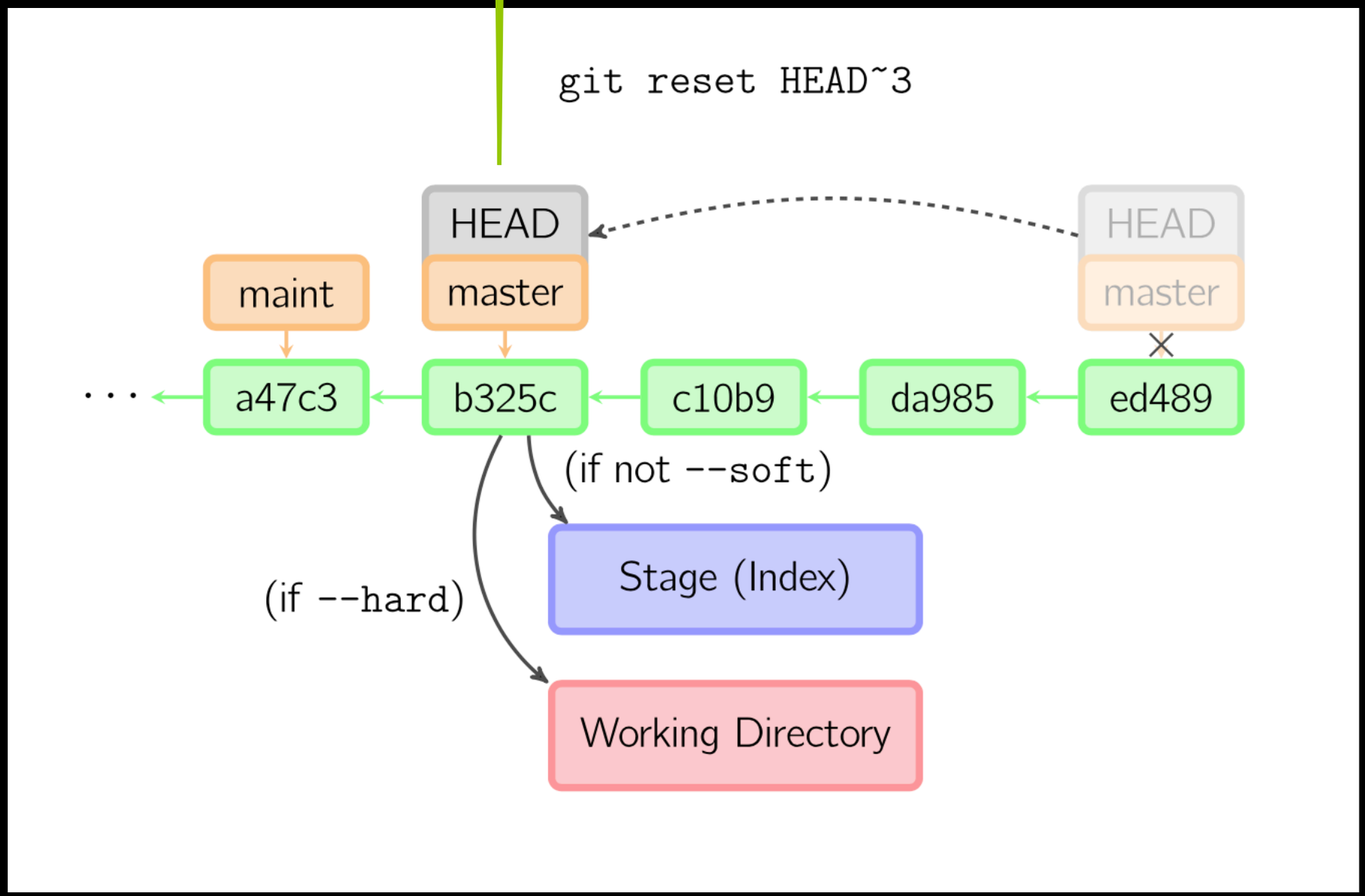
check out file from another commit



Ruby - Sinatra - Deploy

Git

reset will move the end of the current branch to another position, like this case, it move to 3 positions before HEAD



Ruby - Sinatra - Deploy

Git

After install Git, you need to customize your Git environment. you only need to do this once. on any given computer.

They'll stick around between upgrades.

you can change them at any time by running through the commands again:

> git config

Ruby - Sinatra - Deploy

Git

The first thing you should do when you install Git is to set your user name and e-mail address. This is important because every Git commit uses this information:

```
> git config  --global user.name “yuan”
> git config  --global user.email “yuan@email.com”
```

-Git will always use that information for anything you do on that system.

if you want to use different identity for specific projects run that with different name or email without - - global

Ruby - Sinatra - Deploy

Git

> git config - - list

to check config

the config file could be in:

/etc/gitconfig	(- -system option)
~/.gitconfig	(- - global option)
.git/config	(local option)

> git help

Ruby - Sinatra - Deploy

Git

to initialize .git repository:

>git init

to place code in repository:

> git add filename

to check file status:

> git status

to check what has changed:

> git diff

to remove file:

> git rm filename

to move file:

> git mv file1 file2

to check commit history:

> git log

the repository will be created in .git folder

Ruby - Sinatra - Deploy

Now we are ready to deploy…..

Ruby - Sinatra - Deploy

Create our application:

- use Bundler to prepare all gems (http://bundler.io)

Bundler is a program to install all gems required by your application.

It will use a file called "Gemfile" (no extension) to track all gems used by the application

Ruby - Sinatra - Deploy

Bundler itself is a gem.

to check if you have it:

$ gem list bundler

if you don’t, then install it as installing any gem

$ gem install bundler

Ruby - Sinatra - Deploy

create “Gemfile” for bundler to use with following content:

```
source :rubygems
gem "sinatra"
gem "slim"
gem "sass"
gem "dm-core"
gem "dm-migrations"
gem "thin"
gem "pg", :group => :production
gem "dm-postgres-adapter", :group => :production
gem "dm-sqlite-adapter", :group => :development
```

you can group different gem together so that you can specify different bunch of gems for different type of environment, like for example production deployment will use a different set of gems

use sqlite locally for development, use postgres for production on server (heroku)

Ruby - Sinatra - Deploy

run **bundle** to create a file called “Gemfile.lock” containing all the gems we are using and their dependencies.

>bundle install --without production

this means, create all the gems and their dependencies, and gems in production group will not be installed locally

Ruby - Sinatra - Deploy

- Heroku uses ‘rackup’ program to run application. It need a config file called: config.ru.

- Create config.ru:

```
require './main'
run Sinatra::Application
```

other configuration and settings can be put here too

Ruby - Sinatra - Deploy

Place all our code in GIT repository
(on our local machine)

Create .git repository folder (empty)

```
> git init
```

set ‘git’ identity

```
> git config  user.name “yuan”
> git config  user.email “yuan@email.com”
```

Ruby - Sinatra - Deploy

Add our application files into Git repository,
in our application folder:

> git add .

add all the files in current folder to the staging area

> git commit -m ‘initial commit’

commit the changes to repository, attach some message to describe the changes.

all codes are under version control now

Ruby - Sinatra - Deploy

- Deploy file from Git repository to Heroku server

create app on Heroku

```
> heroku create myfirstapp
```

push git repository onto Heroku server

```
> git push heroku master
```

now our application is live on Internet

Ruby - Sinatra - Deploy

Check application

> heroku open

This will open the browser and point
to the URL on Heroku server automatically

Ruby - Sinatra - Deploy

One more thing to do for database to work

if you type "/songs" url, you will find that there is a Server Error, this is because database table is not created yet.

To create database table, we need to run

DataMapper.auto_migrate!

on server

To do this, run Heroku console: it will start IRB on remote server:

```
> heroku run console
irb> require './main'
irb> DataMapper.auto_migrate!
```

Ruby - Sinatra - Deploy

- Everything should be ok now.

Ruby - Sinatra

End