

TALLER DE GIT(SISTEMA DE CONTROL DE VERSIONES)

INGENIERIA DE SOFTWARE II

PARTE I

Esta primera parte se estudiaran los comandos básicos de git a través de un ejemplo sencillo de una aplicación de Rails, esta parte es individual.

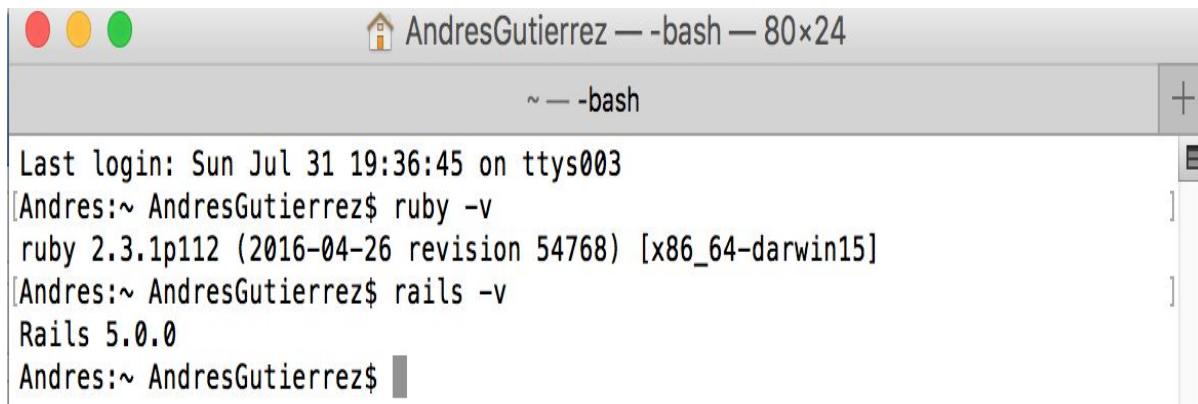
¿Qué es git?

Es un sistema de control de versión distribuido, en el que cada desarrollador está trabajando con una copia del código así como con toda la historia de cambios del proyecto.

Vamos a crear nuestro repositorio

El primer paso que vamos a hacer es crear un nuevo proyecto de Ralis, para esto por favor confirme que tanto Ruby como Ralis se encuentran instalados en el computador.

- Para comprobar que Ruby está instalado ingrese el siguiente comando en consola **ruby -v**
- Para comprobar que Ralis se encuentra instalado ingrese el siguiente comando en consola **rails -v**



The screenshot shows a terminal window titled "AndresGutierrez — -bash — 80x24". The window has three colored tabs (red, yellow, green) at the top left. The title bar also displays the user's name and session type. The terminal prompt is "~ — -bash". The text area contains the following output:

```
Last login: Sun Jul 31 19:36:45 on ttys003
[Andres:~ AndresGutierrez$ ruby -v
ruby 2.3.1p112 (2016-04-26 revision 54768) [x86_64-darwin15]
[Andres:~ AndresGutierrez$ rails -v
Rails 5.0.0
Andres:~ AndresGutierrez$ ]
```

Por cuestiones del curso se estará utilizando la versión de Ruby 2.3 y la última versión de Ralis que se encuentra en fase beta, Ralis 5, los resultados se pueden ver en la siguiente imagen.

Para la creación del proyecto de Ralis nos dirigimos en consola al directorio en el cual se va almacenar el proyecto y lo creamos con el siguiente comando.

rails new myFirstRailsProject

```

RoR — ruby ▾ rails new myFirstRailsProject — 80x24
~/RoR — ruby ▾ rails new myFirstRailsProject
[Andres:~ AndresGutierrez$ cd RoR
[Andres:RoR AndresGutierrez$ ls
demo example newDepot unOverflow
[Andres:RoR AndresGutierrez$ rails new myFirstRailsProject
  create
  create README.md
  create Rakefile
  create config.ru
  create .gitignore
  create Gemfile
  create app
  create app/assets/config/manifest.js
  create app/assets/javascripts/application.js
  create app/assets/javascripts/cable.js
  create app/assets/stylesheets/application.css
  create app/channels/application_cable/channel.rb
  create app/channels/application_cable/connection.rb
  create app/controllers/application_controller.rb
  create app/helpers/application_helper.rb
  create app/jobs/application_job.rb
  create app/mailers/application_mailer.rb
  create app/models/application_record.rb
  create app/views/layouts/application.html.erb
  create app/views/layouts/mailер.html.erb

```

Después comprobamos que la creación del proyecto haya sido exitosa para eso vamos a correr nuestra primera aplicación de Rails, con el comando anteriormente ejecutado, este nos ha creado una carpeta con el mismo nombre del proyecto nos dirigiremos a ella y correremos el comando.

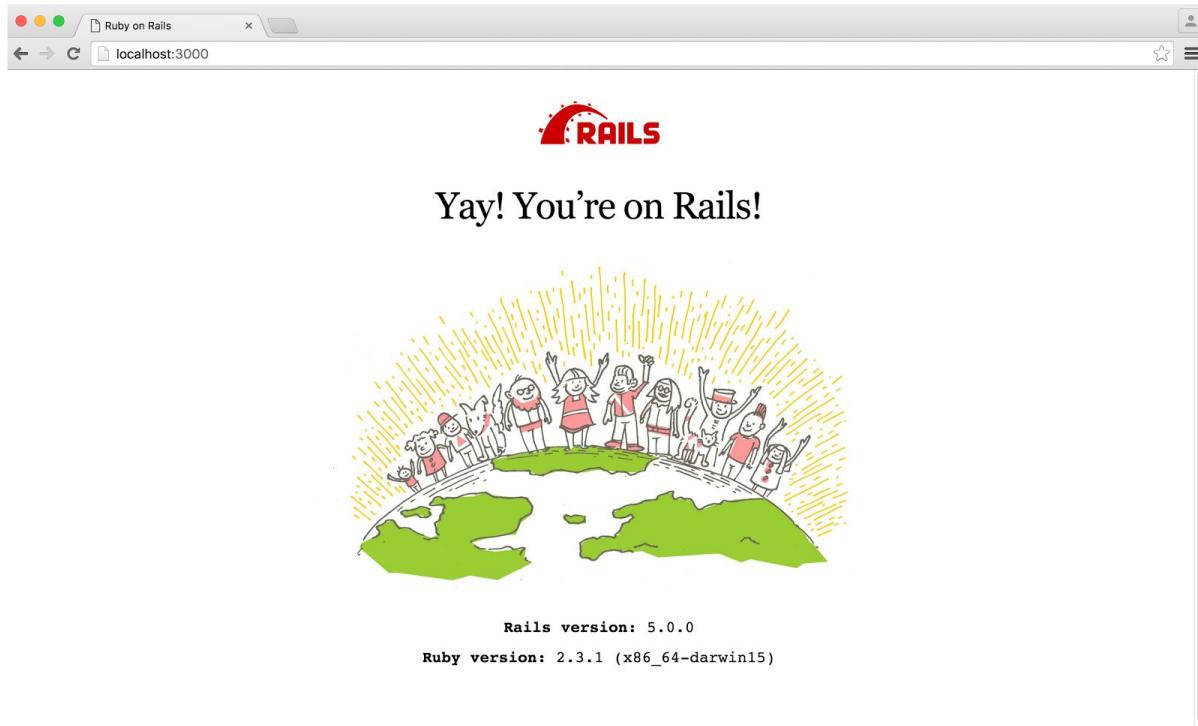
rails sever o para más sencillez solamente ***rails s***

```

bin — fsevent_watch ▾ localhost:3000) [myFirstRailsProject] RBENV_VERSION=2....
fsevent_watch ▾ localhost:3000) [myFirstRailsPr...resGutierrez/.sdkman/candidates/grails/current
[Andres:RoR AndresGutierrez$ ls
demo myFirstRailsProject unOverflow
example newDepot
[Andres:RoR AndresGutierrez$ cd myFirstRailsProject/
[Andres:myFirstRailsProject AndresGutierrez$ ls
Gemfile Rakefile config lib test
Gemfile.lock app config.ru log tmp
README.md bin db public vendor
[Andres:myFirstRailsProject AndresGutierrez$ cd bin/
[Andres:bin AndresGutierrez$ ls
bundle rails rake setup spring update
[Andres:bin AndresGutierrez$ rails s
=> Booting Puma
=> Rails 5.0.0 application starting in development on http://localhost:3000
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.6.0 (ruby 2.3.1-p112), codename: Sleepy Sunday Serenity
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://localhost:3000
Use Ctrl-C to stop

```

Luego de que el servidor está corriendo lo comprobamos en nuestro navegador apuntando a la dirección que aparece en consola.



Felicidades has creado tu primer proyecto en Rails.

Ahora vamos a inicializar nuestro repositorio.

¿Qué es git init?

Este comando crea un nuevo repositorio. Este también puede ser usado para convertir un proyecto no versionado a un repositorio de git o para inicializar un nuevo repositorio vacío, la mayoría de comandos de git no están disponibles a fuera de repositorio así que este comando es usualmente el primero que se ejecuta.

Al correr el comando git init este crear un subdirectorio .git en raíz del proyecto este subdirectorio contiene todos los metadatos acerca del repositorio.

Para crea nuestro repositorio vamos a la raíz del proyecto y corremos git init, después comprobamos que el subdirectorio haya sido creado.

```

.git — -bash — 80x24
~/RoR/myFirstRailsProject/.git — -bash

[Andres:myFirstRailsProject AndresGutierrez$ pwd
/Users/AndresGutierrez/RoR/myFirstRailsProject
[Andres:myFirstRailsProject AndresGutierrez$ ls
Gemfile      Rakefile      config      lib      test
Gemfile.lock  app          config.ru   log      tmp
README.md    bin          db          public   vendor
[Andres:myFirstRailsProject AndresGutierrez$ git init
Initialized empty Git repository in /Users/AndresGutierrez/RoR/myFirstRailsProj
ct/.git/
[Andres:myFirstRailsProject AndresGutierrez$ cd .git
[Andres:.git AndresGutierrez$ ls
HEAD      config      hooks      objects
branches  description  info      refs
Andres:.git AndresGutierrez$ 

```

Sin embargo, para mayoría de los proyectos, git init solamente necesita ser ejecutado una vez para crear el repositorio central, los desarrolladores típicamente no utilizan git init para crear sus repositorios locales, en vez de eso ellos utilizan git clone para copiar un repositorio existente en sus máquinas locales.

¿Qué es git clone?

git clone copia un repositorio de git existente, el repositorio de git copiado tiene su propia historia y sus propios archivos y es completamente aislado del entorno del original repositorio.

Cuando se clona el repositorio se crea automáticamente una conexión al repositorio original llamada origin, esto sirve para interactuar de una manera sencilla con el repositorio central.

Por el momento no nos concentraremos en este comando, volveremos a hablar de este en la parte II.

¿Qué es git config?

Este comando permite configurar la instalación de git globalmente o para un repositorio en particular.

- **git config user.name <name>**: define el nombre del autor para todos los commits en el repositorio actual si tú quieres que esta configuración sea permanente para todos tus repositorios tu deberías usar **git config --global user.name <name>**.

Para edición rápida de tus preferencias tu puedes usar.

- **git config --edit** o **git config --global --edit**: para editar todas las preferencias para tu repositorio actual o para todos tus repositorios respectivamente, al ejecutar este comando se abrirá tu editor de texto automáticamente, recomiendo tener instalado vi o nano para una edición rápida.

```
~/RoR/myFirstRailsProject/.git — nano - git config --global --edit
GNU nano 2.0.6      File: /Users/AndresGutierrez/.gitconfig

[user]
  name = Andres Gutierrez
  email = agutierrezt@unal.edu.co
[core]
  editor = nano
  excludesfile = /Users/AndresGutierrez/.gitignore_global
  excludesfile = /Users/AndresGutierrez/.gitignore_global
[difftool "sourcetree"]
  cmd = opendiff \"$LOCAL\" \"$REMOTE\""
  path =
[mergetool "sourcetree"]
  cmd = /Applications/SourceTree.app/Contents/Resources/opendiff-w.sh \"\$\$"
  trustExitCode = true
[commit]
  template = /Users/AndresGutierrez/.stCommitMsg

[ Read 15 lines ]
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U Uncut Text ^T To Spell
```

¿Qué es un archivo .gitignore?

Git utiliza este archivo para determinar cuáles archivos y directorios ignorar antes de hacer un commit este archivo debería ser subido al repositorio central para compartir las reglas con otros usuarios que clonen el repositorio, no hay nada de qué preocuparse Rails es bastante inteligente y ya ha creado este archivo por nosotros si no lo puedes encontrar con el comando ls en el terminal no te preocunes los archivos que comienzan con . son ignorados por este comando si quieras ver el archivo ve a la raíz de la aplicación e ingresa el comando ls -a.

```
myFirstRailsProject — -bash — 80x24
~/RoR/myFirstRailsProject — -bash

[Andres:myFirstRailsProject AndresGutierrez$ pwd
/Users/AndresGutierrez/RoR/myFirstRailsProject
[Andres:myFirstRailsProject AndresGutierrez$ ls -a
.          Gemfile      app        db        test
..         Gemfile.lock  bin        lib        tmp
.git       README.md    config     log        vendor
.gitignore  Rakefile    config.ru public
Andres:myFirstRailsProject AndresGutierrez$
```

para conocer que tiene el archivo utiliza tu editor favorito la siguiente imagen muestra el contenido de este archivo.

The screenshot shows a terminal window with the title bar reading "GNU nano 2.0.6 File: .gitignore". The main area contains the following .gitignore content:

```
# See https://help.github.com/articles/ignoring-files for more about ignoring files
#
# If you find yourself ignoring temporary files generated by your text editor
# or operating system, you probably want to add a global ignore instead:
#   git config --global core.excludesfile '~/.gitignore_global'

# Ignore bundler config.
/.bundle

# Ignore the default SQLite database.
/db/*.sqlite3
/db/*.sqlite3-journal

# Ignore all logfiles and tempfiles.
/log/*
/tmp/*
!/log/.keep
!/tmp/.keep
```

At the bottom of the terminal window, there is a menu of keyboard shortcuts:

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U Uncut Text ^T To Spell

Bueno ya has entendido las bases de git entonces continuemos con algo más divertido, vamos a hacer nuestro primer commit.

¿Qué es Staging Area?

El staging área es una de las características más importantes en git, es como un buffer entre el directorio de trabajo y la historia del proyecto, en vez de preparar todos los cambios de una vez para hacer commit el staging área permite agrupar cambios relacionados en diferentes snapshots antes de hacer commit a la historia del proyecto.

¿Qué es git add?

Añade un cambio en el directorio de trabajo al staging área, esto permite conocer a git que tú quieres incluir las actualizaciones de un archivo en el siguiente commit, git add no afecta el repositorio de ninguna manera, los cambios no son en realidad grabados hasta que uno ejecuta git commit.

¿Qué es git commit?

Este comando sirve para hacer commit de la snapshot a la historia del proyecto. ***Las snapshots son realizadas en el repositorio local y no interfiere con ninguno de los otros repositorios de los colaboradores.***

El comando **git commit** abre el editor definido en la configuración para que el usuario entre un mensaje acerca de los cambios del proyecto, si este mensaje no es largo se puede usar el comando **git commit -m "<message>"** para un mensaje de commit en una sola línea.

¿Qué es git status?

Muestra el estado del directorio de trabajo y del staging área; esto permite identificar cuales archivos o directorios se ha modificado y se han preparado para hacer commit; permite también identificar cuales archivos no están siendo seguidos por git. En otras palabras se les ha hecho seguimiento con git add.

Git status muestra tres tipos de archivo

staged: Los archivos que han sido preparados para el siguiente commit. Estos archivos fueron añadidos con *git add*.

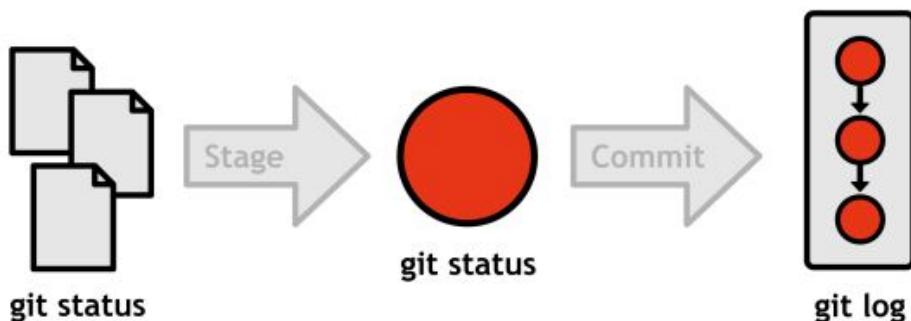
unstaged: Archivos que ya fueron añadidos en anteriores commits a la historia del proyecto y que presentan cambios nuevos.

untracked: Son archivos nuevos y que nunca han sido añadidos con *git add*.

¿Qué es git log?

Muestra los snapshots a los cuales ya se le hicieron commit, este muestra la historia del proyecto y permite filtrar y buscar por cambios específicos, mientras git status permite inspeccionar el directorio de trabajo y el stagingárea, git log solo opera el historia del proyecto.

- ***git log -n <limit>***: muestra los últimos commits especificados en limit
- ***git log --oneline***: muestra los commits en una sola línea.
- ***git log --stat***: incluye cuales archivos fueron alterados y las relativas líneas que fueron añadidas y borradas.
- ***git log -p***: muestra las diferencias entre cada commit.
- ***git log --author=<patern>***: muestra los commits para el autor especificado, este puede ser un texto plano o una expresión regular.
- ***git log --grep= "<patern>"***: busca commits que coincidan con el patrón.
- ***git log <since>..<until>***: muestra commits que ocurran entre <since> y <until> los argumentos pueden ser un commit ID, un nombre de una rama otro tipo de argumento válido.
- ***git log <file>***: busca commits que incluyan este archivo.



Ya conocemos los básicos para crear snapshots de nuestro proyecto vamos a crear nuestro primer commit.

1. Primero vamos a la raíz del proyecto y ejecutamos **git status** para ver el estado de nuestro proyecto.

```
cd
```

```
myFirstRailsProject — bash — 80x27
~/RoR/myFirstRailsProject — bash
[Andres:myFirstRailsProject AndresGutierrez$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  .gitignore
  Gemfile
  Gemfile.lock
  README.md
  Rakefile
  app/
  bin/
  config.ru
  config/
  db/
  lib/
  log/
  public/
  test/
  tmp/
  vendor/

nothing added to commit but untracked files present (use "git add" to track)
Andres:myFirstRailsProject AndresGutierrez$
```

2. Luego utilizamos el **git add .gitignore** para seguir este archivo y luego comprobamos como está el estado del proyecto antes de hacer commit.

```
myFirstRailsProject — bash — 80x31
~/RoR/myFirstRailsProject — bash
[Andres:myFirstRailsProject AndresGutierrez$ git add .gitignore
[Andres:myFirstRailsProject AndresGutierrez$ git status
On branch master

Initial commit

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

    new file:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  Gemfile
  Gemfile.lock
  README.md
  Rakefile
  app/
  bin/
  config.ru
  config/
  db/
  lib/
  log/
  public/
  test/
  tmp/
  vendor/

Andres:myFirstRailsProject AndresGutierrez$
```

3. Antes de hacer el commit vamos a añadir el resto de archivos a nuestro staging área una forma rápida de hacer esto es utilizar el comando **git add .** el cual añade todos los archivos que se encuentran en la raíz de nuestro proyecto.

```

myFirstProject — -bash — 85x30
[Lauras-MacBook-Air:myFirstProject lpceron$ git add .
[Lauras-MacBook-Air:myFirstProject lpceron$ git status
On branch master

Initial commit

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

    new file:  .gitignore
    new file:  Gemfile
    new file:  Gemfile.lock
    new file:  README.md
    new file:  Rakefile
    new file:  app/assets/config/manifest.js
    new file:  app/assets/images/.keep
    new file:  app/assets/javascripts/application.js
    new file:  app/assets/javascripts/cable.js
    new file:  app/assets/javascripts/channels/.keep
    new file:  app/assets/stylesheets/application.css
    new file:  app/channels/application_cable/channel.rb
    new file:  app/channels/application_cable/connection.rb
    new file:  app/controllers/application_controller.rb
    new file:  app/controllers/concerns/.keep
    new file:  app/helpers/application_helper.rb
    new file:  app/jobs/application_job.rb
    new file:  app/mailers/application_mailer.rb
    new file:  app/models/application_record.rb
    new file:  app/models/concerns/.keep
    new file:  app/views/layouts/application.html.erb

```

4. Por último vamos a ejecutar el comando ***git commit*** para hacer estos cambios permanentes en la historia del proyecto y vemos el ***git log***.

```

logs — -bash — 80x31
~/RoR/myFirstRailsProject/.git/logs — -bash
[Andres:logs AndresGutierrez$ git log
commit 1292fa3a4bbbae0b45be4d56b76f850c4ea963b0
Author: Andres Gutierrez <agutierrezt@unal.edu.co>
Date:   Tue Aug 2 17:51:40 2016 -0500

  Set up

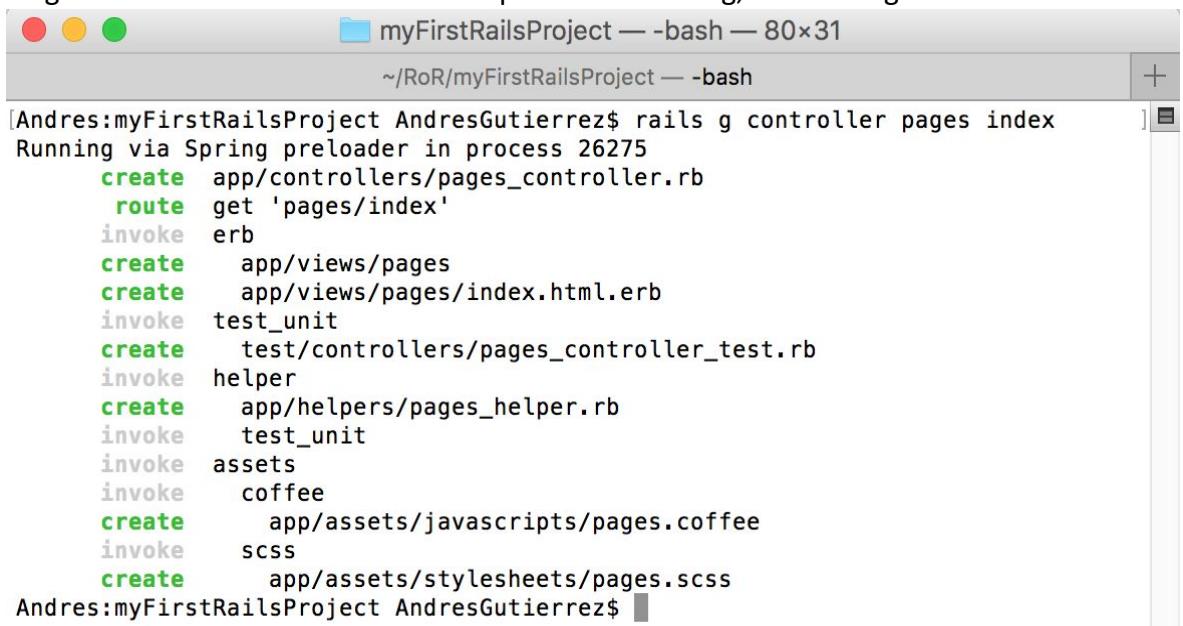
  This is our first commit, we are learning git and have a lot of fun
Andres:logs AndresGutierrez$ ]

```

Para este momento ya eres un experto en git entonces vamos a hacer algunos commits mas.

1. Primero vamos a crear un nuevo controlador en Rails llamado pages con una función llamada index, vamos a la raíz del proyecto corremos el comando ***rails g controller pages index***.

2. Despues vamos a volver esta acción con la raíz de la aplicación por lo tanto nos dirigimos a la carpeta config, luego routes



```
[Andres:myFirstRailsProject AndresGutierrez$ rails g controller pages index
Running via Spring preloader in process 26275
  create  app/controllers/pages_controller.rb
  route   get 'pages/index'
invoke  erb
  create  app/views/pages
  create  app/views/pages/index.html.erb
invoke  test_unit
  create  test/controllers/pages_controller_test.rb
invoke  helper
  create  app/helpers/pages_helper.rb
invoke  test_unit
invoke  assets
invoke  coffee
  create    app/assets/javascripts/pages.coffee
invoke  scss
  create    app/assets/stylesheets/pages.scss
Andres:myFirstRailsProject AndresGutierrez$ ]
```

.rb y cambios la predefina por **root to: 'pages#index'**, luego añadimos los archivos a el staging área y hacemos el commit.

A screenshot of a Mac OS X desktop environment. On the left, a file browser window titled "myFirstProject" shows the project structure. It includes a ".git" folder, an "app" folder, a "bin" folder, a "config" folder containing "environments", "initializers", and "locales" subfolders, and files like "application.rb", "boot.rb", "cable.yml", "database.yml", "environment.rb", "puma.rb", and "routes.rb". The "routes.rb" file is selected and highlighted in orange. On the right, a code editor window titled "routes.rb" displays the following Ruby code:

```
1 Rails.application.routes.draw do
2   get 'pages/index'
3 
4   # For details on the DSL available within this
5   # root to: 'pages#index'
6 end
```

A screenshot of a terminal window titled "myFirstRailsProject — -bash — 80x31". The window shows the output of a "git status" command:

```
Andres:myFirstRailsProject AndresGutierrez$ git status
On branch master
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:   config/routes.rb

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    app/assets/javascripts/pages.coffee
    app/assets/stylesheets/pages.scss
    app/controllers/pages_controller.rb
    app/helpers/pages_helper.rb
    app/views/pages/
    test/controllers/pages_controller_test.rb

no changes added to commit (use "git add" and/or "git commit -a")
Andres:myFirstRailsProject AndresGutierrez$
```

3. Luego vamos a la carpeta app, views, pages y modificamos el html del index para que muestre su nombre y volemos hacer un commit, a continuación se muestra el log de cómo debería quedar y la aplicación corriendo.

The screenshot shows a Mac OS X desktop environment. On the left, a file browser window titled "myFirstProject" is open, showing the project structure:

- myFirstProject
- .git
- app
 - assets
 - channels
 - controllers
 - helpers
 - jobs
 - mailers
 - models
 - views
 - layouts
 - pages
- index.html.erb

On the right, a terminal window titled "index.html.erb — ~/Documents/myFirstProject" is open, showing the content of the file:

```
<h1>Welcome to the course of Software Engineering II</h1>
<p>Andres y Laura </p>
```



Welcome to the course of Software Engineering II

Andres y Laura

The screenshot shows a terminal window titled "myFirstRailsProject — -bash — 80x31". The command "git log --oneline" was run, showing the following commit history:

```
[Andres:myFirstRailsProject AndresGutierrez$ git log --oneline
283bc91 Views Modify the main view of the app, now it shows our name
9f24c88 Our first commit
1292fa3 Set up
Andres:myFirstRailsProject AndresGutierrez$ ]
```

¿Qué es git checkout?

Este comando sirve para 3 funciones: para revisar archivos, commits y ramas.

Revisar un commit hace que todo el directorio de trabajo coincida con ese commit. Esto puede ser usado para ver un viejo estado del proyecto sin alterar el actual estado, revisar archivos permite ver versiones viejas de un particular archivo dejando el resto de tu directorio de trabajo intacto.

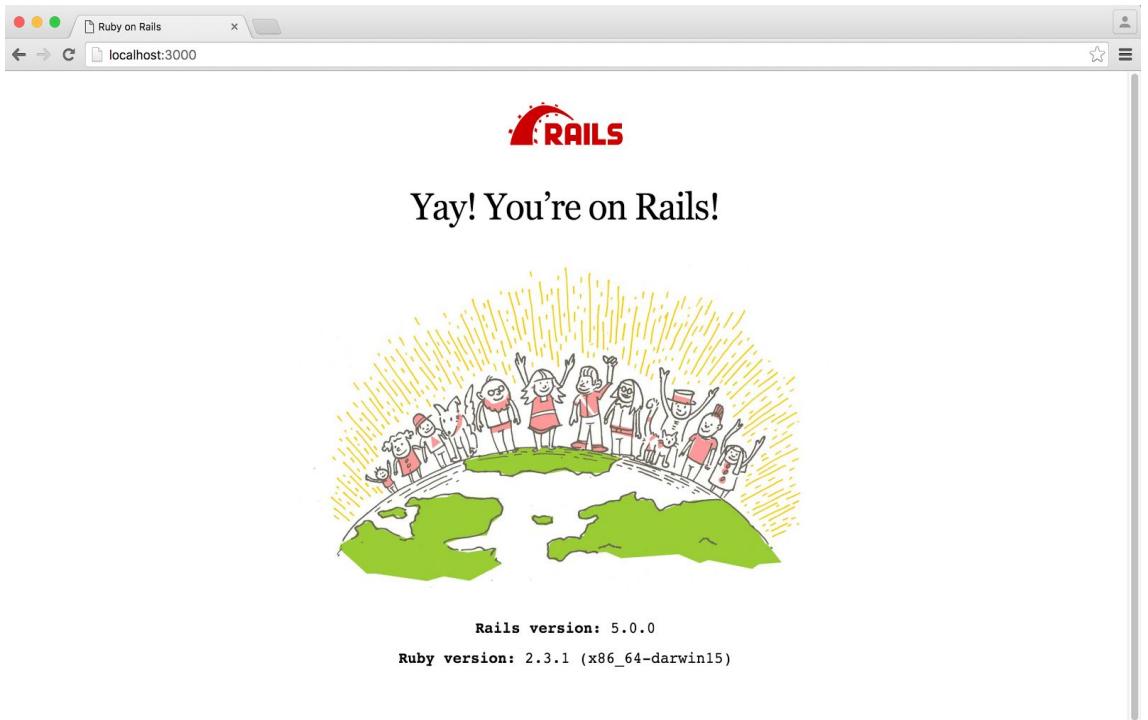
- ***git checkout <commit> <file>***
- ***git checkout <commit>***

Cuando uno revisa un commit esta versión será de solo lectura, es usada para comprobar fases anteriores del proyecto y decidir si uno quiere volver a ellas, para volver al estado actual del proyecto ejecute ***git checkout master***.

Revisando archivos es algo diferente, ya que este no afecta el actual estado del proyecto, la versión antigua del archivo se muestra como un cambio para hacer commit, esta es una forma fácil de volver a versiones más estables de un archivo si un experimento fallo, para volver a la versión más reciente del archivo ***git checkout HEAD <file>***.

Bueno ya que has aprendido la teoría vamos a ver cómo funciona.

1. Primero vas a la raíz de la aplicación y ejecutamos ***git log -oneline*** y buscamos el hash del primer commit que hicimos.



```
fsevent_watch - localhost:3000 [myFirstRailsProject] RBE...
fsevent_watch - localhost:3000 [myFirstRailsPr...resGutierrez/.sdkman/candidates/grails/current] + 
[Andres:myFirstRailsProject AndresGutierrez$ git checkout 1292fa3
Note: checking out '1292fa3'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 1292fa3... Set up
[Andres:myFirstRailsProject AndresGutierrez$ rails s
=> Booting Puma
=> Rails 5.0.0 application starting in development on http://localhost:3000
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.6.0 (ruby 2.3.1-p112), codename: Sleepy Sunday Serenity
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://localhost:3000
Use Ctrl-C to stop
```

2.

Luego ejecutamos **git checkout <hash>** y corremos **rails s** para comprobar cómo fue la etapa inicial de nuestro proyecto.

3. Por ultimo ejecutamos **git checkout master**.

Ha sido un largo camino hasta ahora y como uno sabe no siempre las cosas salen bien cuando uno está desarrollando un proyecto, pero gracias a git podemos deshacer algunos cambios que terminaron mal.

¿Qué es git revert?

Este comando deshace un commit pero en vez de remover el commit de la historia del proyecto este deshace los cambios introduciendo un nuevo commit con el resultado deseado, esto previene a git de perder la historia del proyecto, la cual es importante para la integridad.

- **git revert <commit>**

¿Qué es git reset?

Devuelve el proyecto a un estado antiguo, perdiendo la historia del proyecto, cuando se ejecuta este comando los cambios son perdidos permanentemente, este comando debería ser usado con cuido y siempre en cambios locales nunca con cambios que ya han sido compartidos con otros desarrolladores.

- **git reset <file>**: remueve el archivo especificado de el staging área pero deja el directorio del trabajo sin cambios, dejando los “unstages” archivos sin sobrescribir ningún cambio.
- **git reset**: resetea el staging área para asociar al más reciente commit pero deja el directorio de trabajo sin cambios, “unstages” todos los archivos sin sobrescribir ningún cambio, dando la oportunidad de reconstruir el snapshot desde el inicio.
- **git reset --hard**: resetea el stagingárea y el directorio de trabajo para asociar al más reciente commit, en adición unstanging los cambios, la bandera --hard hace que git sobrescribe los cambios en el directorio de trabajo también, borrando todos los cambios sin commit.
- **git reset <commit>**: mueve la rama actual hacia atrás al commit especificado resemando el stagingárea para asociar al commit especificado, todos los cambios hechos desde el commit residen el directorio de trabajo para re hacer la historia del proyecto.
- **git reset --hard <commit>**: mueve la rama actual hacia atrás al commit especificado resemando el stagingárea para asociar al commit especificado, esto borra todos los cambios sin commit como también todos cambios después del commit especificado.

Git REVERT vs RESET

Es importante mencionar que **git revert** deshace un solo commit, en cambio **git reset** retorna el proyecto a un estado previo eliminando todos los commits hechos después de este.

Tomada de: <http://alexdiliberto.com/talks/all-things-git/#/>

¿Qué es git clean?

Remueve archivos sin seguimiento desde el directorio de trabajo, este comando no se puede deshacer.

- **git clean -n:** realiza un “dry run” del comando git clean, muestra que archivos van a ser removidos sin en realidad hacerlo.
- **git clean -f:** remueve los archivos sin seguimiento del actual directorio de trabajo
- **git clean -f <path>:** remueve los archivos sin seguimiento pero lo limita al path especificado
- **git clean -df:** remueve los archivos y directorios sin seguimiento del directorio actual
- **git clean -xf:** remueve archivos sin seguimiento del actual directorio como también los archivos que usualmente git ignora.

Vamos a deshacer los cambios que hemos hecho en nuestro proyecto.

1. Vamos a correr el comando **git revert <commit>**, para volver al segundo commit que hicimos, ya que el controlador y la ruta que creamos no era lo que esperábamos.

The screenshot shows a macOS terminal window with two tabs. The active tab is titled "myFirstRailsProject — nano" and contains the following text:

```
Revert "Our first commit"

This reverts commit 9f24c888da8c0363094250d476fea12a464a4385.

# Conflicts:
#       app/views/pages/index.html.erb

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# You are currently reverting commit 9f24c88.
#
# Changes to be committed:
#       deleted:    app/assets/javascripts/pages.coffee
#       deleted:    app/assets/stylesheets/pages.scss
#       deleted:    app/controllers/pages_controller.rb
#       deleted:    app/helpers/pages_helper.rb
#       deleted:    app/views/pages/index.html.erb
#       modified:   config/routes.rb
#       deleted:    test/controllers/pages_controller_test.rb
```

At the bottom of the terminal window, there is a menu of keyboard shortcuts:

- [Read 21 lines]
- ^G Get Help
- ^O WriteOut
- ^R Read File
- ^Y Prev Page
- ^K Cut Text
- ^C Cur Pos
- ^X Exit
- ^J Justify
- ^W Where Is
- ^V Next Page
- ^U UnCut Text
- ^T To Spell

2. Luego vamos a correr **git log** para ver cómo ha quedo la historia de nuestro proyecto.

```
~/RoR/myFirstRailsProject — less • git log
commit b21fde424db4c29200fb79e54e5df53006d59868
Author: Andres Gutierrez <agutierrez@unal.edu.co>
Date: Tue Aug 2 22:56:02 2016 -0500

    Revert "Our first commit"

    This reverts commit 9f24c888da8c0363094250d476fea12a464a4385.

commit 45c8293af8561a42df7509734cccd3cad13b941f0
Author: Andres Gutierrez <agutierrez@unal.edu.co>
Date: Tue Aug 2 21:24:17 2016 -0500

    Some file were modified

commit 283bc91d44ec0eb2136df8d87f3231e371efc345
Author: Andres Gutierrez <agutierrez@unal.edu.co>
Date: Tue Aug 2 18:43:00 2016 -0500

    Views
    Modify the main view of the app, now it shows our name

commit 9f24c888da8c0363094250d476fea12a464a4385
Author: Andres Gutierrez <agutierrez@unal.edu.co>
Date: Tue Aug 2 18:39:44 2016 -0500

    Our first commit

commit 1292fa3a4bbbae0b45be4d56b76f850c4ea963b0
Author: Andres Gutierrez <agutierrez@unal.edu.co>
Date: Tue Aug 2 17:51:40 2016 -0500
:
```

3. Por ultimo vamos a correr git reset <commit> para volver al primer commit de nuestro proyecto ya que no queremos mantener ningn n registro de lo que hemos hecho.

```
myFirstRailsProject — -bash — 80x31
~/RoR/myFirstRailsProject — -bash
[Andres:myFirstRailsProject AndresGutierrez$ git reset 1292fa3
[Andres:myFirstRailsProject AndresGutierrez$ git log
commit 1292fa3a4bbbae0b45be4d56b76f850c4ea963b0
Author: Andres Gutierrez <agutierrez@unal.edu.co>
Date: Tue Aug 2 17:51:40 2016 -0500

    Set up

    This is our first commit, we are learning git and have a lot of fun
[Andres:myFirstRailsProject AndresGutierrez$ git status
On branch master
nothing to commit, working directory clean
Andres:myFirstRailsProject AndresGutierrez$
```

Ahora vamos a reescribir la historia de nuestro repositorio.

¿Qué es git amend?

Este comando es una manera conveniente para arreglar el más reciente commit, vamos a combinar cambios con el commit anterior en vez de hacer un nuevo snapshot en la historia del proyecto, este también puede ser utilizado para editar el mensaje del commit anterior sin cambiar el snapshot.

- **git commit --amend**

```
myFirstRailsProject — -bash — 80x31
~/RoR/myFirstRailsProject — -bash

[Andres:myFirstRailsProject AndresGutierrez$ git log
commit b3234d73a16578d9f5b1292d8051782e014ac57e
Author: Andres Gutierrez <agutierrezt@unal.edu.co>
Date: Tue Aug 2 17:51:40 2016 -0500

  Set up

  This is our first commit, we are learning git and have a lot of fun

  Thanks to follow this tutorial
Andres:myFirstRailsProject AndresGutierrez$ ]|+]

GNU nano 2.0.6 File: ...RoR/myFirstRailsProject/.git/COMMIT_EDITMSG Modified | [ ]|+]
```

¿Qué es git reflog?

Git mantiene las actualizaciones de todo el proyecto, esto permite volver a los cambios incluso si ellos no están referenciados por ninguna rama o tag, después de reescribir la historia el reflog contiene información acerca de los viejos estados de las ramas y permite volver a los estados anteriores si es necesaria.

Vamos a aplicar los conceptos a nuestro proyecto.

1. En la raíz del proyecto ingrese el comando **git commit --amend** y edite el commit inicial.
2. Consulte el **git reflog** para ver todo lo que hicimos durante la parte 1

Felicidades has completado la parte uno del tutorial

PARTE II

Para esta sección vamos a trabajar en grupos por favor reúnanse en los grupos de los proyectos, si algún uno de los usuarios no tiene cuenta en [Github](#) por favor créenla antes de empezar ya que vamos a tener nuestro repositorio central allí.

¿Qué es git remote?

Este comando permite ver, crear y borrar conexiones a otros repositorios. Las conexiones remotas son más como marcadores que links directos a otros repositorios, en vez de proporcionar acceso en tiempo real a otro repositorios, ellos sirven como nombres convenientes que pueden ser usados para referenciar los URLs.

- **git remote -v**: muestra las conexiones a otros repositorios con su respectiva url.
- **git remote add <name> <url>**: crea una nueva conexión a un repositorio remoto.
- **git remote rm <name>**: remueve el repositorio remoto llamado <name>.
- **git remote rename <old_name> <new_name>**: renombra la conexión al repositorio remoto.

1. El scrum master del equipo va a crear una organización en github con el nombre nombreDelProyectoUNAL y luego va invitar a sus compañeros de equipo para que contribuyan en él.

The screenshot shows the GitHub homepage. A context menu is open in the top right corner with options: 'New repository', 'Import repository', and 'New organization'. Below the menu, a call-to-action box says 'Learn Git and GitHub without any code!' with 'Read the guide' and 'Start a project' buttons. At the bottom left, a notification for 'Andres930410' says 'You've been added to the TuApp organization!'. On the right, there's a sidebar for 'Reorder issues within a milestone' and a list of repositories contributed to: 'Tuapp2016/Sokol', 'IS2-UNAL/UNOverflow', 'TuApp/Recapp', and 'SoftwareArchit... /examsUnal'.

The screenshot shows the GitHub interface for inviting organization members. At the top, there's a navigation bar with links like Facebook, Twitter, Wikipedia, Development, Noticias, Populares, SE2, Syncing, and Atlassian Git Tutorial. Below the navigation bar, there are tabs for This organization and Search, and links for Pull requests, Issues, and Gist.

The main title is "Invite organization members". The process is divided into three steps:

- Completed**: Set up a personal account (indicated by a green checkmark)
- Step 2:** Set up the organization (indicated by a green person icon)
- Step 3:** Invite members (indicated by a blue document icon)

On the left, there's a search bar with placeholder text "Search by username, full name or email address" and a dropdown showing a result for "alejandroariasz". Below the search bar is a "Finish" button.

On the right, under "Organization members", there's a list of permissions:

- ✓ See all repositories
- ✓ Create repositories
- ✓ Organize into teams
- ✓ Review code
- ✓ Communicate via @mentions

Below this, there's a note for organization owners: "As an organization owner, you'll have complete access to all of the organization's repositories and have control of what members have access using fine-grained permissions." It also mentions the ability to change billing info and cancel.

2. Despues de haber creado la organización vamos a crear nuestro primer repositorio llamado practiGit

The image shows two screenshots side-by-side. The top screenshot is a GitHub repository page for 'practicaGitIngeSoft / practicaGit'. It displays sections for giving access to collaborators, quick setup via desktop or command line, and creating a new repository or pushing an existing one. The bottom screenshot is a terminal window titled 'ingesoftPracticaGit — bash — 80x31' showing the creation of a local Git repository and its connection to a remote origin on GitHub.

GitHub Repository Page:

- Give access to the people you work with:** You should give access to the collaborators and teams you need to work with. [Add teams and collaborators](#)
- Quick setup — if you've done this kind of thing before:**
 - [Set up in Desktop](#) or [HTTPS](#) [SSH](#) <https://github.com/practicaGitIngeSoft/practicaGit.git>
 - We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).
- ...or create a new repository on the command line:**

```
echo "# practicaGit" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/practicaGitIngeSoft/practicaGit.git
git push -u origin master
```
- ...or push an existing repository from the command line:**

```
git remote add origin https://github.com/practicaGitIngeSoft/practicaGit.git
git push -u origin master
```

Terminal Session:

```
[Andres:ingesoftPracticaGit AndresGutierrez$ pwd
/Users/AndresGutierrez/RoR/ingesoftPracticaGit
[Andres:ingesoftPracticaGit AndresGutierrez$ git init
Initialized empty Git repository in /Users/AndresGutierrez/RoR/ingesoftPracticaGit/.git/
[Andres:ingesoftPracticaGit AndresGutierrez$ git remote add origin https://github.com/practicaGitIngeSoft/practicaGit.git
[Andres:ingesoftPracticaGit AndresGutierrez$ git remote -v
origin https://github.com/practicaGitIngeSoft/practicaGit.git (fetch)
origin https://github.com/practicaGitIngeSoft/practicaGit.git (push)
[Andres:ingesoftPracticaGit AndresGutierrez$ ]]
```

3. Antes de hacer cualquier cosa en los settings del nuevo repositorio cree un equipo, nómbralo como development y agregue a su compañeros de grupo, luego de esto asígneles permisos al grupo para que pueda tanto leer como escribir en el repositorio.

Create new team

Team name
development ✓

Description (optional)
What is this team all about?

Team visibility

Visible Recommended
A visible team can be seen and @mentioned by every member of this organization.

Secret
A secret team can only be seen by its members.

Create team

4. Ahora el scrum master va crear un nuevo proyecto de Rails como hicimos al principio de la práctica con el comando **rails new nombreDelProyectoPracticaGit** y comprueban con el comando **rails s** que el proyecto haya sido creado exitosamente.

```
[Andres:RoR AndresGutierrez$ ls
ingesoftPracticaGit      newDepot
myFirstRailsProject      unOverflow
[Andres:RoR AndresGutierrez$ cd ingeso...
[Andres:ingesoftPracticaGit AndresGutierrez$ rails s
=> Booting Puma
=> Rails 5.0.0 application starting in development on http://localhost:3000
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.6.0 (ruby 2.3.1-p112), codename: Sleepy Sunday Serenity
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://localhost:3000
Use Ctrl-C to stop]
```

5. El scrum master va a crear el git con el comando **git init** luego va añadir el remote del proyecto y va hacer su primer commit.

```

nombreDelProyectoPracticaGit — bash — 110x30
Lauras-MacBook-Air:Documents lpceron$ cd nombreDelProyectoPracticaGit/
Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ git init
Initialized empty Git repository in /Users/lpceron/Documents/nombreDelProyectoPracticaGit/.git/
Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ git remote add origin https://github.com/nombreDelProyectoUnals2/practicaGit.git
Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ git add .gitignore
Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ git add .
Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ git status
On branch master

Initial commit

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

    new file:  .gitignore
    new file:  Gemfile
    new file:  Gemfile.lock
    new file:  README.md
    new file:  Rakefile
    new file:  app/assets/config/manifest.js
    new file:  app/assets/images/.keep
    new file:  app/assets/javascripts/application.js
    new file:  app/assets/javascripts/cable.js
    ...

ingesoftPracticaGit — bash — 80x31
~/RoR/ingesoftPracticaGit — bash

Andres:ingesoftPracticaGit AndresGutierrez$ git commit -m "set up"
[master (root-commit) 7332da9] set up
 75 files changed, 1131 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Gemfile
 create mode 100644 Gemfile.lock
 create mode 100644 README.md
 create mode 100644 Rakefile
 create mode 100644 app/assets/config/manifest.js
 create mode 100644 app/assets/images/.keep
 create mode 100644 app/assets/javascripts/application.js
 create mode 100644 app/assets/javascripts/cable.js
 create mode 100644 app/assets/javascripts/channels/.keep
 create mode 100644 app/assets/stylesheets/application.css
 create mode 100644 app/channels/application_cable/channel.rb
 create mode 100644 app/channels/application_cable/connection.rb
 create mode 100644 app/controllers/application_controller.rb
 create mode 100644 app/controllers/concerns/.keep
 create mode 100644 app/helpers/application_helper.rb
 create mode 100644 app/jobs/application_job.rb
 create mode 100644 app/mailers/application_mailer.rb
 create mode 100644 app/models/application_record.rb
 create mode 100644 app/models/concerns/.keep
 create mode 100644 app/views/layouts/application.html.erb
 create mode 100644 app/views/layouts/mailers.html.erb

nombreDelProyectoPracticaGit — bash — 110x30
Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ git push -u origin master
Username for 'https://github.com': lpceronm
Password for 'https://lpceronm@github.com':
Counting objects: 85, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (70/70), done.
Writing objects: 100% (85/85), 20.19 KiB | 0 bytes/s, done.
Total 85 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/nombreDelProyectoUnals2/practicaGit.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

```

This screenshot shows a GitHub repository page for 'practicaGitIngeSoft/practicaGit'. The repository has 1 commit, 1 branch, 0 releases, and 1 contributor. The files listed are: app, bin, config, db, lib, log, public, test, tmp, and vendor/assets, all of which are set up. On the right, there is a 'Clone with HTTPS' section with the URL <https://github.com/practicaGitIngeSoft/practicaGit>.

6. A continuación los integrantes restantes del grupo va a clonar el repositorio para ella van a la URL del proyecto

```
[Andres:~ AndresGutierrez$ git clone https://github.com/practicaGitIngeSoft/practicaGit.git
Cloning into 'practicaGit'...
remote: Counting objects: 85, done.
remote: Compressing objects: 100% (68/68), done.
remote: Total 85 (delta 2), reused 85 (delta 2), pack-reused 0
Unpacking objects: 100% (85/85), done.
Checking connectivity... done.
Andres:~ AndresGutierrez$ ]
```

nombreDelProyectoUnals2 / practicaGit

No description, website, or topics provided.

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

1 commit 1 branch 0 releases 0 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

	Laura Ceron Martinez Set up	Latest commit ec451fa 10 minutes ago
app	Set up	10 minutes ago
bin	Set up	10 minutes ago
config	Set up	10 minutes ago
db	Set up	10 minutes ago
lib	Set up	10 minutes ago
log	Set up	10 minutes ago
public	Set up	10 minutes ago
test	Set up	10 minutes ago
tmp	Set up	10 minutes ago

Create new file Upload files Find file Clone or download

Clone with HTTPS Use SSH
Use Git or checkout with SVN using the web URL.
<https://github.com/nombreDelProyectoUn>

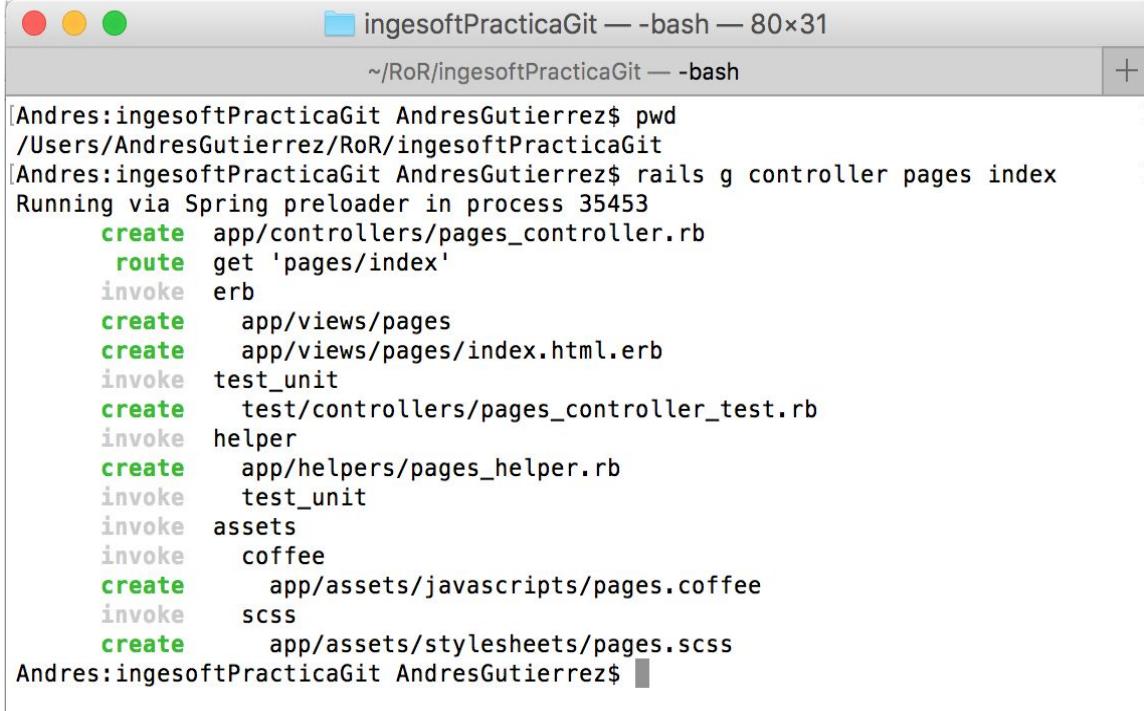
Open in Desktop Download ZIP 15 minutes ago

git y buscan el url para clonar.

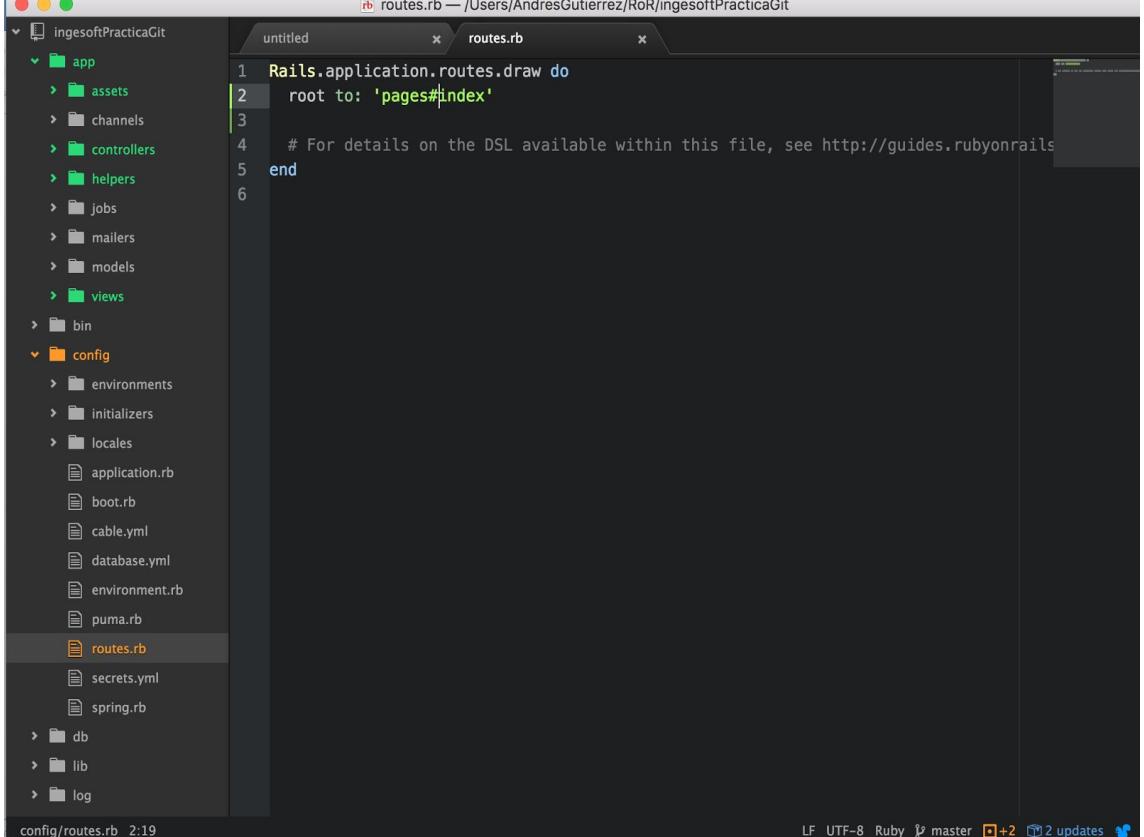
7. Luego en la consola ejecutan el comando **git clone URL** para clonar el repositorio.

En este momento todos los integrantes del grupo ya debieron haber clonado su repositorio y el proyecto debería estar corriendo en cada computador de los integrantes compruébelo con el comando **rails s**.

Ahora nos vamos a dividir en grupos de 2



```
[Andres:ingesoftPracticaGit AndresGutierrez$ pwd  
/Users/AndresGutierrez/RoR/ingesoftPracticaGit  
[Andres:ingesoftPracticaGit AndresGutierrez$ rails g controller pages index  
Running via Spring preloader in process 35453  
  create app/controllers/pages_controller.rb  
    route get 'pages/index'  
  invoke erb  
  create app/views/pages  
  create app/views/pages/index.html.erb  
  invoke test_unit  
  create test/controllers/pages_controller_test.rb  
  invoke helper  
  create app/helpers/pages_helper.rb  
  invoke test_unit  
  invoke assets  
  invoke coffee  
  create app/assets/javascripts/pages.coffee  
  invoke scss  
  create app/assets/stylesheets/pages.scss  
Andres:ingesoftPracticaGit AndresGutierrez$
```



```
routes.rb — /Users/AndresGutierrez/RoR/ingesoftPracticaGit  
untitled routes.rb  
1 Rails.application.routes.draw do  
2   root to: 'pages#index'  
3  
4   # For details on the DSL available within this file, see http://guides.rubyonrails.org  
5 end  
6
```

The sidebar shows the project structure:

- app
 - assets
 - channels
 - controllers
 - helpers
 - jobs
 - mailers
 - models
 - views
- bin
- config
 - environments
 - initializers
 - locales
 - application.rb
 - boot.rb
 - cable.yml
 - database.yml
 - environment.rb
 - puma.rb
 - routes.rb**
 - secrets.yml
 - spring.rb
- db
- lib
- log

el cual cada uno va trabajar una funcionalidad de la aplicación.

1. El grupo 1 va crear un nuevo controlador llamado pages con la acción index y va modificar el archivo **routes.rb** en la carpeta config para que este sea el root de la aplicación.

2. Luego vamos a hacer el commit.

¿Qué es git fetch?

Este comando importa todos los commits desde el repositorio remoto dentro de tu repositorio local, el resultado de los commits son almacenados como ramas remotas en vez de ramas locales. Esto le da a uno la oportunidad de revisar los cambios antes de integrar ellos dentro de la copia de tu proyecto.

- **git fetch <remote>**: descarga todas las ramas y commits y archivos



Welcome to our project

```
ingesoftPracticaGit — bash — 80x31
~/RoR/ingesoftPracticaGit — bash
[Andres:ingesoftPracticaGit AndresGutierrez$ git add .
[Andres:ingesoftPracticaGit AndresGutierrez$ 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:   app/assets/javascripts/pages.coffee
    new file:   app/assets/stylesheets/pages.scss
    new file:   app/controllers/pages_controller.rb
    new file:   app/helpers/pages_helper.rb
    new file:   app/views/pages/index.html.erb
    modified:  config/routes.rb
    new file:   test/controllers/pages_controller_test.rb

Andres:ingesoftPracticaGit AndresGutierrez$ git commit -m "Our first controller"
```

dentro del repositorio.

- **`git fetch <remote> <branch>`**: hace lo mismo que lo anterior pero solo se concentra en la rama especificada.

Si tu quieres combinar los cambios de las ramas remotas a tu repositorio tu deberías utilizar el comando **`git merge`** que discutiremos más tarde.

¿Qué es git pull?

Combina los cambios del repositorio remoto dentro de tu repositorio local, este es un atajo para los comando **`git fetch`** y **`git merge`**.

- **`git pull <remote>`**: descarga la copia de la remota especificada e inmediatamente combina está dentro tu copia local.
- **`git pull --rebase <remote>`**: lo mismo que arriba pero en vez de utilizar **`git merge`** para integrar la rama remota utiliza **`git rebase`**.

¿Qué es git push?

Es la manera de cómo transferir commits desde tu repositorio local a un repositorio remoto. Esta es la contraparte de **`git fetch`** donde **`git fetch`** importa commits a tu repositorio local mientras que **`git push`** exporta commits a las ramas remotas. Esta comando podría sobrescribir cambios por lo que toca usarlo con mucho cuidado.

- **`git push <remote> <branch>`**: envía los cambios a la rama especificado a la remota especificada, git no permite hacer push cuando los resultados no son fast-forward merge en el repositorio de destino.
- **`git push <remote> --force`**: lo mismo que arriba pero obliga a git a hacer el push aun cuando el repositorio no sea fast-forward merge.
- **`git push <remote> --all`**: envía todas las ramas locales al remoto especificado.
- **`git push <remote> --tags`**: envía los tags locales al repositorio remoto.

RECOMENDACIÓN

Siempre utiliza **`git fetch`** y luego de que hayas comprobado los cambios **`git merge`** para tener actualizando tu repositorio ya que **`git pull`** puede presentar conflictos si tu repositorio local y remoto estuvieron trabajando en el mismo archivo.

Después de esta pequeña introducción te has dado cuenta que el commit que acabaste de hacer se encuentra solo en tu repositorio local y no ha sido compartido al repositorio central por lo que los otros desarrolladores están esperando a que tú lo subas para hacer **`git fetch`** or **`git pull`**.

1. Primero siempre comprueba que ramas locales se encuentran al día ya que si no es así el comando **`git push <origin> <branch>`** será rechazado entonces como buena práctica siempre utiliza el comando **`git fetch`** o **`git pull`** primero y luego has el commit a tu repositorio central.
2. El grupo 2 descarga el commit con algunas de las estrategias ya mencionadas quiero decir utilizar fetch y merge o pull.

Las 2 posibles estrategias podrían ser así:

- a. Con fetch
 - i. **`git fetch origin`** para comprobar si hay algo nuevo en el repositorio.
 - ii. **`git log master..origin/master`** para comprobar que ahí de nuevo.

- iii. ***git checkout master*** para mover el cursor a la rama master.
- iv. ***git merge origin/master***.

b. Con push

- i. ***git pull origin***

Push a remoto del primer grupo

```

set up
[Andres:ingesoftPracticaGit AndresGutierrez$ git fetch origin
[Andres:ingesoftPracticaGit AndresGutierrez$ git push origin master
Counting objects: 20, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (17/17), done.
Writing objects: 100% (20/20), 2.05 KiB | 0 bytes/s, done.
Total 20 (delta 5), reused 0 (delta 0)
To https://github.com/practicaGitIngeSoft/practicaGit.git
    7332da9..833e29b master -> master
Andres:ingesoftPracticaGit AndresGutierrez$ 

```

2. Comprobar el grafico de Github y el log de los repo locales.

Felicidades ya estás trabajando de una forma colaborativa ahora vamos a complicar un poco más las cosas.

¿Qué son branches o ramas?

Una rama representa una línea independiente de desarrollo, se puedes pensar en ellas como un nuevo directorio de trabajo con su propia staging área e historia del proyecto. Los nuevos commits son grabados en la historia del proyecto para la rama actual.

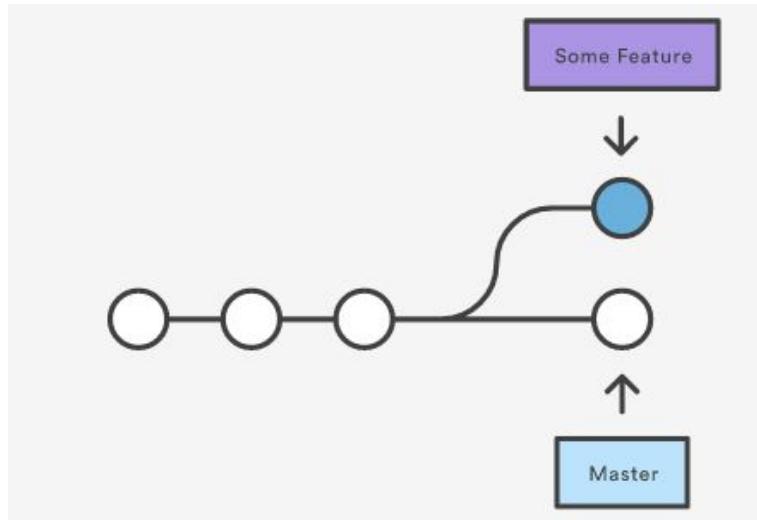
El comando ***git branch*** permite crear, listar, renombrar y borrar ramas, este no permite moverse entre ramas.

- ***git branch***: lista las ramas de tu repositorio

- **git branch <branch>**: crea una nueva rama llamada <branch> pero no te mueve a la rama.
- **git branch -d <branch>**: borrar la rama, de una manera segura ya que git previene de borrarla si esta tiene cambios no combinados.
- **git branch -D <branch>**: fuerza el borrado de la rama.
- **git branch -m <branch>**: renombrar la rama actual por <branch>.

Para cambiar de entre ramas utilizamos el comando git checkout, nosotros utilizamos ese comando para ver viejos commits pero a diferencia de ver viejos commits esta operación no es de solo lectura.

- **git checkout <existing-branch>**: nos cambiamos a la rama especificada esto actualiza el directorio de trabajo para que los archivos asocien con los especificados en el último commit de rama.
- **git checkout -b <new-branch>**: este es un atajo para crear una nueva rama y trasladarnos a ella



Tomada de: <https://www.atlassian.com/git/tutorials/using-branches>

Ya que hemos aprendido los conceptos de las ramas vamos a ponerlo a prueba, los 2 grupos lo deben hacer en paralelo.

1. Grupo 1

- Vamos a crear una nueva rama con el comando **git checkout -b contactUS**.

```

ingesoftPracticaGit — bash — 80x31
~/RoR/ingesoftPracticaGit — bash
[Andres:ingesoftPracticaGit AndresGutierrez$ git checkout -b contactUS
Switched to a new branch 'contactUS'
Andres:ingesoftPracticaGit AndresGutierrez$ ]

```

- Luego vamos a la carpeta app controllers y buscamos pages_controller y agregamos un nuevo método llamado contactUS, luego vamos a la carpeta app views pages y creamos un nuevo archivo llamado contactUS.html.erb.

The screenshot shows a Mac OS X desktop environment. On the left is a terminal window titled 'ingesoftPracticaGit — -bash — 80x31' with the command 'git add .' and its status output. On the right is a code editor window titled 'pages_controller.rb — /Users/AndresGutierrez/RoR/ingesoftPracticaGit'. The code editor displays the 'pages_controller.rb' file, which contains the following code:

```

1 class PagesController < ApplicationController
2   def index
3   end
4   def contactUS
5   end
6 end

```

The file structure on the left side of the code editor shows the project directory structure, including 'app/controllers/pages_controller.rb', 'routes.rb', 'index.html.erb', 'pages_controller.rb', 'contactUS.html.erb', and 'index.html.erb' under 'views/pages'.

- c. Luego hacemos el commit como ya hemos aprendido y lo subimos al repositorio central.

```

Andres:ingesoftPracticaGit AndresGutierrez$ git add .
[Andres:ingesoftPracticaGit AndresGutierrez$ git status
On branch contactUS
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   app/controllers/pages_controller.rb
    new file:   app/views/pages/contactUS.html.erb

[Andres:ingesoftPracticaGit AndresGutierrez$ git commit -m "contact us"
[contactUS fc47362] contact us
 2 files changed, 2 insertions(+)
  create mode 100644 app/views/pages/contactUS.html.erb
[Andres:ingesoftPracticaGit AndresGutierrez$ git push origin
fatal: The current branch contactUS has no upstream branch.
To push the current branch and set the remote as upstream, use

  git push --set-upstream origin contactUS

[Andres:ingesoftPracticaGit AndresGutierrez$ git push origin contactUS
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 635 bytes | 0 bytes/s, done.
Total 7 (delta 3), reused 0 (delta 0)
To https://github.com/practicaGitIngeSoft/practicaGit.git
 * [new branch]      contactUS -> contactUS
Andres:ingesoftPracticaGit AndresGutierrez$ 

```

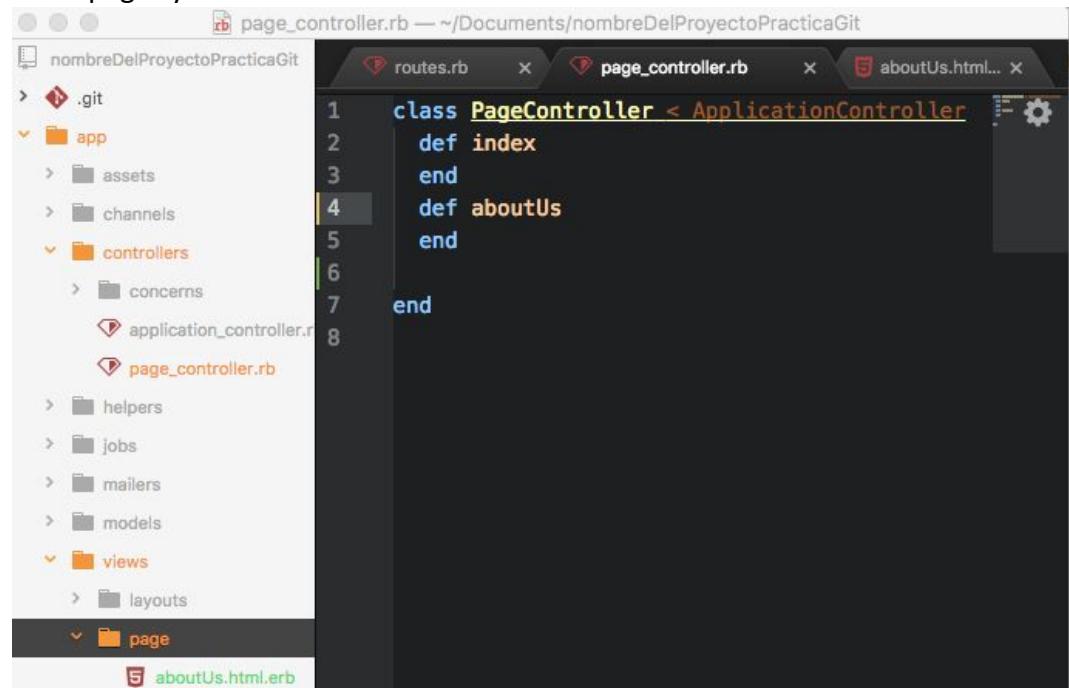
2. Grupo 2

- a. Vamos a crear una nueva rama con el comando **git checkout -b aboutUs**.



```
nombreDelProyectoPracticaGit — -bash — 93x30
~/Documents/nombreDelProyectoPracticaGit — -bash fsevent_watch • 0.0.0:3000 [no...TERM_PROGRAM=Apple_Terminal +]
Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ git checkout -b aboutUs
Switched to a new branch 'aboutUs'
Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$
```

- b. Luego vamos a la carpeta app controllers y buscamos pages_controller y agregamos un nuevo método llamado aboutUS, luego vamos a la carpeta app views pages y creamos un nuevo archivo llamado aboutUS.html.erb



```
routes.rb x page_controller.rb x aboutUs.html... x
page_controller.rb — ~/Documents/nombreDelProyectoPracticaGit
routes.rb x page_controller.rb x aboutUs.html... x
1 class PageController < ApplicationController
2   def index
3   end
4   def aboutUs
5   end
6
7 end
```

The sidebar shows the project structure:

- .git
- app
 - assets
 - channels
 - controllers
 - concerns
 - application_controller.rb
 - page_controller.rb
 - helpers
 - jobs
 - mailers
 - models
 - views
 - layouts
 - page
 - aboutUs.html.erb

- c. Luego hacemos el commit como ya hemos aprendido y lo subimos al

```

nombreDelProyectoPracticaGit — -bash — 93x30
~/Documents/nombreDelProyectoPracticaGit — -bash fsevent_watch • 0.0.0.0:3000 [no...TERM_PROGRAM=Apple_Terminal +]
[Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ git add .
[Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ git commit -m "about us"
[aboutUs 32cf6de] about us
  Committer: Laura Ceron Martinez <lpceron@Lauras-MacBook-Air.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

  git config --global user.name "Your Name"
  git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

  git commit --amend --reset-author

3 files changed, 2 insertions(+), 2 deletions(-)
create mode 100644 app/views/page/aboutUs.html.erb
delete mode 100644 app/views/page/contactUs.html.erb

```



```

nombreDelProyectoPracticaGit — -bash — 93x30
~/Documents/nombreDelProyectoPracticaGit — -bash fsevent_watch • 0.0.0.0:3000 [no...TERM_PROGRAM=Apple_Terminal +]
[Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ git push origin aboutUs
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 616 bytes | 0 bytes/s, done.
Total 7 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/nombreDelProyectoUnals2/practicaGit.git
 * [new branch]      aboutUs -> aboutUs
Lauras-MacBook-Air:nombreDelProyectoPracticaGit lpceron$ 

```

repositorio central.

Ya hemos creado nuestras 2 ramas, ahora cada grupo va trabajar un poco más en ella y luego vamos a fusionar lo que hemos hecho en la rama master ya que en esta es donde siempre se tiene que encontrar la versión estable del código.

1. Grupo 1
 - a. Van al archivo config route.rb y crean la route para el método que acaban de crear.

The screenshot shows a terminal window with the following details:

- File Structure:** On the left, a tree view of the application's directory structure. It includes: jobs, mailers, models, views (with layouts and pages), bin, config (environments, initializers, locales, application.rb, boot.rb, cable.yml, database.yml, environment.rb, puma.rb, routes.rb), db, lib, log, public, and test.
- routes.rb Content:** The routes file contains the following code:

```
1 Rails.application.routes.draw do
2   root to: 'pages#index'
3   get 'contactUS', to: 'pages#contactUS'
4
5   # For details on the DSL available within this file, see http://guides.rubyonrails.org
6 end
```
- Bottom Status Bar:** Shows LF, UTF-8, Ruby, contactUS, +1, 2 updates, and a GitHub icon.

- b. Luego modifican el html.erb que crearon con algún mensaje.

The screenshot shows a terminal window with the following details:

- File Structure:** On the left, a tree view of the application's directory structure, identical to the previous screenshot.
- contactUS.html.erb Content:** The file contains the following HTML code:

```
<h1>Este es mi telefono de contacto:</h1>
<h3>3108003068</h3>
```
- Bottom Status Bar:** Shows LF, UTF-8, HTML (Ruby - ERB), contactUS, +2, 2 updates, and a GitHub icon.

- c. Luego en la carpeta app views pages index.htm.erb crean un link para esa nueva página.

```

index.html.erb — /Users/AndresGutierrez/RoR/ingesoftPracticaGit
untitled x routes.rb x contactUS.html.erb x Index.html.erb x pages_controller.rb x
> jobs
> mailers
> models
views
> layouts
pages
> contactUS.html.e
index.html.erb
> bin
config
> environments
initializers
locales
application.rb
boot.rb
cable.yml
database.yml
environment.rb
puma.rb
routes.rb
secrets.yml
spring.rb
db
lib
log
public
test
app/views/pages/index.html.erb 2:47
LF UTF-8 HTML (Ruby - ERB) ⌂ contactUS ⌂ +1 ⌂ 2 updates ⌂

```

- d. Hacer el commit y luego lo subirlo al repositorio central.

```

ingesoftPracticaGit — -bash — 80x31
~/RoR/ingesoftPracticaGit — -bash
[Andres:ingesoftPracticaGit AndresGutierrez$ git add .
[Andres:ingesoftPracticaGit AndresGutierrez$ git status
On branch contactUS
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

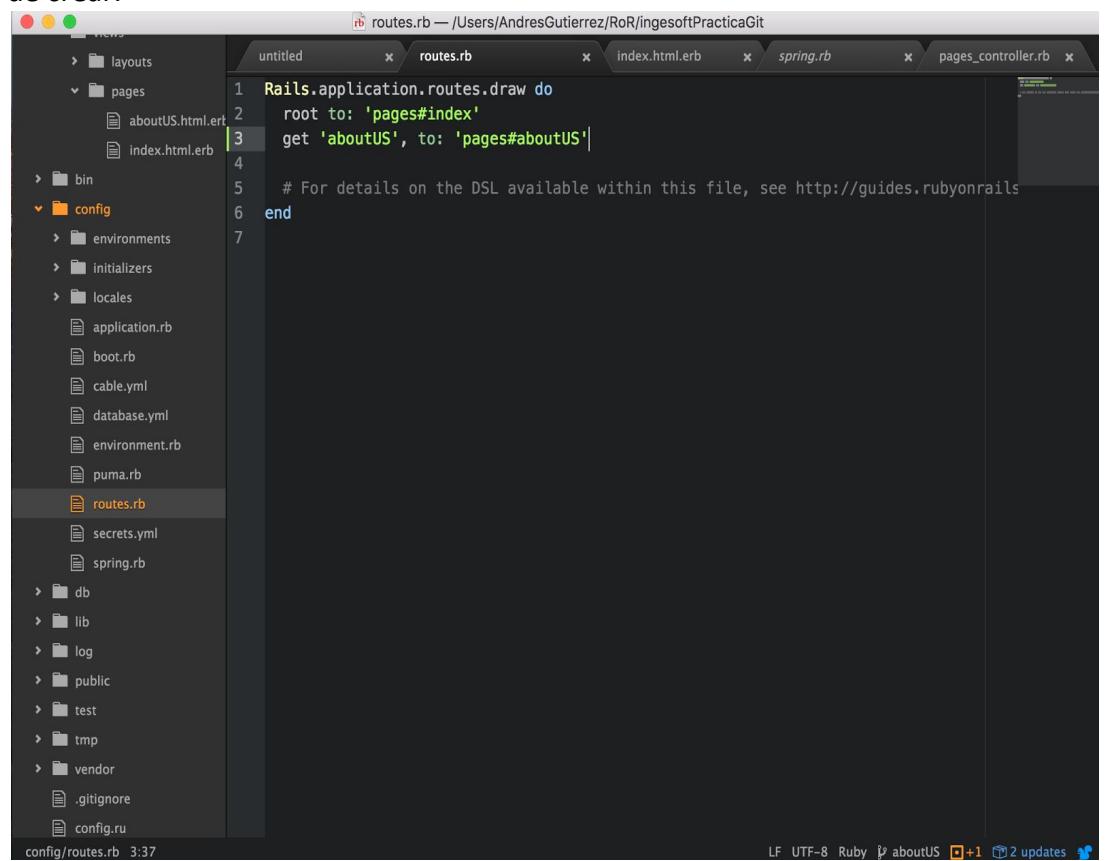
modified:   app/views/pages/contactUS.html.erb
modified:   app/views/pages/index.html.erb
modified:   config/routes.rb

[Andres:ingesoftPracticaGit AndresGutierrez$ git commit -m "route to contact us"
[contactUS bac2e01] route to contact us
 3 files changed, 4 insertions(+)
[Andres:ingesoftPracticaGit AndresGutierrez$ git push origin contactUS
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 794 bytes | 0 bytes/s, done.
Total 9 (delta 4), reused 0 (delta 0)
To https://github.com/practicaGitInGeSoft/practicaGit.git
  fc47362..bac2e01  contactUS -> contactUS
Andres:ingesoftPracticaGit AndresGutierrez$ ]

```

2. Grupo 2

- Vamos al archivo config route.rb y crean la ruta para el método que se acaba de crear.



The screenshot shows a Mac OS X desktop with a terminal window open. The terminal window has several tabs: 'untitled', 'routes.rb', 'index.html.erb', 'spring.rb', and 'pages_controller.rb'. The 'routes.rb' tab is active and displays the following code:

```
Rails.application.routes.draw do
  root to: 'pages#index'
  get 'aboutUS', to: 'pages#aboutUS'

# For details on the DSL available within this file, see http://guides.rubyonrails.org/routing.html
end
```

The file path 'config/routes.rb' is visible at the bottom of the terminal window. The file has 3 lines of code and was last modified at 3:37. The status bar at the bottom right shows 'LF UTF-8 Ruby' and other system information.

- Luego modifican el html.erb que crearon con algún mensaje.

The screenshot shows a Mac OS X desktop environment. In the foreground, a terminal window titled "aboutUS.html.erb — /Users/AndresGutierrez/RoR/ingesoftPracticaGit" is open. The command line shows the path: "app/views/pages/aboutUS.html.erb". The content of the file is:

```
<h1>Andres Rene Gutierrez</h1>
```

In the background, a file browser window titled "ingesoftPracticaGit" is visible. The sidebar shows the project structure:

- app
 - assets
 - channels
 - controllers
 - concerns
 - application_controller.rb
 - pages_controller.rb
 - helpers
 - jobs
 - mailers
 - models
 - views
 - layouts
 - pages
 - aboutUS.html.erb
 - index.html.erb
- bin
- config
 - environments
 - initializers
 - locales
 - application.rb
 - boot.rb
 - cable.yml
 - database.yml
 - environment.rb

- c. Luego en la carpeta app views pages index.htm.erb crean un link para esa nueva página.

The screenshot shows a Mac OS X desktop environment. In the foreground, a terminal window titled "index.html.erb — /Users/AndresGutierrez/RoR/ingesoftPracticaGit" is open. The command line shows the path: "app/views/pages/index.html.erb". The content of the file is:

```
<h1>Welcome to our project</h1>
<p><%=link_to 'acerca de nosotros',aboutUS_path %></p>
```

In the background, a file browser window titled "ingesoftPracticaGit" is visible. The sidebar shows the project structure:

- app
 - assets
 - channels
 - controllers
 - concerns
 - application_controller.rb
 - pages_controller.rb
 - helpers
 - jobs
 - mailers
 - models
 - views
 - layouts
 - pages
 - aboutUS.html.erb
 - index.html.erb
- bin
- config
 - environments
 - initializers
 - locales
 - application.rb
 - boot.rb
 - cable.yml
 - database.yml
 - environment.rb

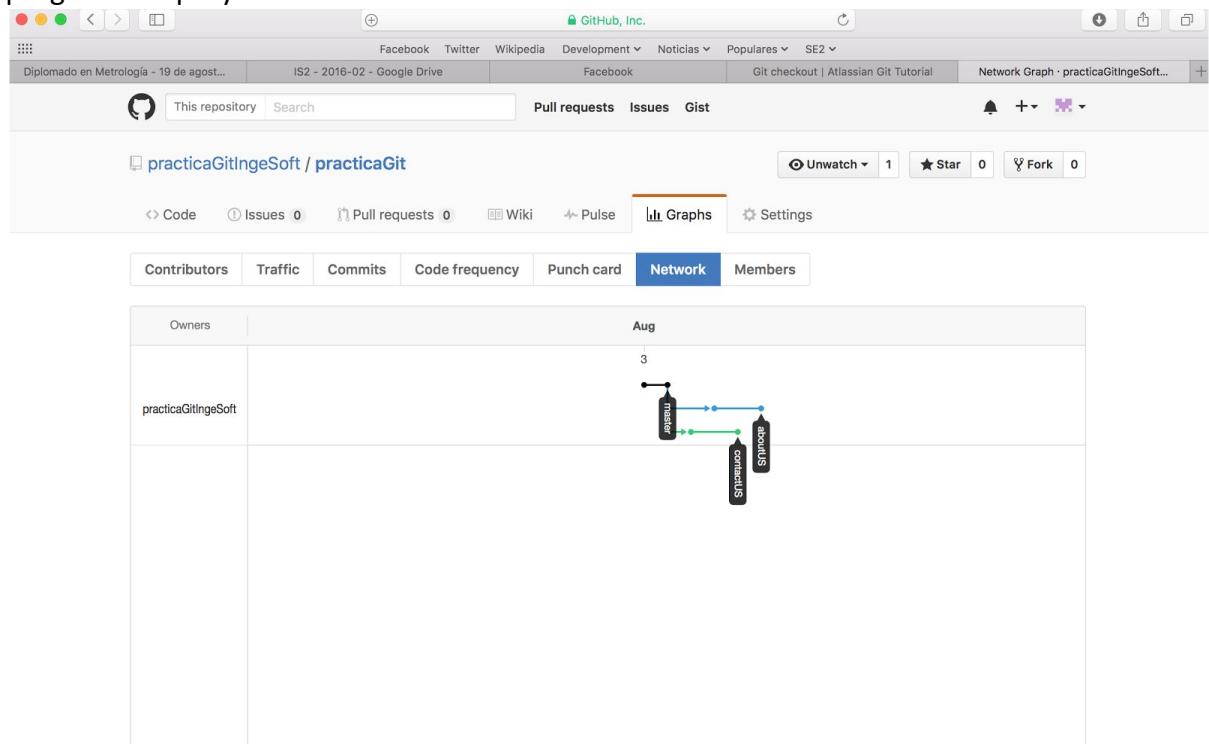
d. Hacer el commit y subirlo al repositorio central.

```
ingesoftPracticaGit — -bash — 80x31
~/RoR/ingesoftPracticaGit — -bash
[Andres:ingesoftPracticaGit AndresGutierrez$ git add .
[Andres:ingesoftPracticaGit AndresGutierrez$ git status
On branch aboutUS
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   app/views/pages/aboutUS.html.erb
    modified:   app/views/pages/index.html.erb
    modified:   config/routes.rb

[Andres:ingesoftPracticaGit AndresGutierrez$ git commit -m "route to aboutUS"
[aboutUS cabc4f8] route to aboutUS
  3 files changed, 3 insertions(+)
[Andres:ingesoftPracticaGit AndresGutierrez$ git push origin aboutUS
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 766 bytes | 0 bytes/s, done.
Total 9 (delta 4), reused 0 (delta 0)
To https://github.com/practicaGitIngeSoft/practicaGit.git
  4a034d1..cabc4f8  aboutUS -> aboutUS
Andres:ingesoftPracticaGit AndresGutierrez$ ]
```

Después de que los 2 grupos hayan terminado consultar el network de github para ver el progreso del proyecto.

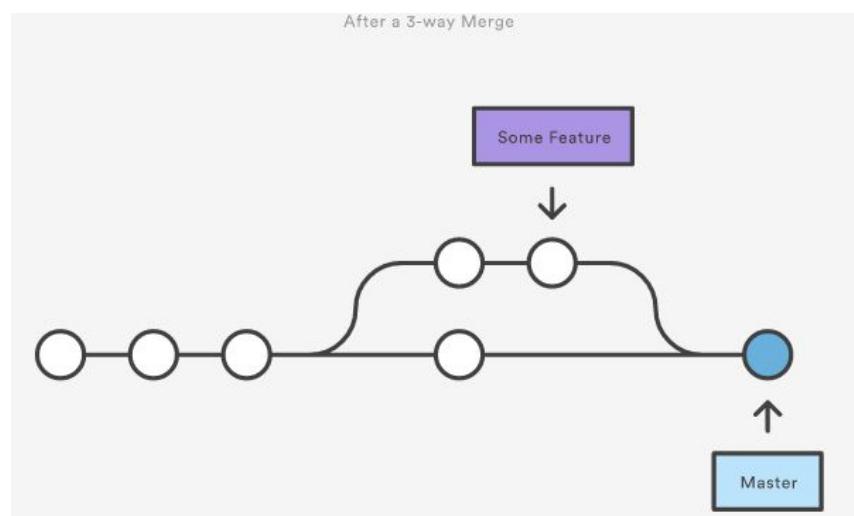


Como has podido ver ya has desarrollado la funcionalidad que tenías a tu cargo ahora llegó el momento de fusionar todo a master y que este sea la versión estable del proyecto.

¿Qué es git merge?

Permite combinar independientes líneas de desarrollo e integrar ellas dentro de una simple rama los comandos del merge trabajan dentro de la rama actual eso quiere decir que si esta en master y ejecuta git merge <branch> la rama <branch> será fusionada en master.

- **git merge <branch>**: fusiona la rama especificada dentro de la actual rama.



¿Qué es git rebase?

Es mover una rama a un nuevo commit base.

- **git rebase <base>**: rebase la actual rama dentro de <base>, el cual puede ser cualquier tipo de referencia de commit(id,nombre de la rama,o tag).

¿Qué son conflictos?

Si 1 o más archivos han cambiado en tu rama actual y la que tu estas intentando fusionar, git no será capaz de descubrir cual versión usar, cuando esta situación ocurre el commit del merge es detenido así tu puedes resolver los conflictos manualmente.

Git status muestra cuales archivos necesitan ser arreglados antes de continuar con el merge, después de que tú has resuelto los conflictos tú tienes que notificarle a git que ellos fueron resueltos utilizando el comando **git add** para añadir esos archivos, después debes usar el comando **git commit** para continuar con el merge.

Para combinar nuestras ramas primero nos tenemos que ubicar en master entonces ejecutamos el comando **git checkout master**

Empezaremos con el grupo 1, lo primero que tienen que hacer es ejecutar el comando **git merge contactUS** para fusionar su rama dentro de master y luego ejecutar **git push origin master** para subir los nuevos cambios al repositorio central esto será una tarea sencilla ya que master se encuentra en el mismo punto que cuando empezó el desarrollo de la rama, por ultimo consultamos el network de nuestro proyecto.

```
[Andres:ingesoftPracticaGit AndresGutierrez$ git merge contactUS
Updating 833e29b..bac2e01
Fast-forward
 app/controllers/pages_controller.rb | 2 ++
 app/views/pages/contactUS.html.erb | 2 ++
 app/views/pages/index.html.erb    | 1 +
 config/routes.rb                  | 1 +
 4 files changed, 6 insertions(+)
 create mode 100644 app/views/pages/contactUS.html.erb
[Andres:ingesoftPracticaGit AndresGutierrez$ git log
commit bac2e0142787a9cfabf007f0a41980eeeea4b6d
Author: Andres Gutierrez <agutierrezt@unal.edu.co>
Date:   Wed Aug 3 14:55:21 2016 -0500

    route to contact us

commit fc473626595049c4e57758996f7fabeeaf838d50
Author: Andres Gutierrez <agutierrezt@unal.edu.co>
Date:   Wed Aug 3 14:29:31 2016 -0500

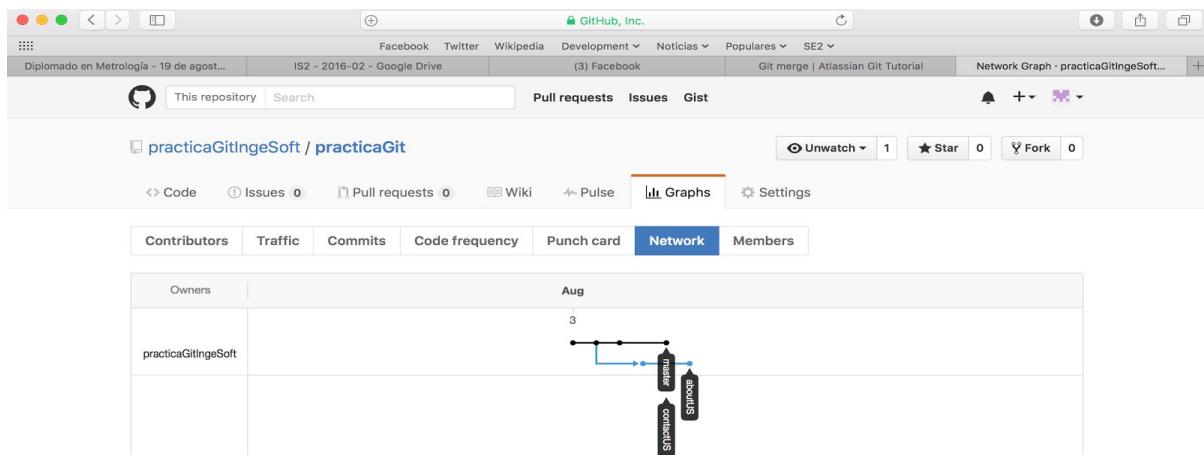
    contact us

commit 833e29b836b7b38d2b2ac5b8ce298d5f5b3c73e8
Author: Andres Gutierrez <agutierrezt@unal.edu.co>
Date:   Wed Aug 3 12:21:28 2016 -0500

    Our first controller

commit 7332da90669b67a926ca480a5d81505233cea20a
Author: Andres Gutierrez <agutierrezt@unal.edu.co>
```

```
[Andres:ingesoftPracticaGit AndresGutierrez$ git push origin master
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/practicaGitIngeSoft/practicaGit.git
 833e29b..bac2e01  master -> master
Andres:ingesoftPracticaGit AndresGutierrez$ ]
```



A continuación vamos a fusionar la rama restante dentro de master, esta tarea será realizada por el grupo 2.

1. ***git checkout master***
2. ***git fetch origin*** para comprobar los nuevos cambios que ha habido en el repositorio.
3. Luego ejecute el comando ***git log master..origin/master*** para comprobar cómo ha cambiado el repositorio.
4. Si se siente agosto con los cambios ejecute el comando ***git merge origin/master***.
5. Hasta el momento lo único que hemos hecho es traer los cambios actuales a nuestro repositorio ahora vamos fusionar nuestra rama para ello vamos a ejecutar el comando ***git merge aboutUS***.

```
ingesoftPracticaGit — bash — 80x31
~/RoR/ingesoftPracticaGit — bash
[Andres:ingesoftPracticaGit AndresGutierrez$ git merge aboutUS
Auto-merging config/routes.rb
CONFLICT (content): Merge conflict in config/routes.rb
Auto-merging app/views/pages/index.html.erb
CONFLICT (content): Merge conflict in app/views/pages/index.html.erb
Auto-merging app/controllers/pages_controller.rb
CONFLICT (content): Merge conflict in app/controllers/pages_controller.rb
Automatic merge failed; fix conflicts and then commit the result.
Andres:ingesoftPracticaGit AndresGutierrez$ ]
```

6. Como se puede notar en la imagen hubieron algunos conflictos porque durante el desarrollo ambos grupos estuvieron trabajando sobre el mismo archivo, vamos a ejecutar el comando ***git status*** para conocer qué problemas tenemos.

```
ingesoftPracticaGit — bash — 80x31
~/RoR/ingesoftPracticaGit — bash
[Andres:ingesoftPracticaGit AndresGutierrez$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
You have unmerged paths.
  (fix conflicts and run "git commit")

Changes to be committed:

      new file:   app/views/pages/aboutUS.html.erb

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

        both modified:  app/controllers/pages_controller.rb
        both modified:  app/views/pages/index.html.erb
        both modified:  config/routes.rb
Andres:ingesoftPracticaGit AndresGutierrez$ ]
```

1. Durante el desarrollo ambos equipos tocaron 3 archivos comunes, para solucionarlo abrimos nuestro IDE y nos dirigimos a los 3 archivos y solucionamos los conflictos, en las imágenes siguientes muestran cómo se soluciona el conflicto.

The screenshot shows a terminal window with a file browser on the left and a code editor on the right. The file browser lists files like aboutUS.html.erb, contactUS.html.erb, index.html.erb, routes.rb, and pages_controller.rb. The code editor shows the routes.rb file with the following content:

```
1 Rails.application.routes.draw do
2   root to: 'pages#index'
3   get 'contactUS', to: 'pages#contactUS'
4   get 'aboutUS', to: 'pages#aboutUS'
5   # For details on the DSL available within this file, see http://guides.rubyonrails.org
6 end
```

The line 'get 'aboutUS', to: 'pages#aboutUS'' is highlighted with a yellow background, indicating it is a conflict. The status bar at the bottom right shows 'LF UTF-8 Ruby' and 'master 2 updates'.

The screenshot shows a code editor window with the following details:

- Title Bar:** index.html.erb — /Users/AndresGutierrez/RoR/ingesoftPracticaGit
- File Tree:** On the left, the project structure is shown under "app":
 - assets
 - channels
 - controllers
 - concerns
 - application_controller.rb
 - pages_controller.rb
 - helpers
 - jobs
 - mailers
 - models
 - views
 - layouts
 - pages
 - aboutUS.html.erb
 - contactUS.html.erb
 - index.html.erb
- Code Editor:** The main pane displays the content of index.html.erb:

```
<h1>Welcome to our project</h1>
<p><%=link_to 'contactenos', contactUS_path %></p>
<p><%=link_to 'acerca de nosotros', aboutUS_path %></p>
```
- Status Bar:** LF UTF-8 HTML (Ruby – ERB) master 2 updates

The screenshot shows a code editor window with the following details:

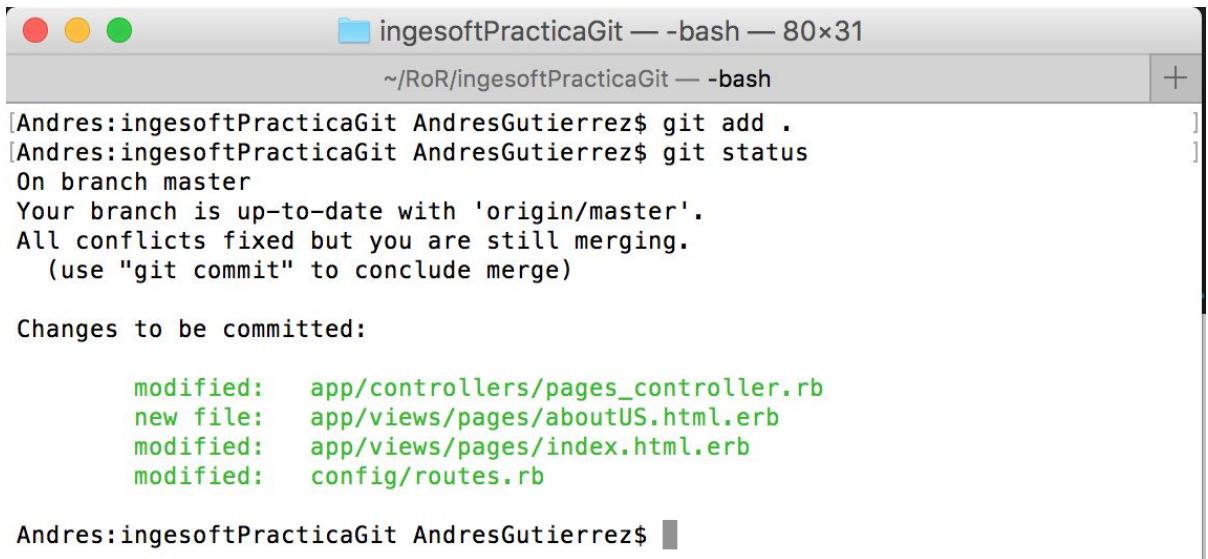
- Title Bar:** pages_controller.rb — /Users/AndresGutierrez/RoR/ingesoftPracticaGit
- File Tree:** On the left, the project structure is shown under "app":
 - assets
 - channels
 - controllers
 - concerns
 - application_controller.rb
 - pages_controller.rb
 - helpers
 - jobs
 - mailers
 - models
 - views
 - layouts
 - pages
 - aboutUS.html.erb
 - contactUS.html.erb
 - index.html.erb
- Code Editor:** The main pane displays the content of pages_controller.rb:

```
class PagesController < ApplicationController
  def index
  end

  def contactUS
  end

  def aboutUS
  end
end
```
- Status Bar:** LF UTF-8 Ruby master 2 updates

2. Antes de añadir los archivos solucionados con **git add**. ejecutamos el comando **rails s** para comprobar que todo esté funcionando como esperamos.
3. **git commit** para continuar con el merge y luego hacemos push a nuestro repositorio central.



```
[Andres:ingesoftPracticaGit AndresGutierrez$ git add .
[Andres:ingesoftPracticaGit AndresGutierrez$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

  modified:   app/controllers/pages_controller.rb
  new file:   app/views/pages/aboutUS.html.erb
  modified:   app/views/pages/index.html.erb
  modified:   config/routes.rb

Andres:ingesoftPracticaGit AndresGutierrez$ ]]
```

ingesoftPracticaGit — nano ▾ git commit — 80x31

~ /RoR/ingesoftPracticaGit — nano ▾ git commit

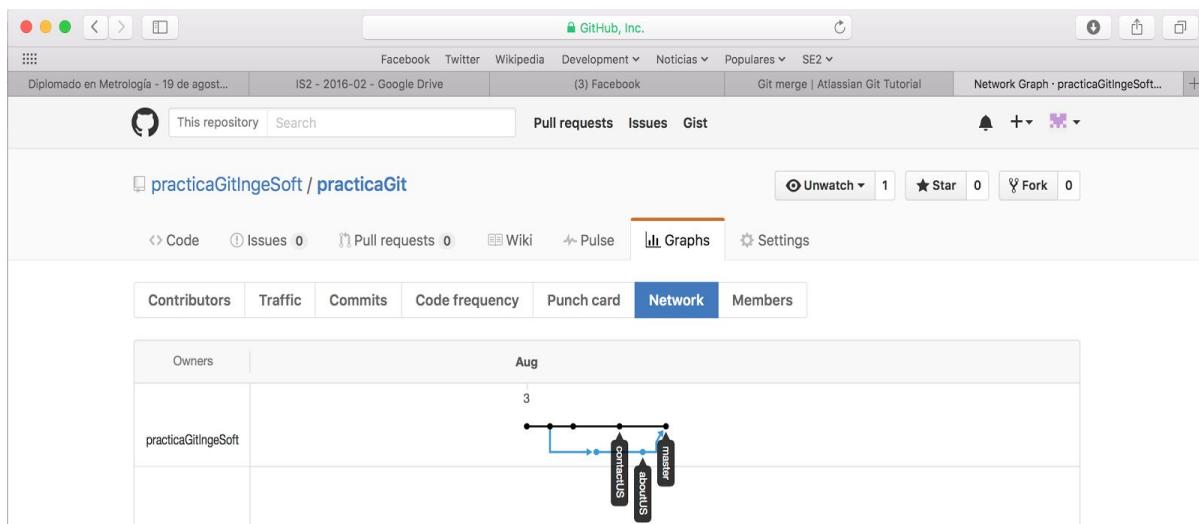
GNU nano 2.0.6 File: ...RoR/ingesoftPracticaGit/.git/COMMIT_EDITMSG

```
Merge branch 'aboutUS'

# Conflicts:
#       app/controllers/pages_controller.rb
#       app/views/pages/index.html.erb
#       config/routes.rb
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is up-to-date with 'origin/master'.
#
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   app/controllers/pages_controller.rb
#       new file:   app/views/pages/aboutUS.html.erb
#       modified:   app/views/pages/index.html.erb
#       modified:   config/routes.rb

[ Read 26 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```



¿Qué es git tag?

Un tag es un nombre simbólico dado a una versión, ellos siempre apuntan al mismo objeto y no cambian.

- **git tag -a <name> <commitID> -m “message”**: crear un tag para un commit específico.
- **git tag**: Para ver todos los tags creados.

Por último el grupo 1 va crear un tag sobre el ultimo commit que se encuentra en master.

1. **git pull origin** para traer los últimos cambios.
2. **git log --oneline** para buscar el commit id del ultimo commit.

```
Andres:ingesoftPracticaGit AndresGutierrez$ git log --oneline
d1abeba Merge branch 'aboutUS'
cabc4f8 route to aboutUS
bac2e01 route to contact us
4a034d1 about us
fc47362 contact us
833e29b Our first controller
7332da9 set up
Andres:ingesoftPracticaGit AndresGutierrez$
```

3. **git tag -a v1.0 <commitID> -m “We have finished the second part”**

```
Andres:ingesoftPracticaGit AndresGutierrez$ git tag -a v1.0 d1abeba -m "We have finished the second part"
Andres:ingesoftPracticaGit AndresGutierrez$ git tag v1.0
Andres:ingesoftPracticaGit AndresGutierrez$
```

4. Por ultimo **git push origin --tags**.

```
Andres:ingesoftPracticaGit AndresGutierrez$ git push origin --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 180 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To https://github.com/practicaGitIngeSoft/practicaGit.git
 * [new tag]           v1.0 -> v1.0
Andres:ingesoftPracticaGit AndresGutierrez$
```

Felicidades ya sabes cómo solucionar conflictos, hacer merge y crear tags has terminado la parte 2 del taller.

PARTE III

Todos los comandos que hemos utilizado nos son fácil de recordar además son un poco problemáticos ya que toca ejecutarlos desde consola y dentro del directorio del proyecto, lo bueno del caso es que existe muchas herramientas que nos ayudan a facilitar esta tarea, la herramienta recomendada para todo el manejo del repositorio es [sourcetree](#), como se indicó en las reglas del curso todos los equipos deben utilizar el workflow de git, sourcetree viene una con una característica para inicializar este workflow de una manera sencilla.

¿Qué es git workflow?

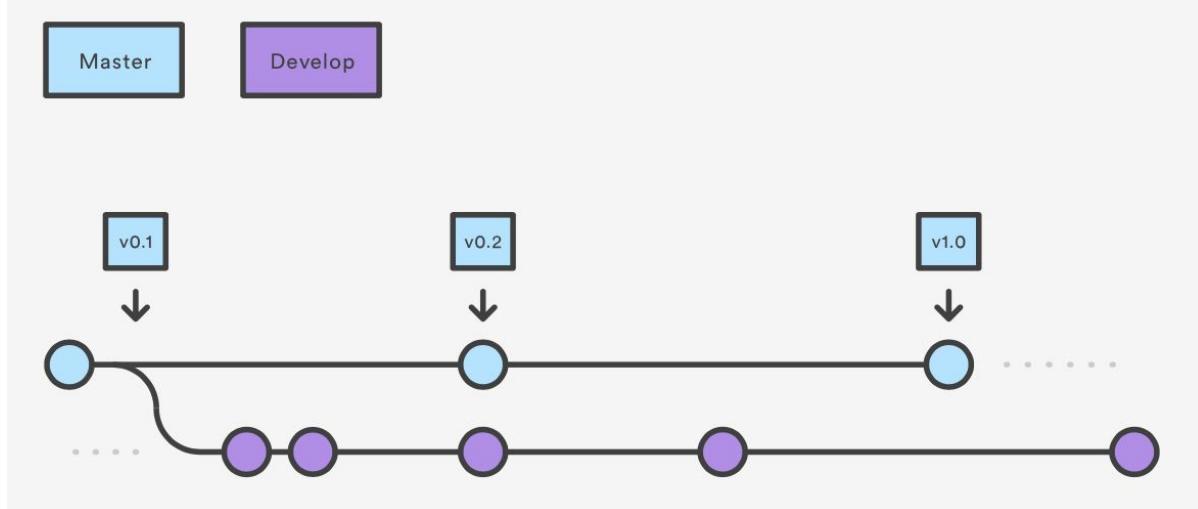
Este define un estricto modelo de ramas diseñado alrededor de los releases del proyecto, esto proporciona un robusto framework para manejar proyectos largos.

Este workflow no añade nuevos conceptos o comandos, en vez este asigna roles muy específicos a diferentes ramas y define como y cuando ellas deberían interactuar.

¿Cómo funciona?

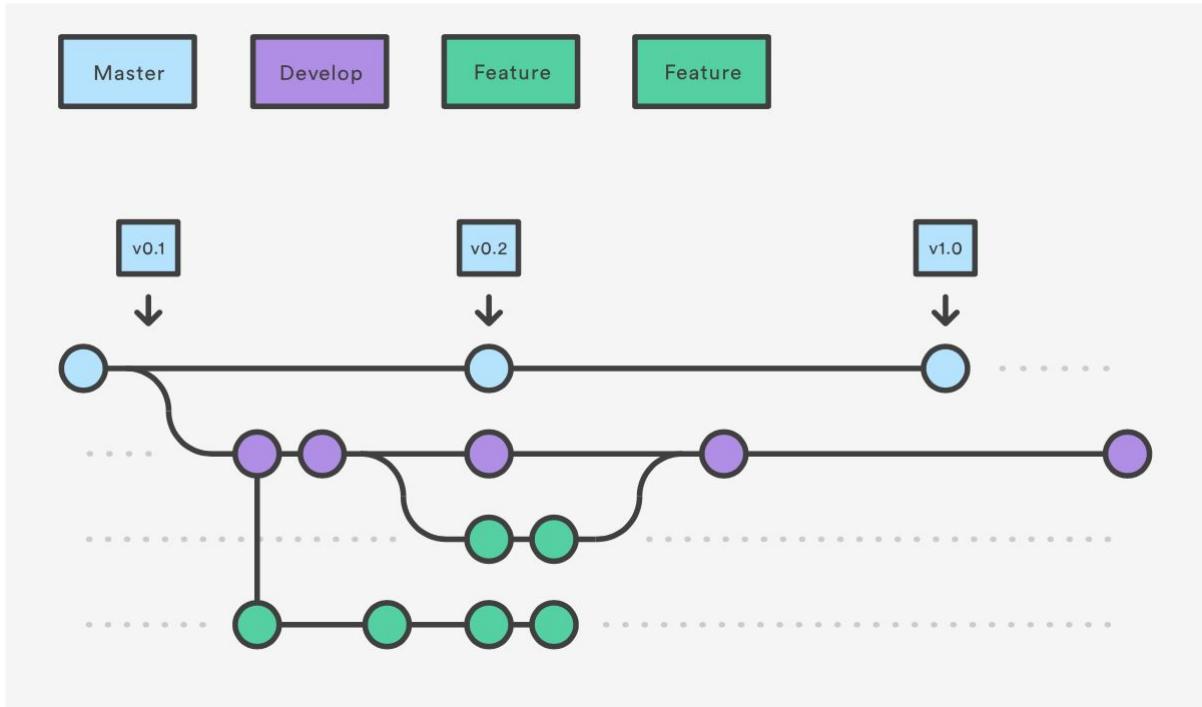
Ramas Históricas

En vez de la rama master, este workflow usa 2 ramas para registrar la historia del proyecto. La rama master almacena la historia de los releases oficiales y la rama develop sirve como una rama de integración para las features, es también conveniente hacer tag a todos los commits in master.



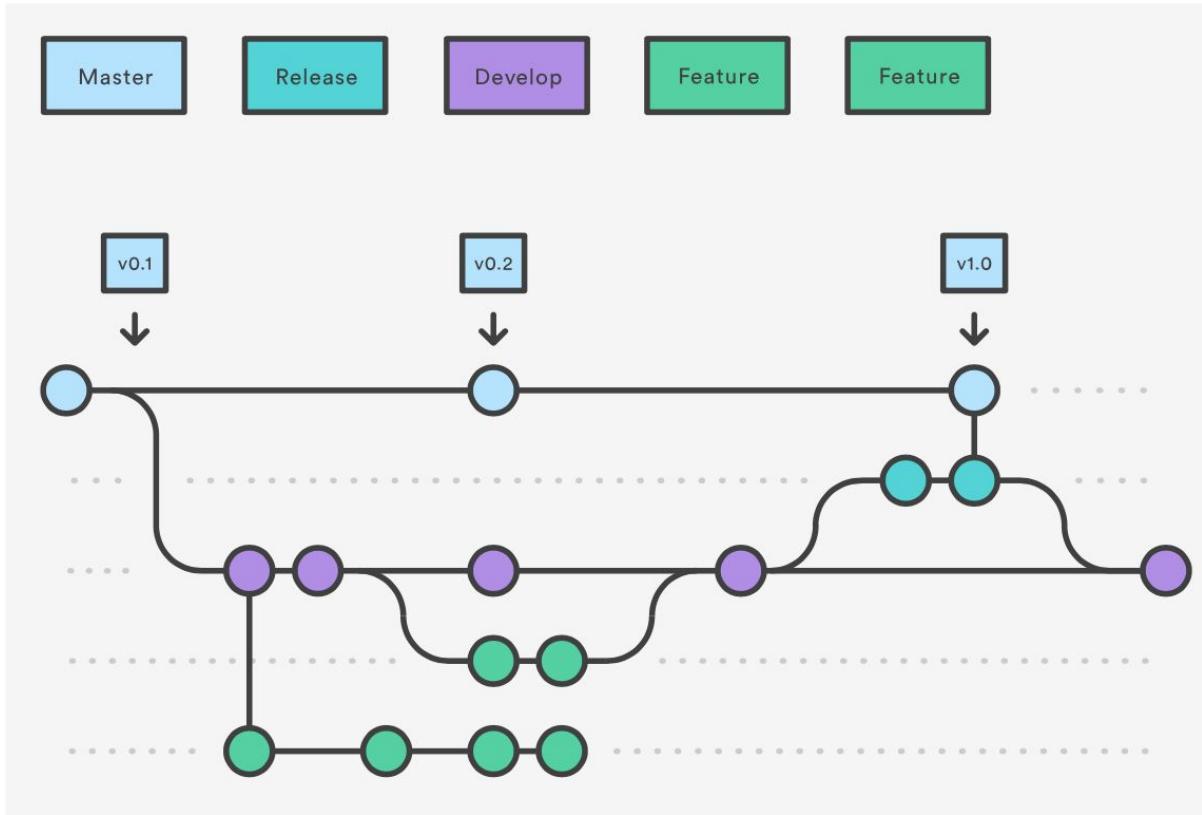
Ramas Features

Cada nueva feature debería residir en su propia rama, pero en vez de usar a master, la rama feature usa a la rama develop como su parente. Cuando una rama es finalizada esta hace merge dentro de develop. Features nunca deberían interactuar directamente con master.



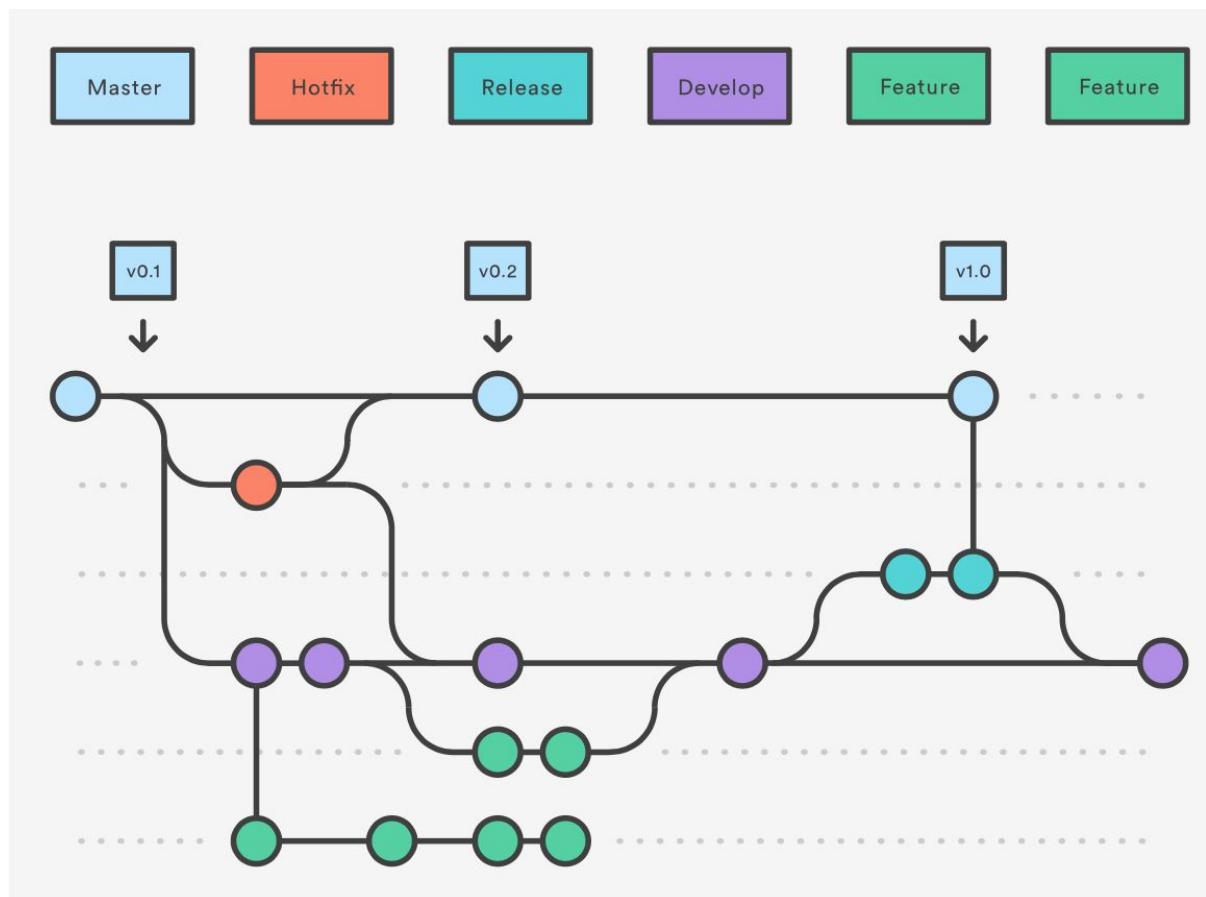
Ramas Releases

Una vez el desarrollo ha acumulado suficiente características para un release o una fecha predeterminada de release se acerca se crea una nueva rama release desde develop. Creando esta nueva rama inicia el nuevo ciclo de release, así que nuevas características no pueden ser agregadas después de este punto, solamente, arreglos de bugs, generación de documentación y otras tareas relacionadas con el release deberían ir en esta rama. Una vez el release está listo, este consigue hacer merge dentro de master y taggeado con un numero secuencial de versión. Ademas este también debería hacer merge devuelta a develop, cual podría haber progresado desde que el release fue iniciado.

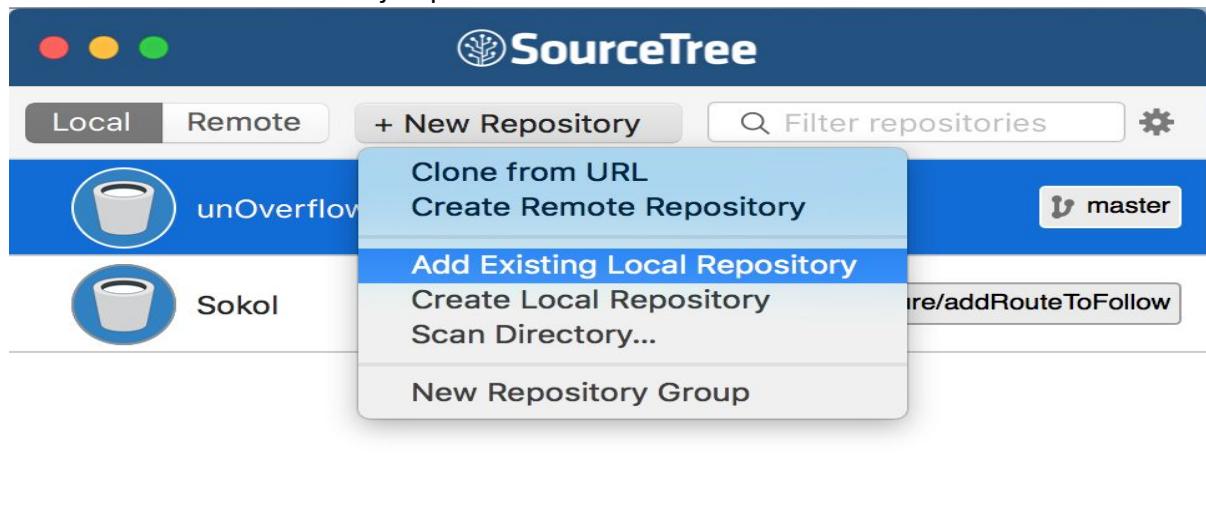


Ramas de mantenimiento

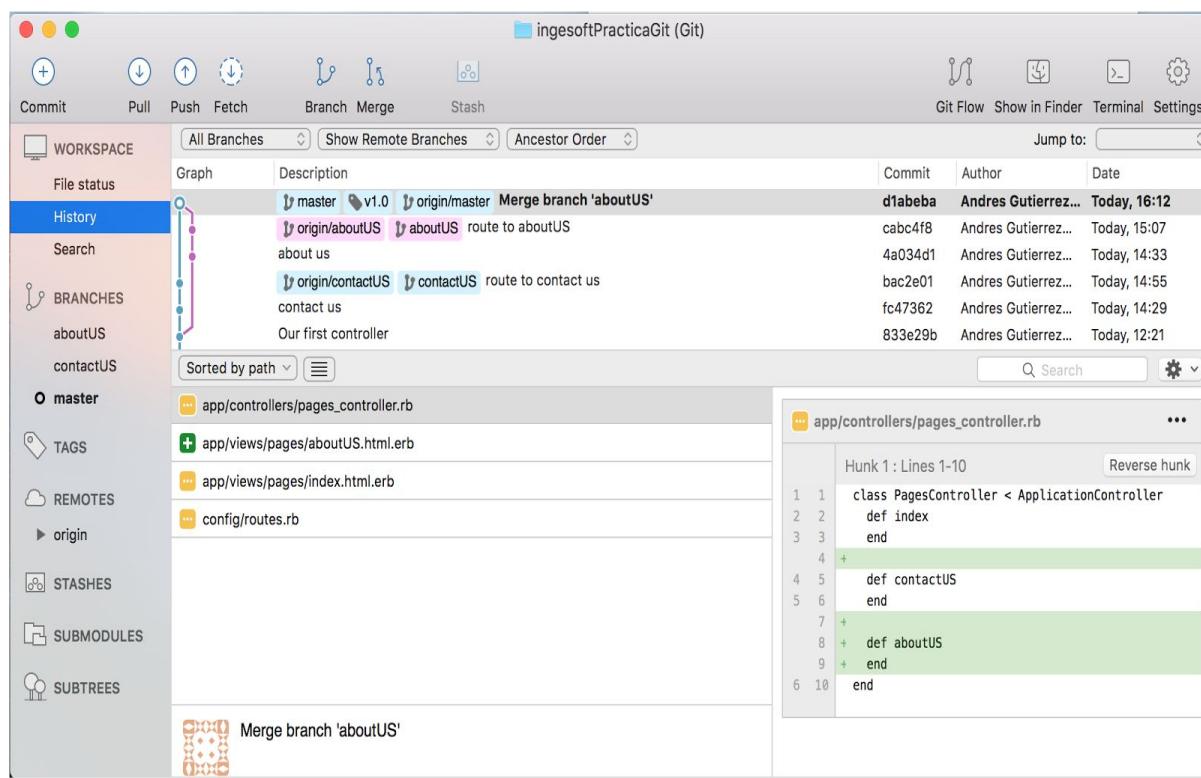
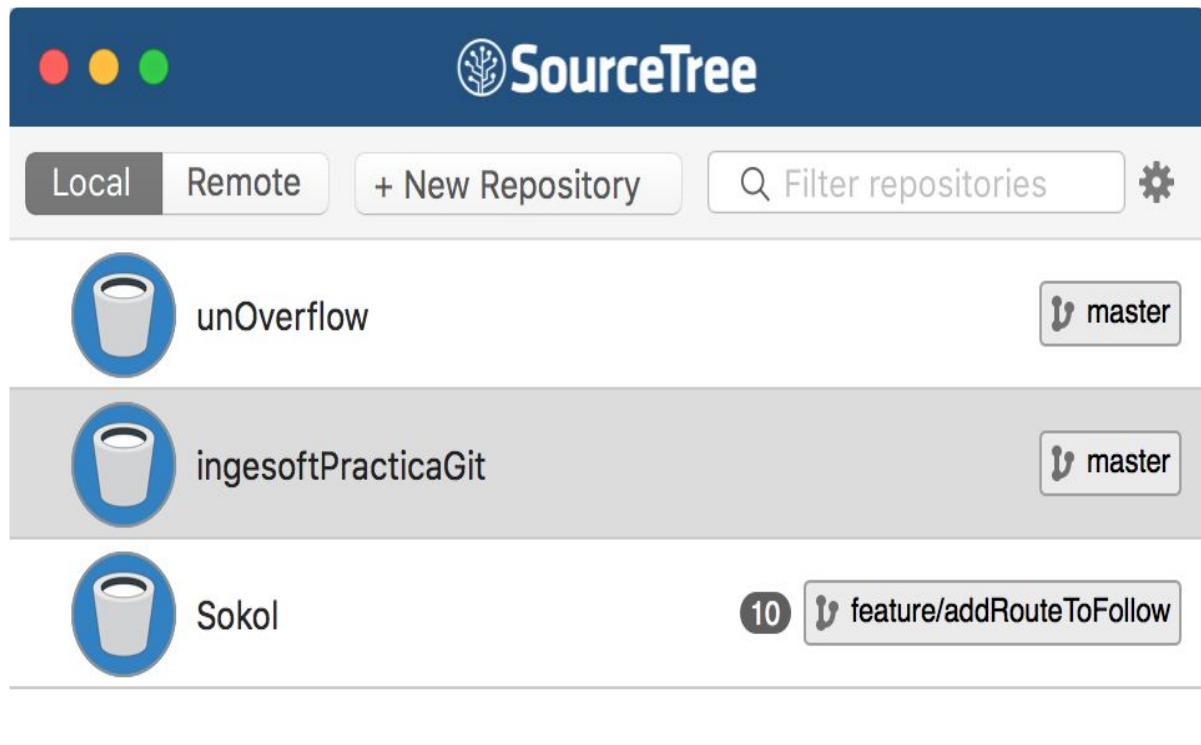
Ramas de mantenimiento o “hotfix” son usadas para rápidamente arreglar releases de producción, esta es la única rama que debiera sacarse directamente desde master. Tan pronto como el arreglo este completo esta debería hacer merge dentro de master y develop y master debería ser taggeado con un numero de versión actualizado.



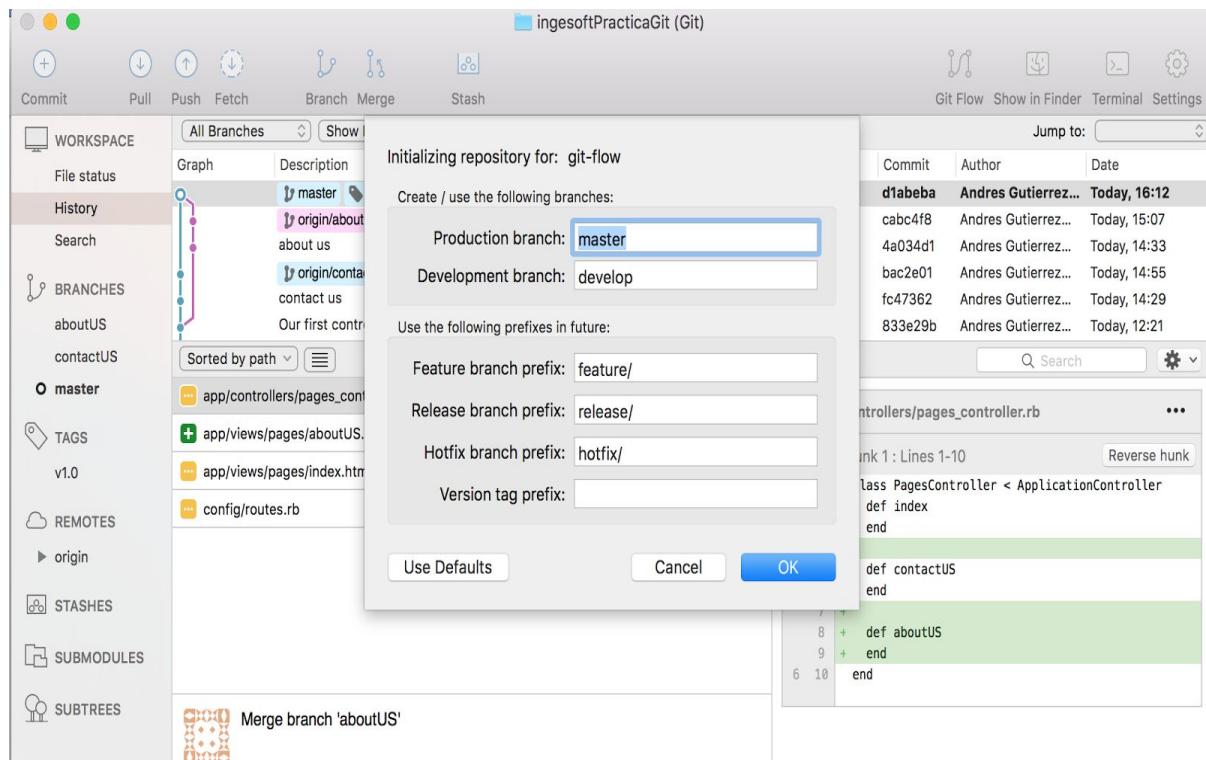
Para continuar con nuestro ejemplo vamos abrir Sourcetree



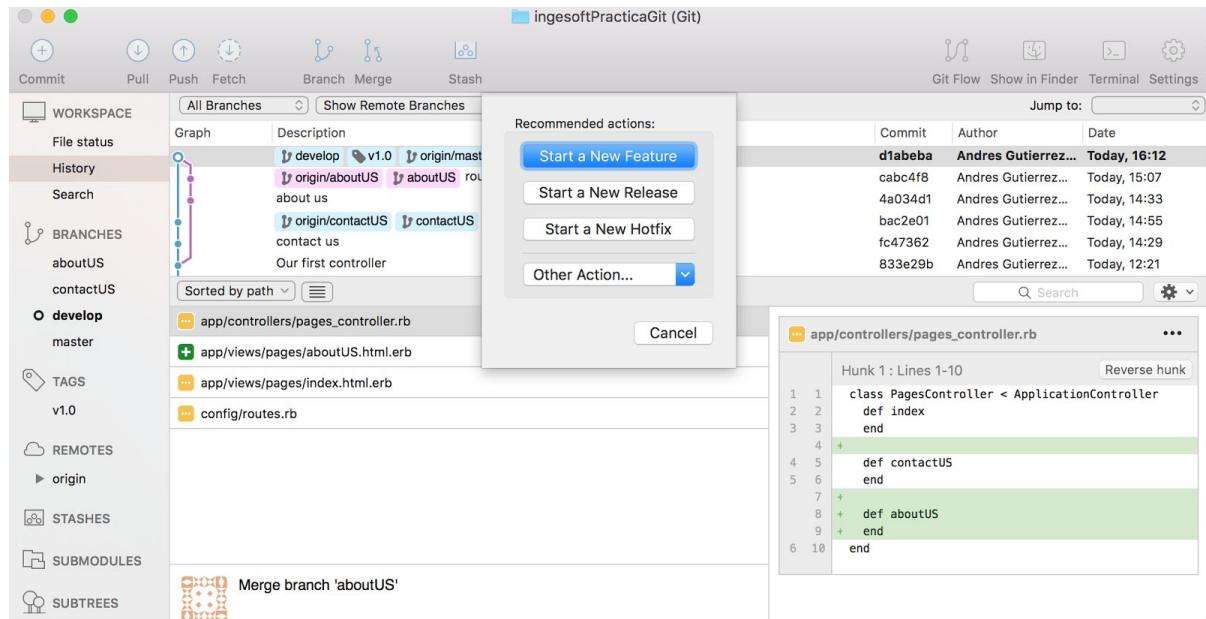
Después vamos aadir el repositorio en el que hemos estado trabajando para ello damos click en añadir repositorio local existente y seleccionamos el proyecto en donde hemos estado trabajando.



En la imagen anterior se muestra la interfaz gráfica de la herramienta con todas las funcionalidades, lo primero que vamos a hacer es dar click en el botón de la esquina superior derecha llamado “git flow” para iniciar el flujo de git.



Al hacer esto creamos una rama llamada **develop** además de prefijos o carpetas para las otras ramas de este workflow, para crear una nueva feature, hotfix o release debemos volver a dar click en el botón.



Ahora vamos a hacer un poco de trabajo en nuestra rama feature y luego haremos el merge dentro de develop.

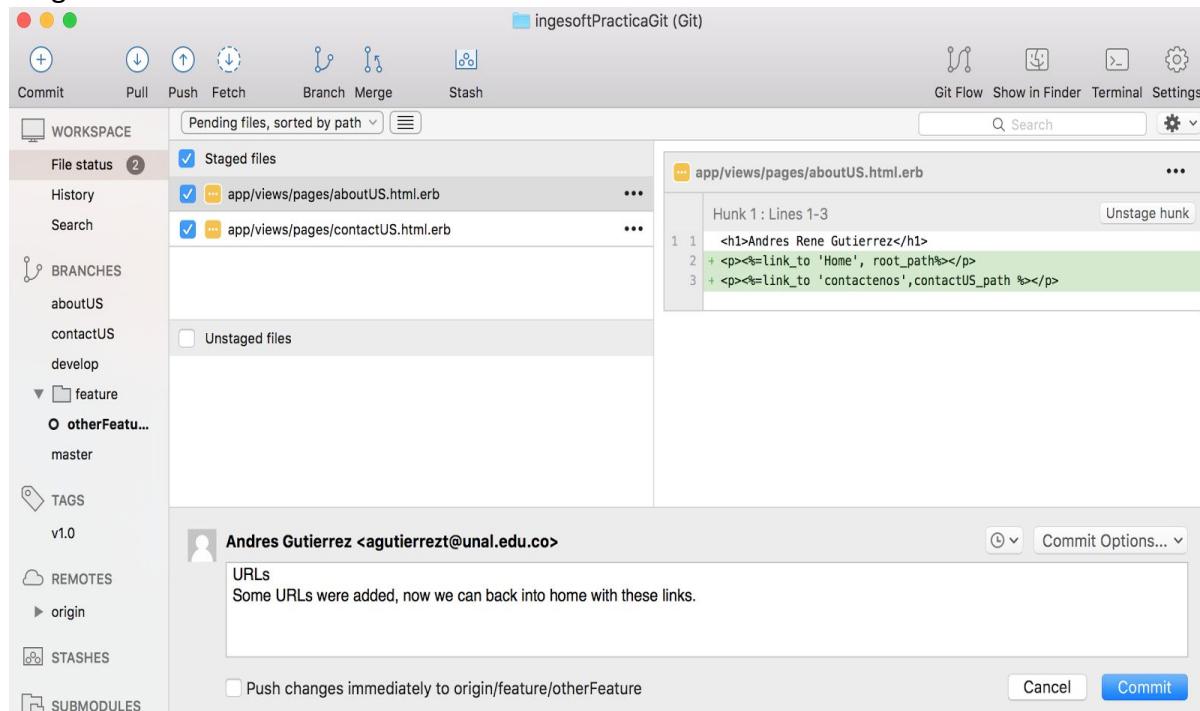
Primero vamos aadir los URLs faltantes a las páginas de de contáctenos y acerca de nosotros.

The screenshot shows a terminal window with several tabs open. The current tab is 'aboutUS.html.erb' which contains the following code:

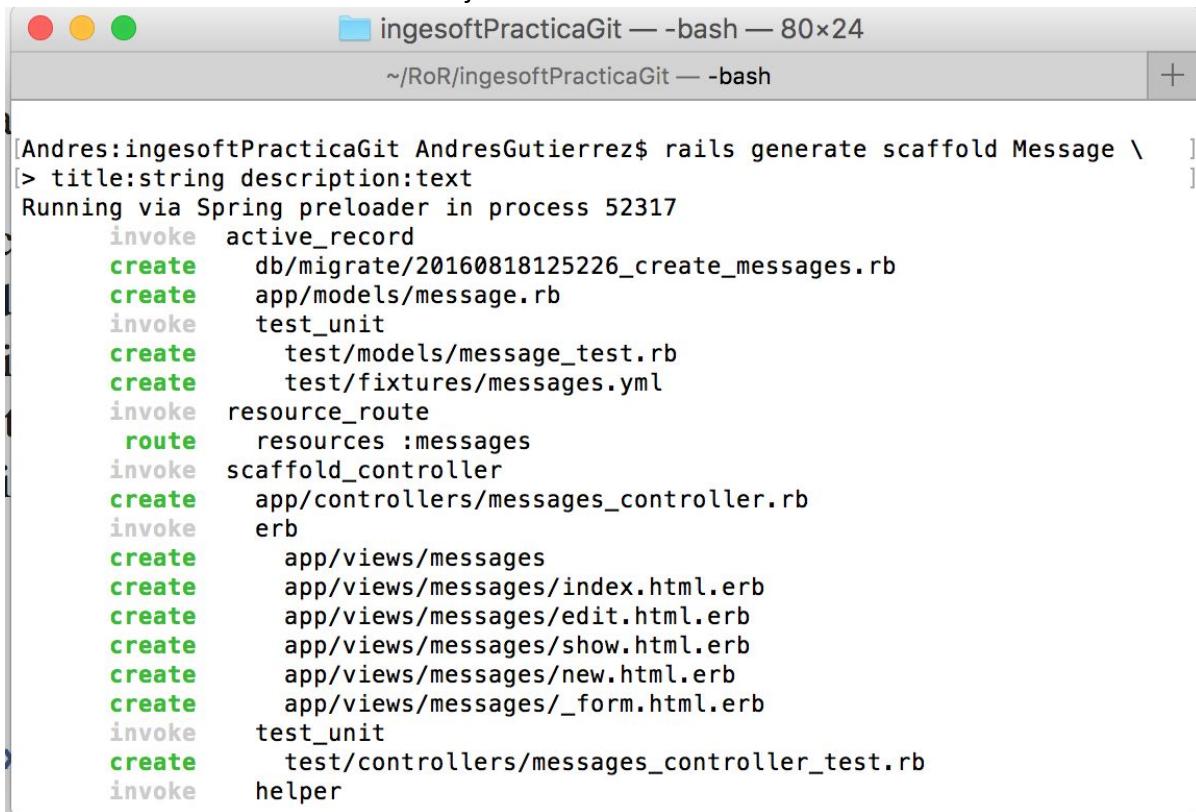
```
<h1>Andres Rene Gutierrez</h1>
<p><%=link_to 'Home', root_path%></p>
<p><%=link_to 'contáctenos', contactUS_path %></p>
```

The file path is indicated as 'app/views/pages/aboutUS.html.erb'. The status bar at the bottom shows 'LF' and 'UTF-8' encoding, and there are 2 updates available.

Luego hacemos commit a través de sourcetree.



Luego vamos a crear nuestro primer modelo de datos y utilizar el scaffolding de rails para ahorrarnos mucho de nuestro trabajo.



```
[Andres:ingesoftPracticaGit AndresGutierrez$ rails generate scaffold Message \
[> title:string description:text
Running via Spring preloader in process 52317
  invoke  active_record
    create    db/migrate/20160818125226_create_messages.rb
    create    app/models/message.rb
  invoke  test_unit
    create    test/models/message_test.rb
    create    test/fixtures/messages.yml
  invoke  resource_route
    route    resources :messages
  invoke  scaffold_controller
    create    app/controllers/messages_controller.rb
  invoke  erb
    create    app/views/messages
    create    app/views/messages/index.html.erb
    create    app/views/messages/edit.html.erb
    create    app/views/messages/show.html.erb
    create    app/views/messages/new.html.erb
    create    app/views/messages/_form.html.erb
  invoke  test_unit
    create    test/controllers/messages_controller_test.rb
  invoke
```

Este generador crea una mano de archivos, entre los más importantes se encuentra datatime_create_message.rb, este es una migración, este representa un cambio que queremos hacer ya se a la base de datos como un todo o a los datos que están contenidos en ella. Esos cambios pueden actualizar el esquema de la base de datos o los datos contenidos en las tablas de la base de datos nosotros aplicamos las migraciones para actualizar nuestra base de datos y hacemos roll back para desaplicar ellas y volver la base de datos a un estado anterior, primero vamos a ver como el archivo de migración luce y por último la vamos a correr.

```

20160818125226_create_messages.rb — /Users/AndresGutierrez/RoR/ingesoftPracticaGit
untitled x app.js x index.html.erb x aboutUS.html...x contactUS.htm...x 20160818125226_create_...x
1 class CreateMessages < ActiveRecord::Migration[5.0]
2   def change
3     create_table :messages do |t|
4       t.string :title
5       t.text :description
6
7       t.timestamps
8     end
9   end
10 end
11
db/migrate/20160818125226_
  seeds.rb
lib
log
public
test
tmp
vendor
.gitignore
config.ru
Gemfile

```

LF UTF-8 Ruby ↗ feature/otherFeature +11 2 updates

Acá podemos ver que la migración crea una nueva tabla llamada messages con los atributos que indicamos anteriormente, además rails crea unos campos de estampa de tiempo para saber cuándo los registros son creados, las migraciones puede tener 3 métodos:

change: El cambio que será aplicado cuando ejecutemos la migración.

up: es igual que change, en si up se utiliza se cuando el cambio no es trivial y necesita una operación específica para hacer roll back.

down: Es la operación que será aplicada cuando hagamos un roll back siempre que se utilice el método up es buena práctica tener su contraparte en el método down.

Entonces ustedes se preguntaran porque en la migración tenemos el método change pero no down para hacer roll back, rails es lo suficientemente inteligente y sabe hacer roll back de algunas operaciones sin necesidad de indicárselo.

```

ingesoSoftPracticaGit — -bash — 80x24
~/RoR/ingesoftPracticaGit — -bash
[Andres:ingesoftPracticaGit AndresGutierrez$ rails db:migrate
== 20160818125226 CreateMessages: migrating =====
-- create_table(:messages)
  -> 0.0069s
== 20160818125226 CreateMessages: migrated (0.0069s) =====
Andres:ingesoftPracticaGit AndresGutierrez$ ]

```

Por últimos vamos aadir un link en la página home para ver la página de mensajes, gracias al scaffolding todos estos URLs ya fueron creados estos los podemos consultar en localhost:3000/info/routes, además su controlador con operaciones y las vistas que renderizan también creadas.

The screenshot shows a browser window with the title 'localhost'. The address bar has a message from 'Lawih added you to paprika - agutierrez@unal.edu.co - Universidad...'. The main content is a table titled 'Action Controller: Exception caught' with the following data:

Helper	HTTP Verb	Path	Controller#Action
Path / Url		Path Match	
messages_path	GET	/messages(.:format)	messages#index
	POST	/messages(.:format)	messages#create
new_message_path	GET	/messages/new(.:format)	messages#new
edit_message_path	GET	/messages/:id/edit(.:format)	messages#edit
message_path	GET	/messages/:id(.:format)	messages#show
	PATCH	/messages/:id(.:format)	messages#update
	PUT	/messages/:id(.:format)	messages#update
	DELETE	/messages/:id(.:format)	messages#destroy
root_path	GET	/	pages#index

The screenshot shows a code editor with the file 'index.html.erb' open. The file content is:

```

1 <h1>Welcome to our project</h1>
2 <p><%= link_to "mensajes", messages_path %></p>
3 <p><%= link_to 'contactenos', contactUS_path %></p>
4 <p><%= link_to 'acerca de nosotros', aboutUS_path %></p>
5

```

The sidebar shows the project structure:

- app
 - assets
 - channels
 - controllers
 - helpers
 - jobs
 - mailers
 - models
 - views
 - layouts
 - messages
 - pages
 - aboutUS.html.erb
 - contactUS.html.e
 - index.html.erb
- bin
- config
- db
 - migrate
 - 20160818125226
 - schema.rb
 - seeds.rb
- lib
- log
- public
- test

Comprobamos que la aplicación esté funcionando perfectamente.

Por últimos vamos hacer algunas validaciones a nuestro nuevo modelo, comprobamos que todo funcione correctamente y hace el commit a la luego hacemos el merge a develop.

The screenshot shows a terminal window with the following details:

- File Structure:** The left pane shows the directory structure of the application:
 - ingesoSoftPracticaGit
 - app
 - assets
 - channels
 - controllers
 - helpers
 - jobs
 - mailers
 - models
 - concerns
 - application_record.rb
 - message.rb
 - views
 - layouts
 - messages
 - pages
 - aboutUS.html.erb
 - contactUS.html.erb
 - index.html.erb
 - bin
 - config
 - db
 - migrate
 - 20160818125226_.rb
 - schema.rb
 - seeds.rb
 - lib

- Code Editor:** The right pane shows the content of `message.rb`:

```
1 class Message < ApplicationRecord
2 validates :title, :description, presence: true
3 validates :description, length: {minimum: 10}
4 validates :title, uniqueness: true
5 end
```
- Bottom Status Bar:** Shows LF, UTF-8, Ruby, feature/otherFeature, +6, 2 updates, and a GitHub icon.

Aquí se crean algunas validaciones básicas sé que entienden por sí mismas, por ejemplo la primera garantiza que los campos no sean nulos, ni vacíos, la segunda exige que la descripción del mensaje se mínimo 10 y la última que el título sea único, comprobamos que los nuevos cambios funcionen bien luego añadimos algunas información a la base datos y hacemos commit.

localhost

Facebook Twitter Wikipedia Development ▾ Noticias ▾ Populares ▾ SE2 ▾

(1) Facebook Lawih added you to paprika - agutierrez@unal.edu.co IngesoftPracticaGit ruby on rails - I got the error syntax error, unexp...

New Message

3 errors prohibited this message from being saved:

- Title can't be blank
- Description can't be blank
- Description is too short (minimum is 10 characters)

Title

Description

[Create Message](#)

[Back](#)

localhost

Facebook Twitter Wikipedia Development ▾ Noticias ▾ Populares ▾ SE2 ▾

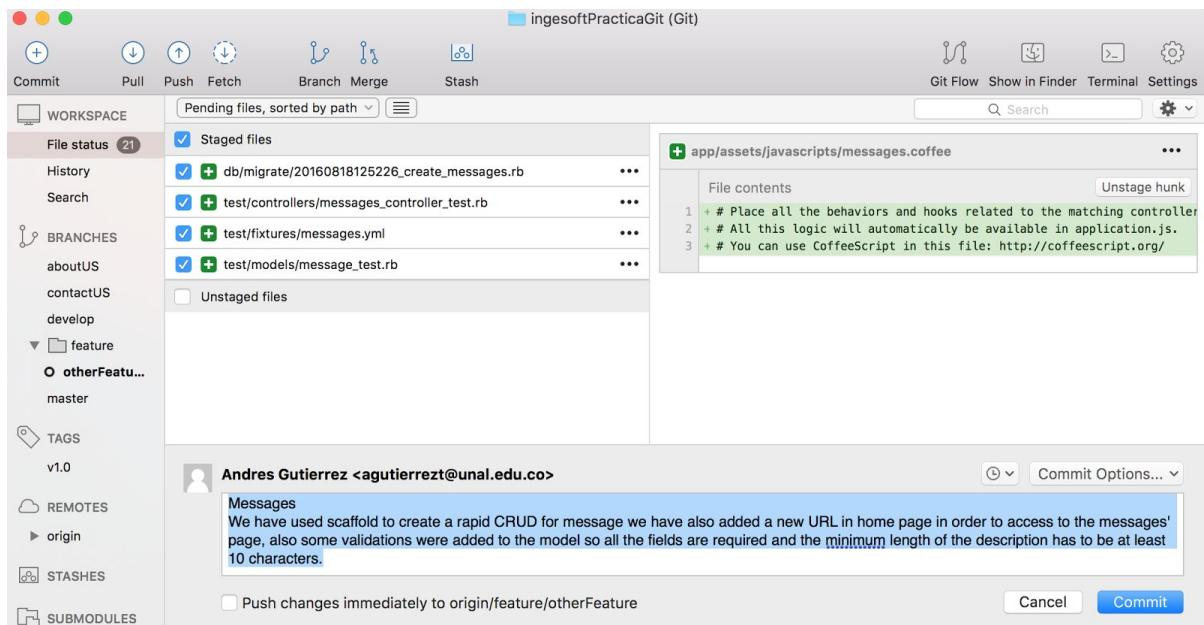
(1) Facebook Lawih added you to paprika - agutierrez@unal.edu.co IngesoftPracticaGit ruby on rails - I got the error syntax error, unexp...

Message was successfully created.

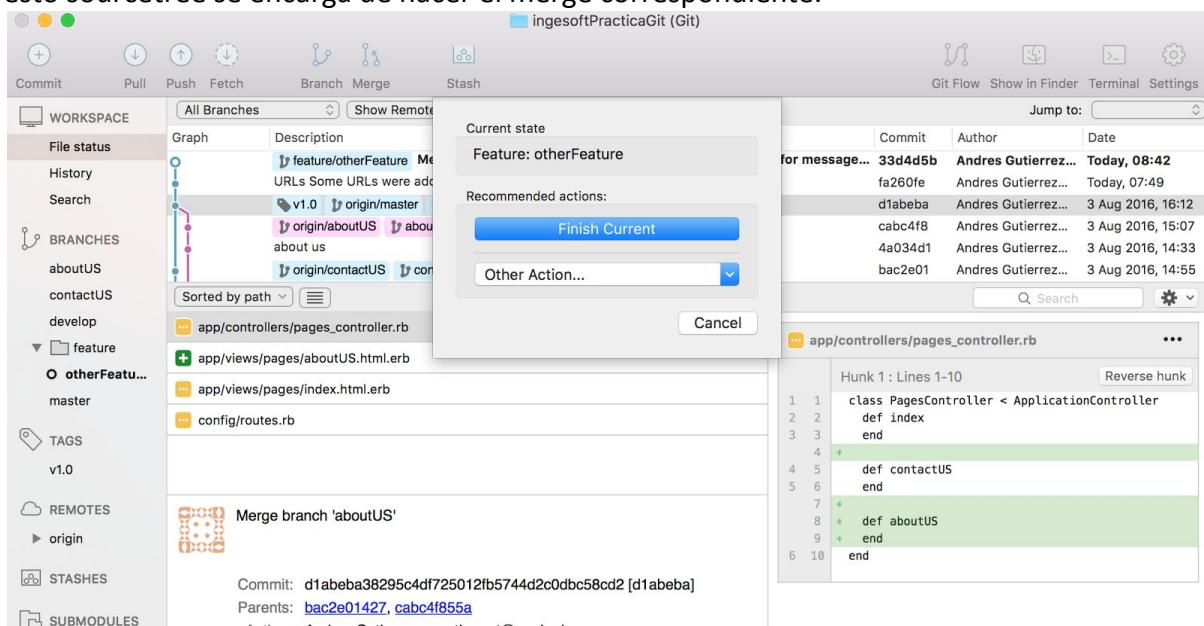
Title: Bienvenidos

Description: Les damos la bienvenida al curso de ingeniería de software2, esperamos que sea de su agrado y aprende mucho. Cordial saludo, Sus monitores

[Edit](#) | [Back](#)

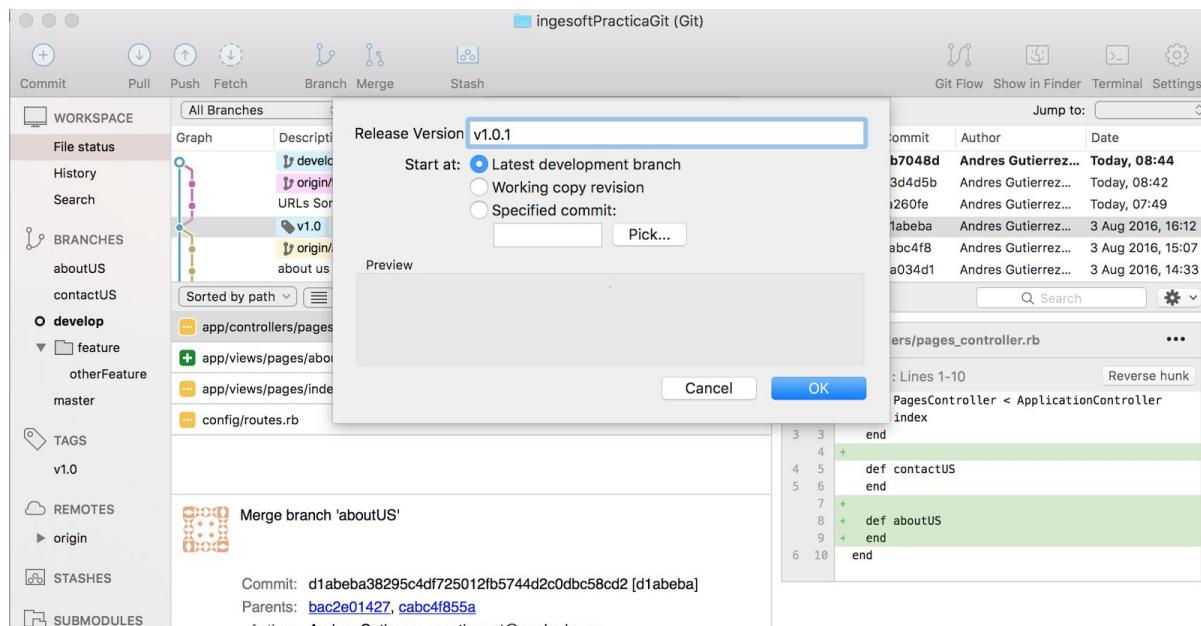
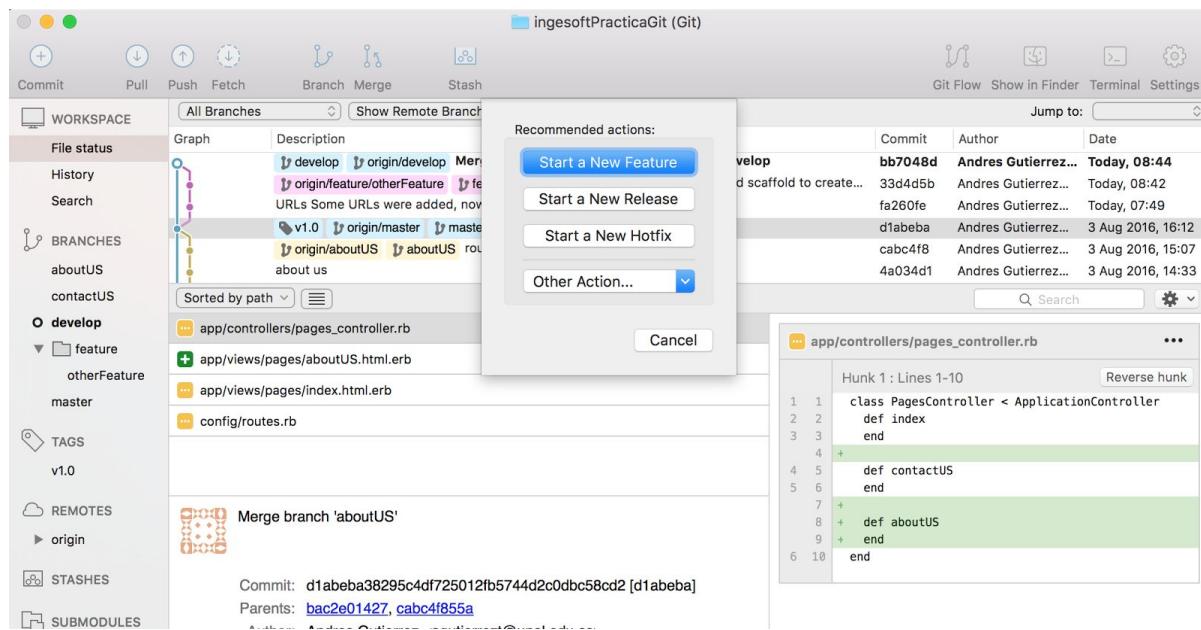


Para hacer el merge a develop vamos a botón git flow y seleccionamos terminar feature con esto sourcetree se encarga de hacer el merge correspondiente.



Despues de hacer el merge a develop, usted podrá haber notado que todos los cambios se encuentran locales, para hacerlos disponibles al repositorio central damos click en el botón push.

Un oficial release se acerca y nosotros estamos contentos con lo que hemos logrado hasta el momento, entonces vamos a crear una rama release con la ayuda del botón git flow y hacer algunos test para realmente comprobar que todo funciona correctamente.



Primero vamos a hacer test al modelo para comprobar que este funcione como está estipulado.

1. Vamos a la carpeta test y luego fixtures aquí vemos un archivo llamado message, este archivo tiene dummy data que será insertada en la base de datos de test cada vez que corramos un test, nosotros también podremos acceder a esos modelos cuando realicemos nuestro test.

2.

```

message_test.rb — /Users/AndresGutierrez/RoR/ingesoftPracticaGit
1 require 'test_helper'
2
3 class MessageTest < ActiveSupport::TestCase
4   test "message attributes must not be empty" do
5     message = Message.new
6     assert message.invalid?
7     assert message.errors[:title].any?
8     assert message.errors[:description].any?
9   end
10  test "message is not valid without a unique title" do
11    message = Message.new(title:messages(:one).title,
12                          description:messages(:one).description)
13    assert message.invalid?
14    assert_equal ["has already been taken"], message.errors[:title]
15  end
16  test "message is not valid if the description is too short" do
17    message = Message.new(title:"This is a title",description: "too short")
18    assert message.invalid?
19    assert message.errors[:description].any?
20  end
21 end
22

```

```

messages.yml — /Users/AndresGutierrez/RoR/ingesoftPracticaGit
1 # Read about fixtures at http://api.rubyonrails.org/classes/ActiveRecord/FixtureSet.html
2
3 one:
4   title: MyString
5   description: My description about IS2
6
7 two:
8   title: MyString1
9   description: My description about IS2
10

```

Segundo vamos a la carpeta de test luego modelos y realizamos los test correspondientes para comprobar que el modelo si cumple con las validaciones, a

continuación se muestra las imágenes de cómo deberían quedar los test y sus resultados.

```
ingesoftPracticaGit — bash — 80x24
~/RoR/ingesoftPracticaGit — bash ...oR/ingesoftPracticaGit — less ▾ git reflog + [Andres:ingesoftPracticaGit AndresGutierrez$ rails test:models
Run options: --seed 43518

# Running:

...
Finished in 0.032780s, 91.5194 runs/s, 213.5452 assertions/s.

3 runs, 7 assertions, 0 failures, 0 errors, 0 skips
Andres:ingesoftPracticaGit AndresGutierrez$ ]
```

3. Por ultimo vamos a hacer un test al controlador de pages, para comprobar que si estamos obteniendo las respuesta deseadas.

```
ingesoftPracticaGit — bash — 80x24
~/RoR/ingesoftPracticaGit — -bash      ...oR/ingesoftPracticaGit — less ◀ git reflog + [Andres:ingesoftPracticaGit AndresGutierrez$ rails test:controllers
Run options: --seed 55176

# Running:

.

Finished in 0.542819s, 1.8422 runs/s, 3.6845 assertions/s.

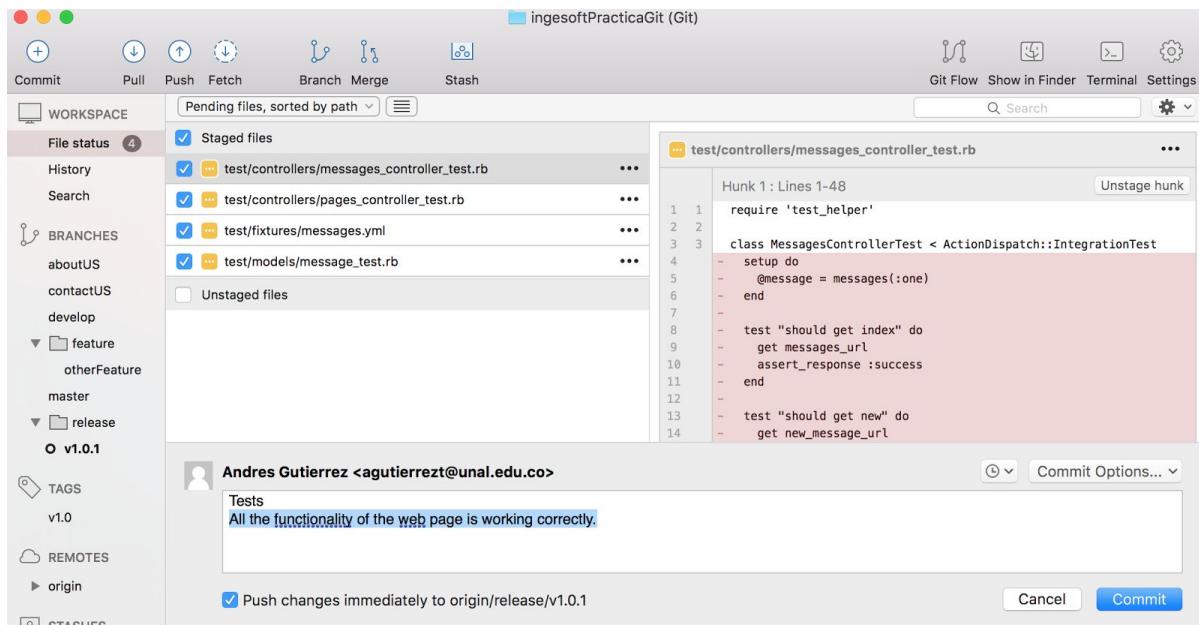
1 runs, 2 assertions, 0 failures, 0 errors, 0 skips
Andres:ingesoftPracticaGit AndresGutierrez$
```

```
pages_controller_test.rb — /Users/AndresGutierrez/RoR/ingesoftPracticaGit
unti... x app.js x inde... x abo... x con... x inde... x mes... x mes... x pages_contr... x mes... x
1 require 'test_helper'
2
3 class PagesControllerTest < ActionDispatch::IntegrationTest
4   test "should get index" do
5     get root_url
6     assert_response :success
7     assert_select 'h1', 'Welcome to our project'
8   end
9
10 end
11
```

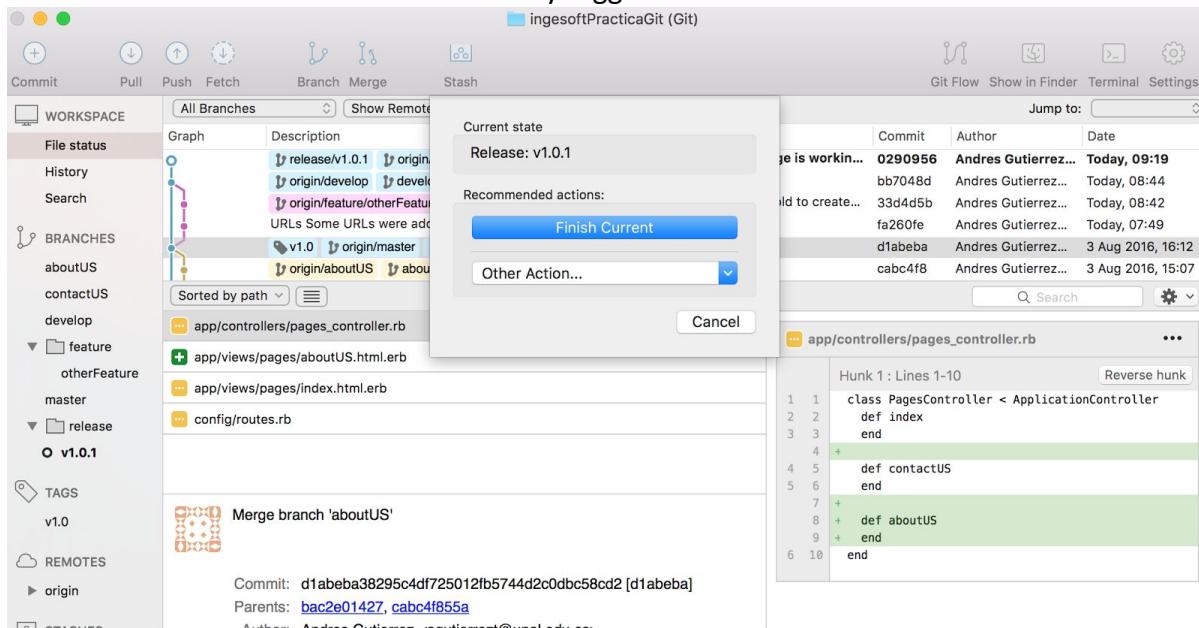
File tree:

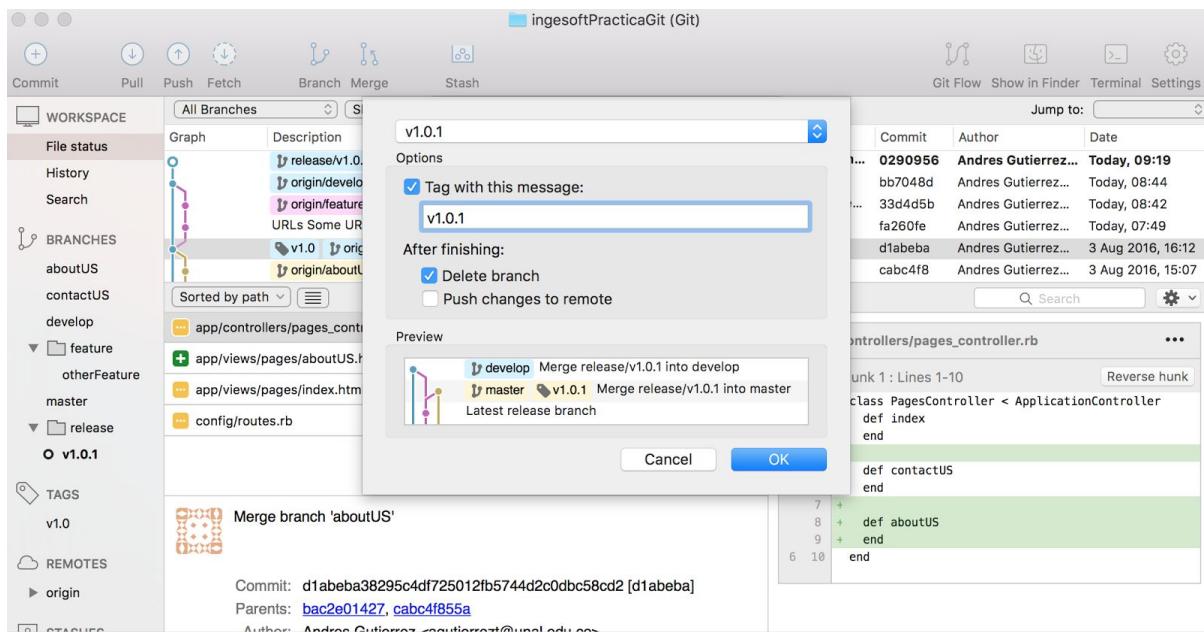
- lib
- log
- public
- test
 - controllers
 - .keep
 - messages_controller
 - pages_controller
 - fixtures
 - files
 - .keep
 - messages.yml
 - helpers
 - integration
 - mailers
 - models
 - .keep
 - message_test.rb
 - test_helper.rb
 - tmp
 - vendor
- .gitignore
- config.ru
- Gemfile
- Gemfile.lock
- Rakefile
- README.md

Si todos los test pasaron exitosamente hacemos el commit en la rama de release.



Por ultimo finalizamos la rama de release y taggeamos el commit a master.





Felicidades te has vuelto un experto en sourcetree y en git.

ENTREGABLES

1. Se entrega un documento en pdf por grupo de desarrollo.
 - a. En la parte uno cada integrante tiene que tomar un screenshot del git reflog que obtuvieron al seguir el tutorial, este screenshot debe tener la fecha y hora además del nombre de usuario de la terminal de cada integrante.
 - b. En la parte 2 se debe tomar un screenshot obtenido con el git reflog además de un screenshot del network de github, esto es uno solo por grupo.
 - c. En la parte 3 se debe tomar un screenshot del sourcetree con el grafo final además de uno en el network de github.com

La entrega debe enviarse por correo con el asunto IS2-GIT al correo is2unal@gmail.com antes del viernes 10 de febrero a la media noche.