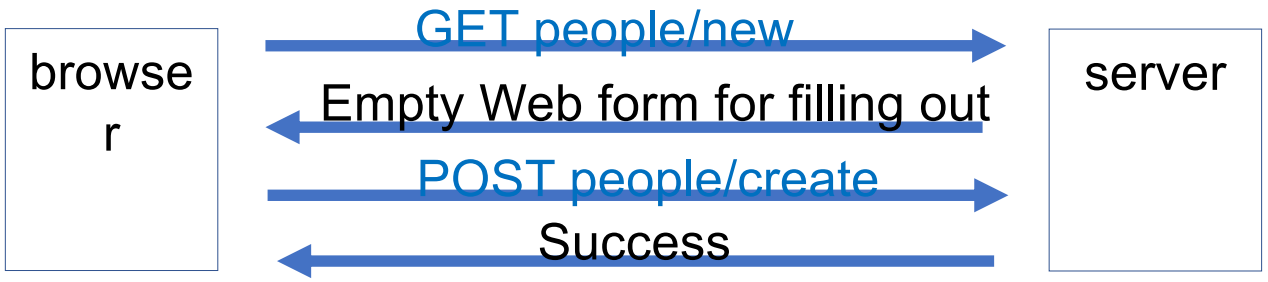
Software Engineering
ECE444

More Advanced Rails

Render v. Redirect

Recall: creating new person (updating person similar):



But: default view may not be user-friendly.

Alternative: render

- render :action_name (when from same controller)
- render "classes/show" (when from different controller)
- render "apps/different_app/app/views/products/show" (when from different app on same server)

Access to cntr. instance vars

Alternative: redirect: ask browser to send new request

- redirect_to people_path
- redirect_to "https://cnn.com"

No access to controller instance vars

Flash

- flash: a hash used to pass message to next action.
- Useful for displaying a message on next page

```
flash[:notice] = "#{@person.fname} #{@person.lname}
                  successfully created"

flash[:warning] = "Error: …<description>…"

flash[:any_symbol] "…<any text>… "
```

and in view:

```
<% [:notice, :error].each do |key| %>
   <% if flash[key] %>
      <div class="<%= key %>" id="flash">
          <%= flash[key] %>
       </div>
    <% end %>
<% end %>
```

Flash hash similar to more general session hash.
Session hash persists across browser requests.
Flash hash is erased after next request.

Active Record validations

- Automatic checking of data before storing in DB
- Advantages:
  - ensure only valid data stored in DB
  - DB agnostic
  - cannot be bypassed
  - cross-cutting DRY functions automatically triggered on
    - create
    - save
    - update
- Alternatives:
  - DB stored procedures
    - DB dependent
  - Client side
    - helpful, but unreliable: can be bypassed
  - Controller
    - not DRY: difficult to maintain, tend to be unwieldly.

Validations specified in model

E.g.,

See RoR Validation Guide

```
class Person < ApplicationRecord
    validates :fname, :lname, :email, presence: true
    validates :email, format: {
       with /^[\w+\-.]+@[a-z\d\-.]+\.[a-z]+$/I,
       message: "malformed email address"
    validates :passwd, length: {in: 6..20}
    validates :age, numericality: true
    validates :age, noericality: {only_integer: true}
    validates :age, greater_than: 12
    validates :email, uniqueness: true
    validates :email, confirmation: true
    validates :terms_of_service, acceptance: true
```

Custom validators & Explicit checks

- in model:

```
validates_with AddressCheck # custom validator
```

- then:

```
class AddressCheck < ActiveModel::Validator
   def validate( p )
      .
      .
      .
   end
end
```

- Explicit check:

```
@person.valid?    # returns T/F
```

Validation errors

When validation fails:

- `save`/`create`/`update` returns false
- `save!`/`create!`/`update!` results in exception
- errors are recorded in Errors object, returned by `errors` method:
  - `@Person.errors` # for all errors
  - `@Person.errors[:email]` # for errors related to email
  - `@Person.errors.full_message` # array of all error msgs

More general: Active Record callbacks

- ActiveRecord callbacks
  - of which validations are just a special case
  - allow you to "intercept" a model object at various points in its lifecycle; e.g.,
    - `before_validation`
      `before_validation_on_create`
      `before_validation_on_update`
    - `after_validation`
      `after_validation_on_create`
      `after_validation_on_update`
    - `before_save`
      `before_save_on_create`
      `before_save_on_update`
    - `after_save`
      `after_save_on_create`
      `after_save_on_update`

Callback example

```
class Journal < ApplicationRecord
    before_save :capitalize_title

    def capitalize_title
        self.title = self.title.split(/\s+/).
                        map(&:downcase).
                        map(&:capitalize).
                        join(' ')
        end
    end
```

Filters

- Callbacks analogous to validations but for controllers
- write filter once, but apply to all actions:
  - before action is called
  - after action is called
  - or around
- Used, e.g.,
  - to check certain conditions are true
  - to implement authentication
  - for logging
  - response compression
  - response customization

Filter example

```
class PersonController < ApplicationController
   before_action  :authenticate_user

   def authenticate_user
      if session[:userid]
          @current_user = User.find session[:userid]
          return true
      else
          flash[:error] = "You must be logged in"
          redirect_to login_page
      end
   end
end
```

- Can override:

```
skip_before_action :authenticate_user, only: [:signup]
```

Filter placement ; filter order

- Filters get inherited
  ➔ put filters in

    ```
    ApplicationController < ActionController
    ```

    so that it applies to all actions in all controllers

- You can define multiple filters
  - they are run in the order they are declared in.

User Authentication

- We need to be able to:
  - signup new user (and get passwd)
  - login
  - restrict access for most pages to authenticated user
  - logout / timeout
- Here: full (but simplistic) implementation to learn how it works
- In practice: use gem 'Devise' that does everything

User Authentication II

- Cardinal rule: never store plaintext passwords!!!
  Instead: store hash of password

one-way encryption

E.g.,
```
require 'digest/sha'
encrypted_passwd = Digest::SHA1.hexdigest(passwd)
```

- Not very safe: want to use salt

pseudo-random value

```
salt= Digest::SHA1.hexdigest("#{email},#{time.now}")
encrypted_passwd = Digest::SHA1.hexdigest(
                                    "#{salt}#{passwd}")
```

- Alternatively: use gem:
  1. In `app/gemfile` add:
     ```
     gem 'bcrypt-ruby', :require => 'bcrypt'
     ```
  2. ```
     salt = Bcrypt::Engine.generate_salt
     encryped_passwd = Bcrypt::Engine.hash_secret(passwd, salt)
     ```
     ```

User Authentication III

- Need to store encrypted_passwd and salt in DB Migration

```
class AddAuthInfo < ActiveRecord::Migration
   def change
      change_table :users do |t|
         t.string :email   # if not already there
         t.string :encrypted_passwd
         t.string :salt
      end
   end
end

>rake db:migrate
```

Note:

- User often needs to input a lot of info about herself.
- Generally not a good idea to ask for all info at one.
- Instead: start just with auth info

User Authentication IV

New view:

```
<% @page_title = "Signup" %>
<dev class = "signup_form">
<h1>Sign Up</h1>

  // (A) error message from previous attempts go here

<%= form_with model @user, local: true,
                url: users_path do |f| %>
   <p>Email:</br> <%= f.text_field :email %> </p>
   <p>Passwd:</br><%= f.password_field :passwd %></p>
   <p>Passwd confirmation </br>
      <%= f.password_field :passwd_confirmation%></p>
   <%= f.submit :signup %>
<% end %>
```

User Authentication V

In model:

```
class User < ApplciationRecord
  attr_accessor :passwd  # adds element not in DB
  #some validations
  validates :email :presence => true
                   :uniqueness => true
                   :format => EMAIL_REGEX
  validates :passwd :confirmation => true
                    # autocreates hidden passwd_conf
  validates_length_of :passwd, :in => 6..20

  before_save :encrypt_passwd
  after_save :clear_passwd

  def encrypt_passwd
     self.salt = Bcrypt . . .
     self.encrypted_passwd = Bcrypt . . .
  end
  def clear_passwd; self.passwd=nil; end
```

B

authentication goes here

User Authentication VI

Controller:

```
class UserController
  def new
    @user = User.new
  end
  def create
    @user = User.new( params.require(:user).
      permit( :email, :passwd, :passwd_confirmation) )
    if @user.save
        flash[:notice] = "Signup successful"
        redirect_to . . .
    else
        flash[:alert] = "Problem"
        render new
    end
  end
end
```

User Authentication VII

Add error message to new view at 


```
<% if @user.errors.any? %>
  <ul class="SignupErrors">
  <%= for message_error in @user.errors.full_messages %>
     <li> <%= message_error %> </li>

  <% end %>
  </ul>
 <% end %>
```

User Authentication VIII

Login Authentication (assume login form: :email :login_passwd)

- in User Model at B

```
def match_passwd( login_passwd = " " )
  encrypted_passwd ==
       Bcrypt::Engine.hash_secret( login_passwd, salt )
end

def self.authenticate( email="", login_passwd="" )
  if EMAIL_REGEX.match( email )
      user=User.find_by_email( email )
  end
  if user && user.match_passwd( login_passwd )
      return user
  else
      return false
  end
 end
```

User Authentication IX: Sessions I

```
class SessionController < ApplicationController
  def  login    # renders login form
  end
  def login_attempt
     auth_user = User.authenticate( params[:email],
                                    params[:login_passwd] )
     if auth_user
        session[:userid] = auth_user.id
        flash[:notice] = "Welcome back #{auth_user.email}"
        redirect_to( :action => "home" )
     else
        flash[:notice] = "Try again"
        render "login"
     end
  end
end
```

User Authentication X: Sessions II

- need to check session on every action requiring authentication
  ➔ use `before_filter`
- to have it apply to everything: add filter to superclass of all controllers: `ApplicationController`

```
def authenticate_user
   if session[:user_id]
       @current_user = User.find session[:user_id]
       return true
   else
       redirect_to( :controller => "session",
                    :action -> "login" )
       return false
   end
 end
```

User Authentication XI

But don't do this yourself!

Instead: user `devise` gem, which does it all for you

User Authentication: 3rd Party Auth

Part 1

1. "Login with Twitter"

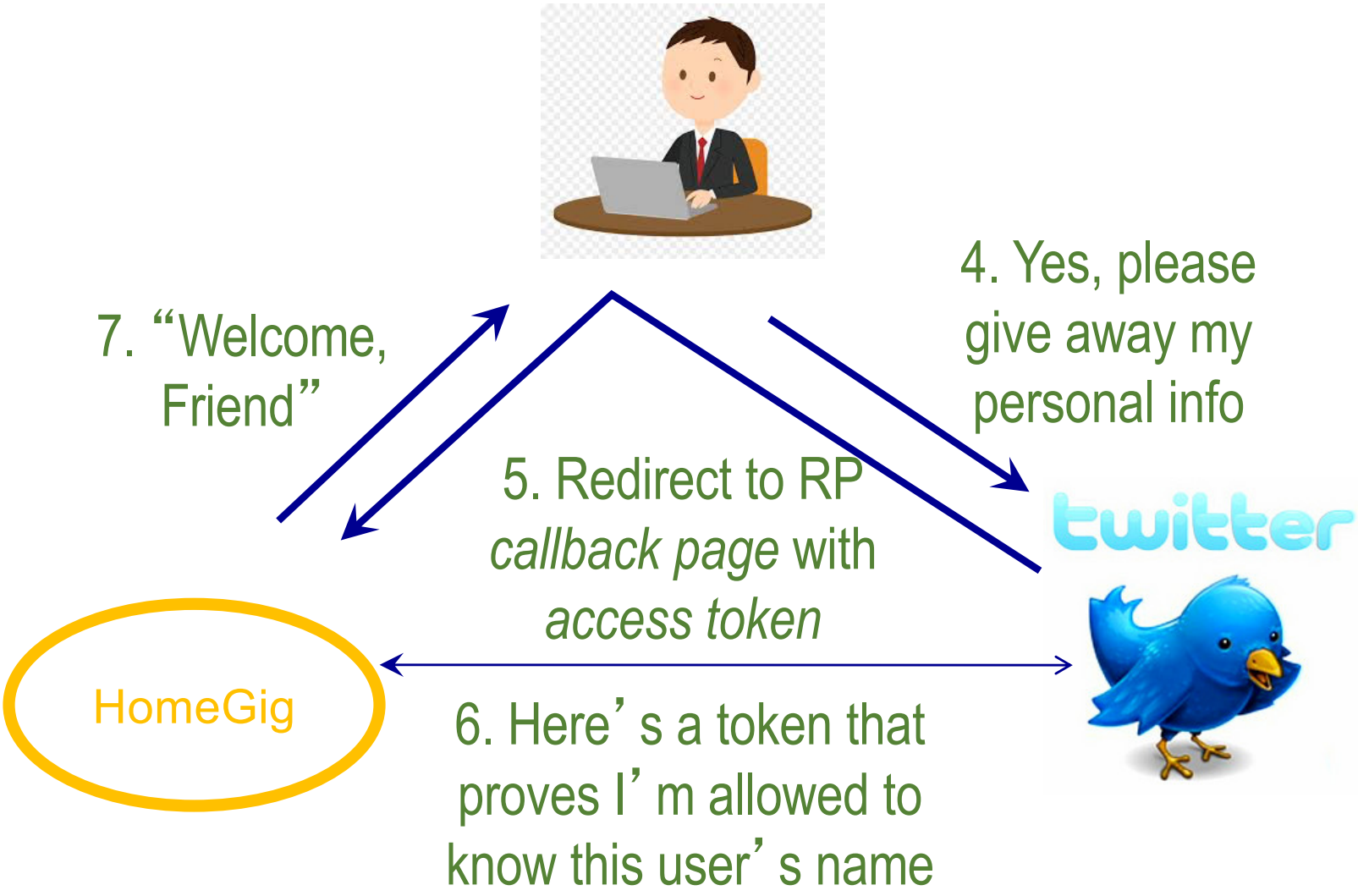
2. Redirect to Twitter login page

3. "OK to authorize this app?"

HomeGig

User Authentication: 3rd Party Auth II

Part 2



User Authentication 3rd Part Auth III

- OmniAuth gems do all this for you
  - E.g., OmniAuth_twitter

- Note: for Twitter authentication:
  - you need to register with twitter and get "dev" account
  - they will give you an API_KEY & API_SECRET

Associations

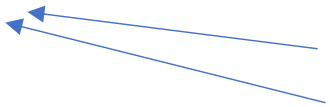
- a logical relationship between two types of entities

User:

id			

Review:

id	user_id		



```
class User < ApplicationRecord
end
class Review < ApplicationRecord
end

@review = Review.create(published_at: Time.now, user_id: @user.id, …)

@reviews = Review.where(user_id: @user.id)
@reviews.each do |review|
  review.destroy
end
@user.destroy
```

Rails Associations establish such relationships and greatly simplify coding so you don't have to write code like this…

Associations streamline things

```
class User < ApplicationRecord
  has_many :reviews, dependent: :destroy_all
end

class Review < ApplicationRecord
  belongs_to :user
end

@review = @user.reviews.create(published_at: Time.now, …)

@user.destroy

@user.reviews    # collection of all reviews by @pers
@review.user     # person who wrote review
```

Other dependency actions:
:nullify – sets foreign keys to null
:restrict_with_error
:destroy – same as destroy_all, but individually with appropriate callbacks

Types of associations: one-to-one

- one-to-one relationships
  - has_one
  - belongs_to
  - has_one :through

```
class CreateUser < ActiveRecord::Migration
  def change
    create_table :user do |t|
      t.string :fname
      . . .
      t.timestamps
    end

    create_table :profile do |t|
      t.belongs_to :user
      . . .
      t.timestamps
    end
  end
end
```

```
class User < ApplicationRecord
   has_one :profile
end
```

```
class Profile < ApplicationRecord
   belongs_to :user
end
```

User:

id	email	fname	lname	…

Profile:

id	prefs	user_id	…

Types of associations: one-to-one II

- or try:

```
> rails g model User . . .
> rails g model Profile prefs:string … user:references
```

  to create models and migrations for you with:
  - `user_id` col. in profiles table
  - `belongs_to :user` in profile model

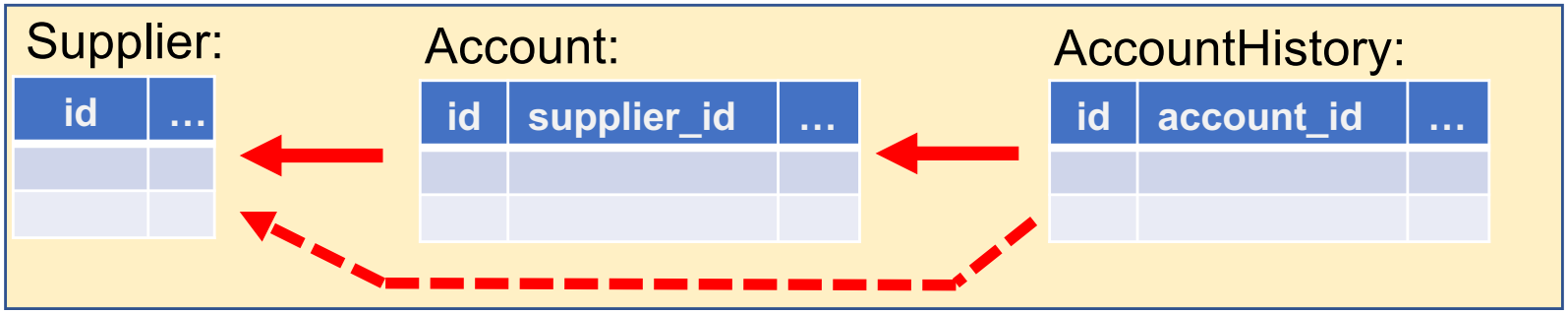
- and add to `models/user.rb`:

```
has_one :profile
```

- Then you can call methods like:
  - `user.profile`
  - `profile.user`
  - `user.build_profile`
  - `user.create_profile`

Types of associations one-to-one :through

- indicates one model has a one-to-relationship as defined through a third model
- E.g.,




```
class Supplier < ApplicationRecord
    has_one :account
    has_one :account_history, through: :account
end

class Account < ApplicationRecord
    belongs_to :supplier
    has_one :account_history
end

class AccountHistory < ApplicationRecord
    belongs_to :account
end
```

Types of associations one-to-many

- As first example . . . (most common type of association)
- or try:

```
> rails g model User . . .
> rails g model Review . . . user:references
```

- and add to `models/user.rb`:

```
    has_many :reviews
```

- Then you can call methods like:
  - `user.reviews`
  - `review.user`
  - `user.reviews << review  # establishes new relation`
  - `user.reviews.build(…)  # without saving`
  - `user.reviews.create(…)  # with saving`

Types of associations many-to-many

- traditional DBs: requires 'join' operations
- here: set up intermediate table:

```
> rails g model User . . .
> rails g model Event . . .
> rails g migration create_events_users user:references
                                        event:references
```

- and add to `models/user.rb`:

```
has_and belongs_to_many :events
```

  and to `models/event.rb`:

```
has_and_belongs_to_many :users
```

- Then you can call methods like:
  - `user.events    event.users`
  - `user.events << event   # establishes new relation`
  - `user.events.destroy(event1)`

```
# only destroys relation, not objects
```

Version Control

Two purposes:

1. retain and provide access to every version of every file ever stored related to a project
2. allow and support teams to collaborate

AKA:

- source control (management)
- revision control system

Have been around for quite some time:

SCCS (1975) → RCS → CVS → . . .
→ Subversion and Mercurial - more modern can do everything
→ Git - invented by Linus Torvalds

Version Control: Fundamental Rules

1. Keep ("check in") absolutely everything in version control, including
   - source code
   - tests
   - scripts
   - documentation
   - makefiles
   - libraries
   - configuration files
2. Check in changes frequently
3. Use meaningful commit messages

Version Control: our modus operandi

You must operate as follows ("the FB way" ☺):

- At Github: maintain a "Central Repo" with one "Master Branch"
- Every group member on their local host maintains a clone
- Changes are only made to local copy
- When done: changes are "push"ed to central Master Branch
- No other branches are allowed
- You must use a Github account based on your UofT email
- You must invite me and the TA's…

Github

→github.com

→ Repositories

→"New" button

→give it a (meaningful) name

→initialize it with README

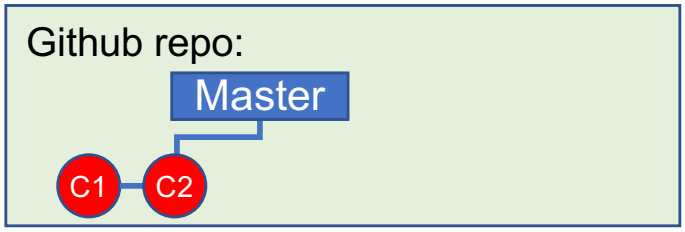
→hit "create" button

You now have a Master branch with 1 commit

→ click on README.md file

→ edit it by adding some text

→ commit changes to Master



Now you have 2 commits

Github cloning

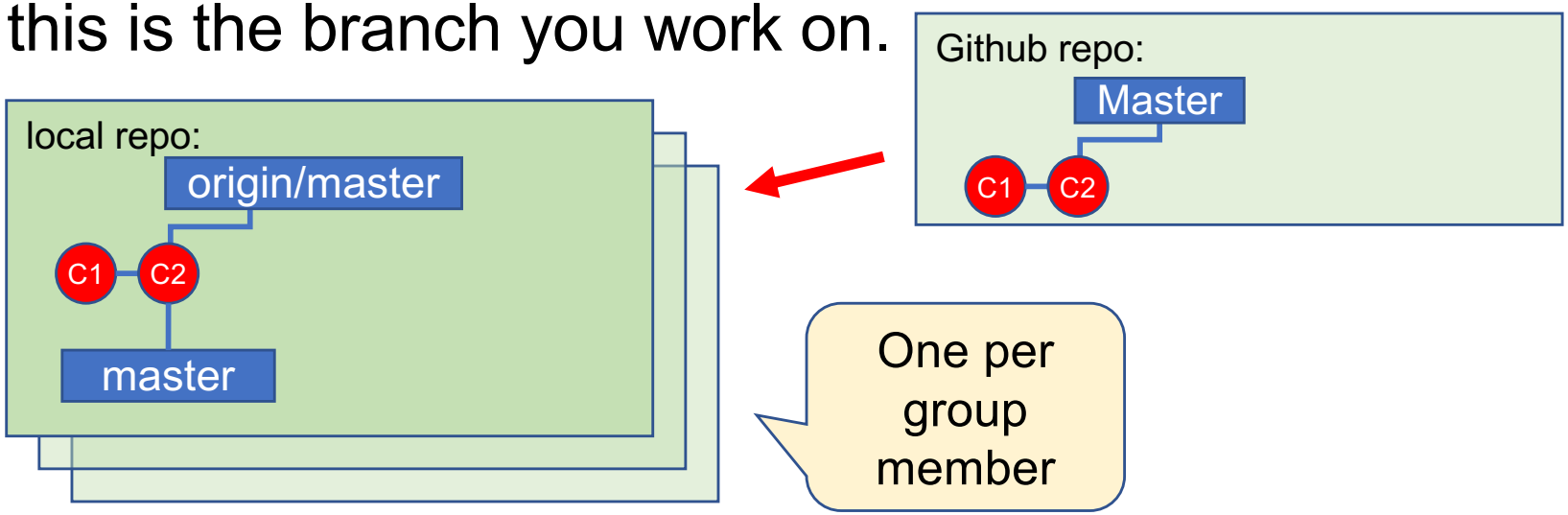
On your laptop make sure you have git installed

Now clone Master from Github

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

This (1) creates a new remote tracking branch: `"origin/master"`
and copies everything from github to this branch

(2) creates a tracking branch called "`master`";
this is the branch you work on.



Github: local commit

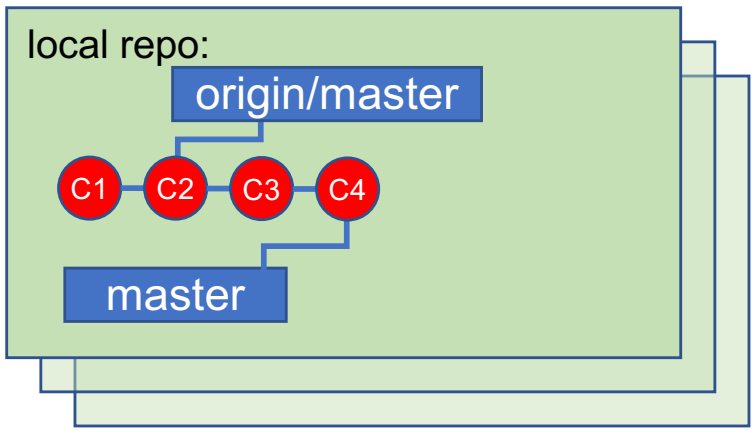
Locally edit README.md file; then

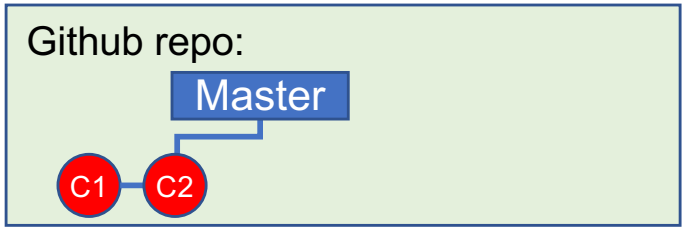
```
> git commit README.md -m "test change"
```

Add new file:

```
> git add Makefile
> echo '#Makefile' > Makefile
> git commit Makefile -m "added makefile"
```

git help commit



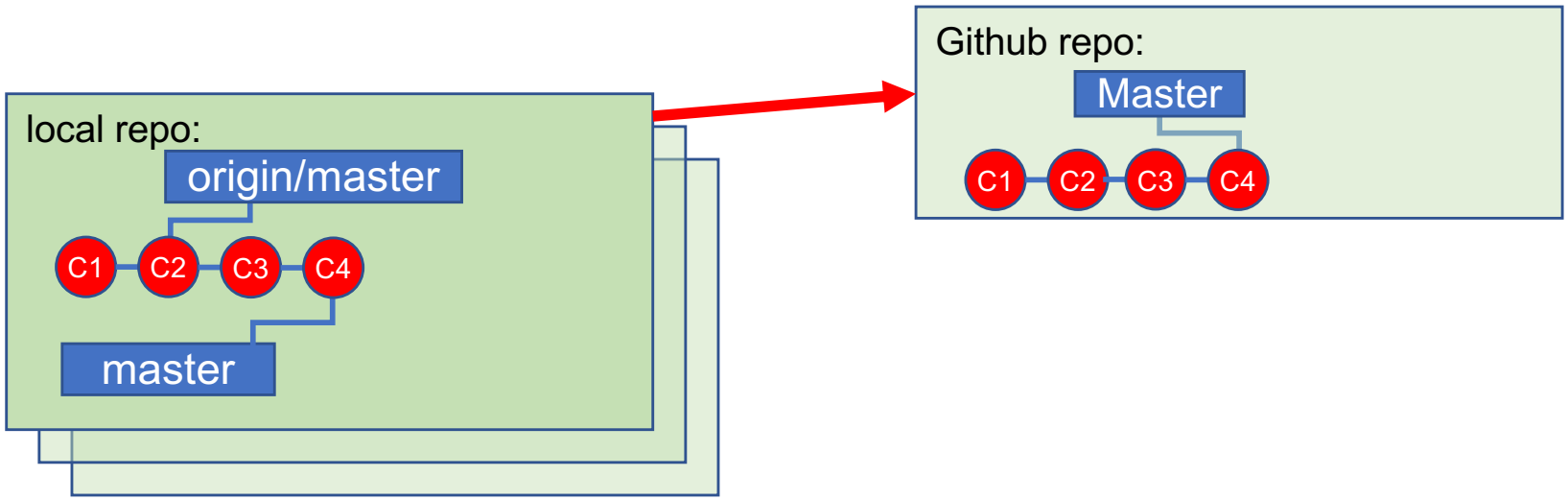


Github: push

Master on Github is not updated until you do a push:

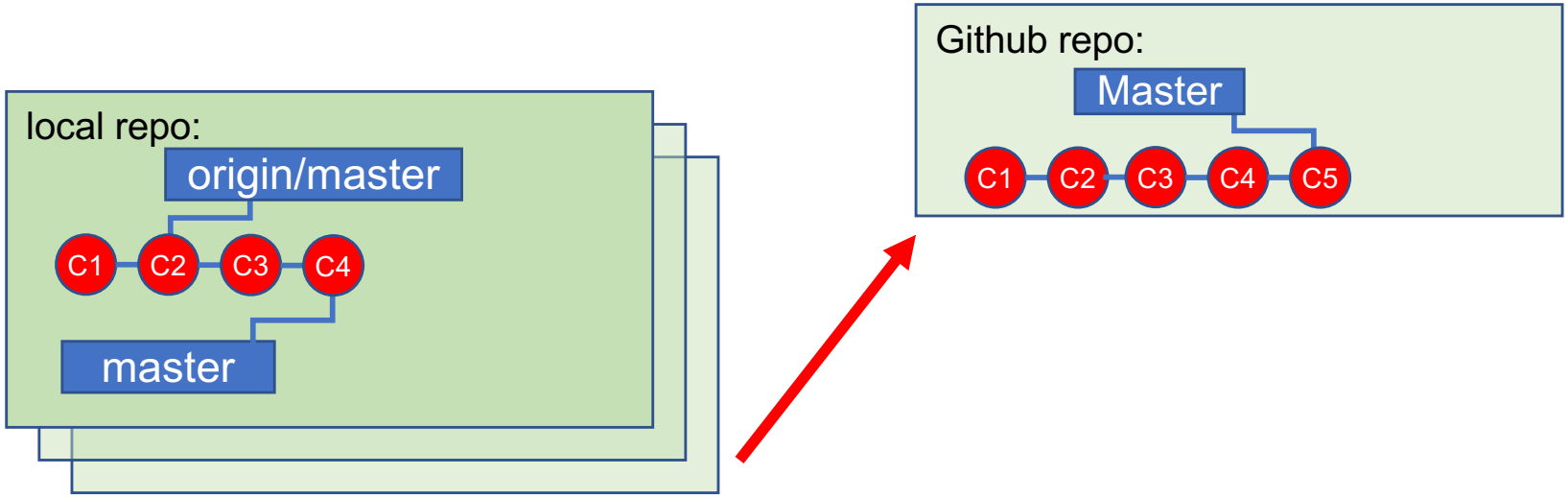
```
> git push
```

May fail if there are conflicts.



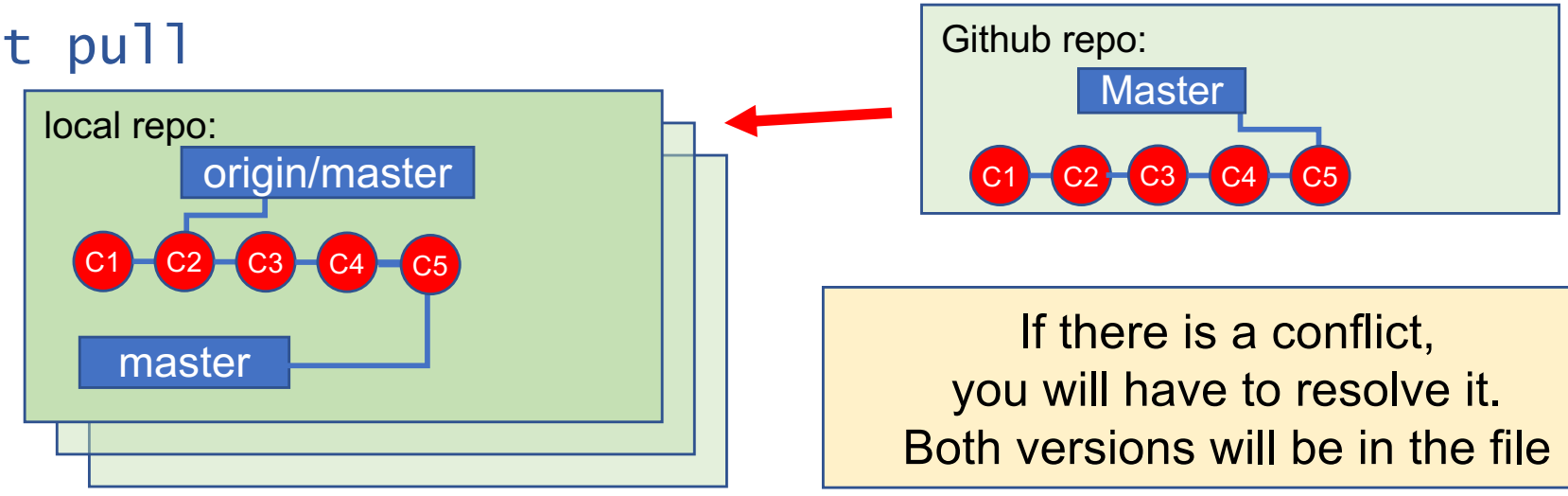
Github: pull

If someone else pushes C5, you will not see it



Until you do a pull

```
> git pull
```



If there is a conflict,
you will have to resolve it.
Both versions will be in the file

Github: other useful git commands

- `git status`
  - see what changes are pending commit and which files not tracked
- `git diff <fname>`
  - diff between current version and last committed version
- `git blame file`
  - annotates each line with who changed it last and when
- many more. . .

- ```
  git config –global user.name "M.Stumm"
  git config –global user.email stumm@eecg. .
  ```

  This is mandatory!
  ```