Prepared exclusively for RubyLearning Course Participants

Git and GitHub

Revised Edition – 28th Feb. 2009
First Edition – Jan. 2009

PLEASE SUPPORT RUBYLEARNING

DONATE pledgie.com
$106.00 Raised!

Table of Contents

1. Sponsor: Josh Software 5
2. Acknowledgements 6
3. Introduction 7
   Who is the eBook for? 7
   Concept and Approach 7
   How to Use This eBook 7
   About the Conventions Used in This eBook 8
4. Resources 9
5. Git and GitHub 11
   What's Version Control? 11
   What Ruby Experts say about Git and GitHub 13
   What's Git? 15
   Downloading and Installing Git 15
      For a Windows box: 15
      For Mac OS X users: 16
      For Windows Vista users: 17
   Create a local folder 17
   Let us start using Git 17
      Windows users: 17
      Other Operating systems: 19
   Confirm Git version 19
   Asking for help 19
   Introduce yourself to Git 19
   Add some additional settings 20
   Create your SSH Key 21
      Mac OS X users: 21
   What's GitHub? 22
   Set up your GitHub account 22
   Creating a new repository 25
   Back to your local gitlocalrepo folder 28
   Exercise 1 29
   Normal workflow 30
   Create your local repository 30

Making changes ... 31
Ignoring files you do not want to track ... 31
Staging ... 32
Commit ... 33
Push changes to your repository on GitHub ... 35
Deleting a repository ... 38
Exercise 2 ... 39
Cloning a public project on GitHub to your local repo ... 40
Using gitk tool ... 43
Forking a repository ... 44
Pushing our changes ... 46
Sending a pull request ... 46
Keeping track of original repository ... 49
Pulling in ashbbrlp changes ... 49
Exercise 3 ... 50
Merging changes from a pull request ... 52
Exercise 4 ... 59
Managing collaborators on GitHub ... 59
Collaboration between collaborators ... 60
Watch projects and people ... 60
Using the project wiki ... 63
Exercise 5 ... 64
Branches ... 65
Creating a branch ... 66
Merging a branch ... 72
Deleting a branch ... 73
Reversing a merge ... 74
Exercise 6 ... 75
Git tag ... 77
6. Distributing and Releasing Ruby Libraries As Gems ... 80
Installing RubyGems ... 80
Hosting a RubyGem on GitHub ... 81
Set up our GitHub account ... 81
Create a new repository ... 81
Creating a Gem ... 82
Structuring Your Files ... 83

Creating a Specification File 83
Back to our local repo 86
Installing our RubyGem from GitHub 87
References 88
7. GitHub Pages 89
8. Gist 92
9. Appendix A 93
10. About the Author 102

Sponsor: Josh Software

Josh Software Pvt. Ltd., Pune, India is a young dynamic company started in 2007. "Josh" means enthusiasm in Hindi (Indian Language) and that is what they propagate through their work culture and ethics. Their goal is to provide satisfaction to their customers with quality deliverables and earn their goodwill.

Josh specializes in software development. Their software expertise spans across various domains like finance, medicine, engineering and IT. With the right mix of experienced professionals and highly motivated young developers, the Josh software team believes in building world class software by combining client's domain knowledge with their software skills.

The Josh team works primarily in **Ruby On Rails** but also has a vast experience in Java, .NET, C, C++ on Windows and Unix platforms.

Acknowledgements

There are a good number of people who deserve thanks for their help and support they provided, either while or before this eBook was written, and there are still others whose help will come after the eBook is released.

I'd like to thank my ‘gang’ of assistant teachers, mentors and patrons at http://rubylearning.org/ for their help in making this eBook far better than I could have done alone.

I have made extensive references to the information related to Git and GitHub, available in the public domain (wikis, blogs and articles) – refer to the **Resources** chapter.

My acknowledgment and thanks to all of them.

Thanks to **Jerry Anning** who initiated me into Git and GitHub and his timely help wherever required. I would like to thank Michèle Garoche, Satoshi Asakawa and Victor Goff (all from RubyLearning) for proofreading this eBook.

Also thanks to Tom Preston-Werner of GitHub (http://github.com/mojombo) for proofreading this eBook and making many suggestions for improvement.

Introduction

Who is the eBook for?

This Git and GitHub eBook is a starting point for people new to Git and GitHub, to learn it as quickly and easily as possible. It would help if you have some exposure to basic Ruby.

Concept and Approach

I have tried to organize this eBook in which each chapter builds upon the skills acquired in the previous chapter, so that you will never be called upon to do something that you have not already learned. This eBook not only teaches you how to do something, but also provides you with the chance to put those morsels of knowledge into practice with exercises. I have therefore included several exercises, or assignments, in this eBook so that you will have opportunities to apply your knowledge.

How to Use This eBook

I recommend that you go through the entire eBook chapter by chapter, reading the text, running the sample mentioned process and doing the assignments along the way. This will give you a much broader understanding of how things are done (and of how you can get things done), and it will reduce the chance of anxiety, confusion, and worse yet, mistakes.

It is best to read this eBook and complete its assignments when you are relaxed and have time to spare. Nothing makes things go wrong more than working in a rush. And keep in mind that the assignments in this eBook are fun, not just work exercises. So go ahead and enjoy it.

About the Conventions Used in This eBook

Explanatory notes (generally provides some hints or gives a more in-depth explanation of some point mentioned in the text) are shown shaded like this:

This is an explanatory note. You can skip it if you like - but if you do so, you may miss something of interest!

When you find some text as follows:

Therefore, type the following in the bash shell:

```
A@COMP /e/GitTest
$ git clone git://github.com/IndianGuru/advgame.git
```

What you need to type is the text in bold. In the above example it is:
`git clone git://github.com/IndianGuru/advgame.git`

Finally, if you notice any errors or typos, or have any comments or suggestions or good exercises I could include, please email me at:
mail@satishtalim.com.

Resources

Version Control –
http://hoth.entp.com/output/git_for_designers.html

Git Community Book –
http://book.git-scm.com/

GitHub Guides –
http://github.com/guides/home

Git Documentation –
http://git.or.cz/gitwiki/GitDocumentation

Git Wiki –
http://git.or.cz/gitwiki

GitCasts –
http://gitcasts.com/

Everyday Git with 20 Commands or so –
http://www.kernel.org/pub/software/scm/git/docs/everyday.html

Git Ready – Daily Tips -
http://gitready.com/

Git for the lazy –
http://www.spheredev.org/wiki/Git_for_the_lazy

Git Cheat Sheet –
http://cheat.errtheblog.com/s/git/

Pastie –
http://gist.github.com/

Insider Guide to GitHub (Free screencast) –
http://pragprog.com/screencasts/v-scgithub/insider-guide-to-github

Learn GitHub –
`http://learn.github.com/`

Git and GitHub

What's Version Control?

Version control also known as *source control* or *source code management* or *revision control* in its simplest form is saving your first draft, and then saving it as draft 2 and continuing on with your editing. This allows you some means of going back to a previous version if you decide that the direction you took in your document is not the direction you want to go, and you want to continue on from where your first draft first was. Version control in a more granular manner is being able to see each change as it was made. In reality, it ends up being likely that you want something between these two extremes.

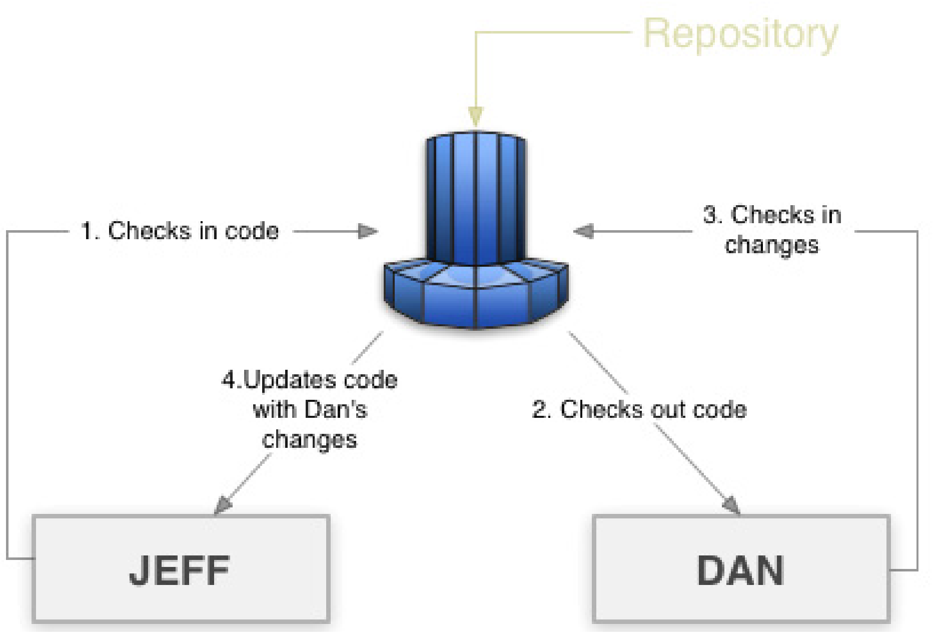
Thus, *version control* is a system that maintains versions of files at progressive stages of development. Every file in the system has a full history of changes, and can easily be restored to any version in its history.

The simplest *version control system* consists of a *repository* where all the files and their versions live. Quite simply, a *repository* works like a database; it can return any version of any file within, or a history of changes for any file, or indeed a history of changes across the entire project.

The repository's users can *check out* a *working copy*, which is a copy of the latest files to which users can make changes. After making some changes, they can then *check in* (or *commit*) the changes back to the repository, which creates a new version with metadata about the files that were changed and the person who changed them.

Each user has a full copy of the *repository* on their local machine. Generally, you will commit changes to your local repository and, once it is complete, *push* your work to the shared repository for your team. You can also *pull* changes from other repositories.

Reference: http://hoth.entp.com/output/git_for_designers.html



[Figure: A basic source control system
Source: http://hoth.entp.com/output/scm.png **]**

Some vocabulary you need to remember:

- *version control* or *source control* or *source code management* or *revision control*
- *repository*
- *check out* a *working copy*
- *check in* (or *commit*)
- *push* and *pull*

For details see here –

http://en.wikipedia.org/wiki/Version_control_system#Common_vocabulary

What Ruby Experts say about Git and GitHub

Fabio Akita: *The first week I used Git I despised it. I was a long time Subversion user. But after that week it grew on me in ways that I had never imagined. It gave me much more power and flexibility than I was used to and make my development routine a joy. When Github arrived it instantly won me over. It evolved a lot, and personally knowing the people behind it I recommend it all the time. They nailed it when they made it a 'social network' for code collaboration. It is very easy to start a new project and have people contributing code without having to deal with messy diff files all the time. Git and Github makes for the ideal development environment, without a doubt.*

Ryan Bates: *If you haven't already, I highly recommend signing up to GitHub. It has made a huge impact on improving myself as a programmer. Get involved with other programmers, read their code, and share your own code.*
http://rubylearning.com/blog/2009/01/20/little-known-ways-to-ruby-mastery-by-ryan-bates/

Thibaut Barrere: *A new Ruby programmer should focus on source control - even if you work alone, even for a small "labs" type of applications, use SVN, Git or whatever you feel comfortable with.*
http://rubylearning.com/blog/2009/01/13/little-known-ways-to-ruby-mastery-by-thibaut-barrere/

Josh Susser: *First off, get an account on GitHub. (If you haven't learned Git yet, get going on that too - it's the future for open source SCM.) Then do some exploring around the various projects and see where you think you can jump in and contribute something. Pick a project that is currently under development and has an active community. You'll have enough going on that you don't want to be championing a project just yet.*
http://rubylearning.com/blog/2009/01/06/little-known-ways-to-ruby-mastery-by-josh-susser/

Ilya Grigorik: *Get involved in the Ruby community. Join GitHub.*
http://rubylearning.com/blog/2008/12/30/little-known-ways-to-ruby-mastery-by-ilya-grigorik/

Chris Matthieu: *You must check out GitHub. It's a source code repository and a social network for developers. Most frameworks (including Rails) and gems and plug-ins exist at GitHub and your can download them and/or contribute on them by simply forking repositories.*
http://rubylearning.com/blog/2008/12/09/little-known-ways-to-ruby-mastery-by-chris-matthieu/

Ezra Zygmuntowicz: *Get yourself a GitHub account and start forking a few projects and poking around. Find one you are interested in and contribute some tests or docs to help yourself get up to speed with the project. Before you know it, you will be fluent in many libraries and techniques.*
http://rubylearning.com/blog/2008/12/02/little-known-ways-to-ruby-mastery-by-ezra-zygmuntowicz/

Guy Naor: *For finding Ruby projects, the best bet is just look at the project that interest you on the mailing lists, RubyForge or GitHub, then contact the maintainer/developer.*
http://rubylearning.com/blog/2008/11/04/little-known-ways-to-ruby-mastery-by-guy-naor/

Chris O'Sullivan: *There's a gazillion open source projects to choose from, and they can all be found at GitHub! Thanks to Git (and GitHub), contributing to a project has never been easier!*
http://rubylearning.com/blog/2008/10/28/little-known-ways-to-ruby-mastery-by-chris-osullivan/

Dr. Nic Williams: *Think of existing OSS projects as clubhouses: initially each project only has one lonely developer. They definitely want people to join their clubhouse. They'll be more than helpful to show you how to join the clubhouse, and probably quickly give you the keys to the door so you can help yourself. They'll show you how to use SVN or Git, how to make changes, how to run the tests, how to fix bugs, and how to prepare patches or push new Git forks.*
http://rubylearning.com/blog/2008/10/07/little-known-ways-to-ruby-mastery-by-dr-nic-williams/

What's Git?

Git is an open source *version control system* designed to handle very large projects with speed and efficiency, but just as well suited for small personal repositories (collection of resources that can be accessed to retrieve information); it is especially popular in the open source community, serving as a development platform for projects like the *Linux Kernel*, *Ruby on Rails*, *WINE* or *X.org*.

Git falls in the category of *distributed source code management tools* (which means that it can work almost entirely offline). You don't need to know what that means to use this eBook, but if interested check this URL –
http://en.wikipedia.org/wiki/Distributed_revision_control

Every Git working directory is a full-fledged *repository* with complete history and full revision tracking capabilities, *not* dependent on network access or a central server. Still, Git stays extremely fast and space efficient.

For a more detailed look at Git, read Git on Wikipedia –
http://en.wikipedia.org/wiki/Git_(software)

Downloading and Installing Git

To download and install Git, the precompiled packages are available here:
http://git.or.cz/

Select the relevant package for your operating system.

For a Windows box:

Git still has some issues on the Windows platform but for normal usage the **`msysgit`** package shouldn't let you down.

Download and install it from the url:

http://code.google.com/p/msysgit/downloads/list

and select the **current version available.** Download – **`Git-1.6.1-preview20081227.exe`**

Install by running the EXE installer. Accept the default install directory. When you get to the "Adjusting your PATH environment" setting screen, it is recommended for Windows users to use the "Use Git Bash only" option. Next select "Use OpenSSH".

Note that Git also runs on top of Cygwin (a POSIX emulation layer), although it is noticeably slower, especially for commands written as shell scripts. This is primarily due to the high cost of the fork emulation performed by Cygwin. However, the recent rewriting of many Git commands implemented as shell scripts in C has resulted in significant speed improvements on Windows. Regardless, many people find a Cygwin installation too large and invasive for typical Windows use. **Reference**:
http://en.wikipedia.org/wiki/Git_(software)

For Mac OS X users:

1. Michèle Garoche says that *you may find that the version offered for your operating system is outdated (it is preferable to use Git version 1.6.1.9 – refer "**Confirm Git version**" section later on, to check your Git version). In this case, you may compile Git yourself. To do that, read the note: "Compiling Git for Mac OS X Leopard" and change the version of Git to the latest one –*
http://blog.kineticweb.com/articles/2007/10/30/compiling-git-for-mac-os-x-leopard-10-5

It works also on Mac OS X Tiger. You may drop the make test as it takes a long time to complete. You don't need to compile the documentation as it needs some extra tools to compile and you may always read the documentation online.

2. Hector Sansores recommends an easier way to install Git on Mac OS X is using the git-osx-installer (**Note**: This is available only on Leopard.).

http://code.google.com/p/git-osx-installer/

3. Cynthia Sadler shows you how to install Git on Mac OS X 10.4 (Tiger) –

http://geekythoughtbubbles.blogspot.com/2009/02/how-to-install-git-on-mac-os-x-104.html

For Windows Vista users:

Al Snow says that *you should remember to use "run as administrator" right button; 3rd option down) when you open the browser prior to installing msysgit as a regular user*.

Create a local folder

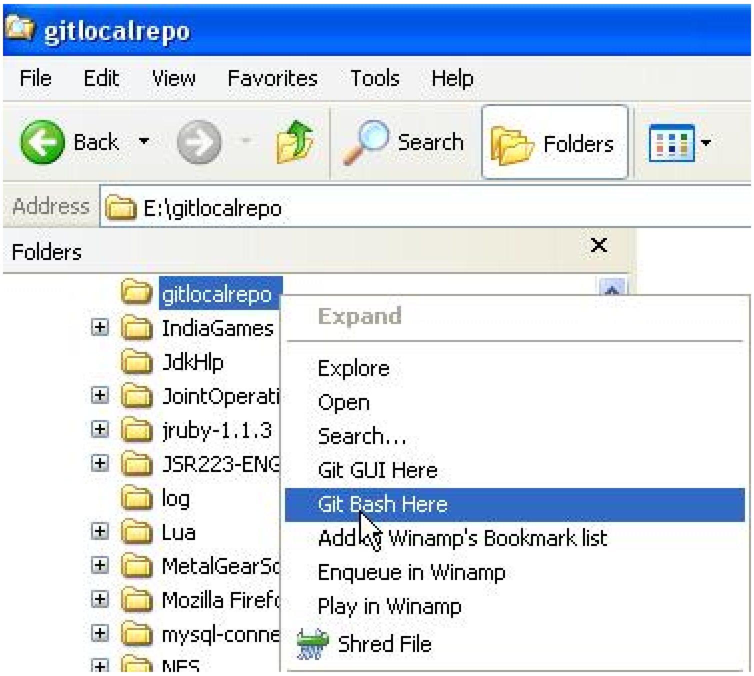
Create a new empty directory in any convenient location on your hard disk (I created and use the folder **`gitlocalrepo`** on E: of my hard disk).

Let us start using Git

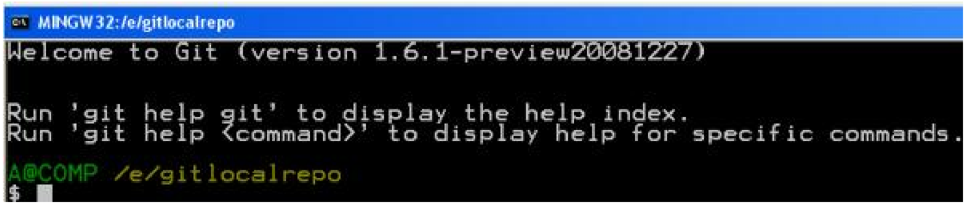
Git is primarily a command line tool and most of the examples in this eBook will show the command line versions of the functions.

Windows users:

In Windows Explorer, right mouse click on your local folder (eg. **`gitlocalrepo`**) and choose "Git Bash Here" as shown in the following screenshot:



This opens up a command window (bash shell) as shown in the next screenshot:



```
MINGW32:/e/gitlocalrepo
Welcome to Git (version 1.6.1-preview20081227)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

A@COMP /e/gitlocalrepo
$
```

Windows users should note that the bash shell is significantly different from the cmd.exe shell you may be used to. Refer to the documentation here –

http://www.gnu.org/software/bash/manual/bashref.html

Other Operating systems:

Here, the users would have to open the console or the Terminal. **Remember**, you need to be inside the **`gitlocalrepo`** folder.

Confirm Git version

Let us first check and confirm the version of Git that we have installed. Type as shown:

```
A@COMP /e/gitlocalrepo
$ git --version
git version 1.6.1.9.g97c34
A@COMP /e/gitlocalrepo
$
```

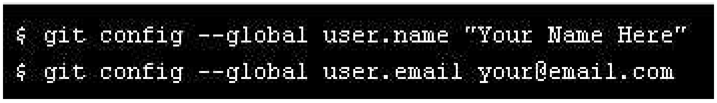
Asking for help

If you need help with any of the commands, you can type '--help' and it will show you the *man* page in your browser window. You can also type '**`git help`** *command*' for the same thing.

```
$ git log --help
$ git help log
```

Introduce yourself to Git

For all operating users, you now need to identify yourself to Git (you need to do this only *once*). With the bash shell still open type in the following:



```
$ git config --global user.name "Your Name Here"
$ git config --global user.email your@email.com
```

Substitute in your own user name and email (Note that Git does not allow accented characters in user name). *This will set the info stored when you commit to a Git repository*. Git has now been set up. You will use these with GitHub later and, when you learn how, for convenience preparing and sending patches as well as to identify your repositories.

Add some additional settings

Since we are going to work with Ruby code which is not whitespace sensitive, the following configuration tells Git to ignore differences in whitespace that otherwise may be odd. Type:

```
$ git config --global apply.whitespace nowarn
```

We can also add some colors:

```
$ git config --global color.status auto
$ git config --global color.branch auto
```

Let us confirm what we added to **config** by typing:

```
A@COMP /e/gitlocalrepo
$ git config --list
core.symlinks=false
core.autocrlf=true
color.diff=auto
pack.packsizelimit=2g
help.format=html
user.name=Satish Talim
user.email=satish_talim@hotmail.com
apply.whitespace=nowarn
color.status=auto
color.branch=auto
A@COMP /e/gitlocalrepo
$
```

Notice that Git has already added some settings, by default on Windows. Also, the git config settings are stored in a .gitconfig file in the user's home directory.

For full details on other configuration options, refer to:
http://www.kernel.org/pub/software/scm/git/docs/git-config.html

Create your SSH Key

The first step in using Git is to create your SSH Key. This will be used to secure communications between your machine and other machines, and to identify your source code changes. (If you already have an SSH key for other reasons, you can use it here, there is nothing Git-specific about this.)

To create our ssh key, type the command:

```
$ ssh-keygen -C "username@email.com" -t rsa
```

(with your own email address, of course).

Accept the default key file location. When prompted for a passphrase, make one up and enter it. If you feel confident that your own machine is secure, you can use a blank passphrase, for more convenience and less security. Note where it told you it stored the file. On the machine I tested with, it was stored in "C:\Documents and Settings\A\.ssh\". **Memorize your passphrase carefully. If you forget it, you will NOT be able to recover it.**

Open the file **`id_rsa.pub`** with a text editor. The text in there is your "public SSH key". *You will need it to set up your GitHub account soon*, at –
http://github.com/

Mac OS X users:

(Thanks to Michèle Garoche for this note).

To copy and paste your public SSH key into the dialog box on GitHub, use pbcopy and pbpaste, otherwise you'll end up with line breaks in your SSH key, making it invalid.

Copying and pasting the SSH key is different from system to system and it is better to refer to the following url –

http://github.com/guides/providing-your-ssh-key

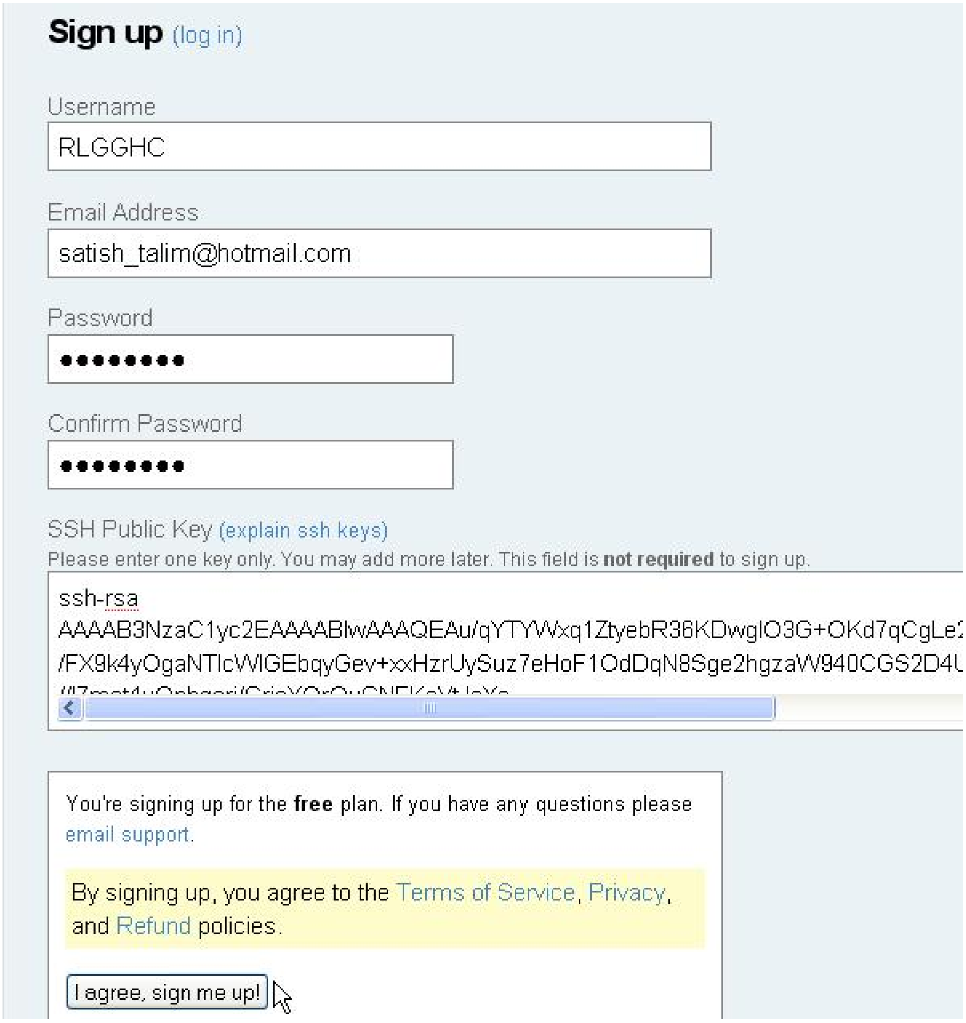
What's GitHub?

With GitHub you can host your public and private projects there and use it to collaborate on projects. In this short eBook, you'll learn the essential features you'll end up using every day. You'll come away ready to host your projects and contribute to other projects, and feeling like a GitHub insider!

Set up your GitHub account

Go to https://github.com/signup/free and sign up for a free account. Remember, this account can have unlimited public repositories and public collaborators (the total number of users who may read, write and fork your public repositories). Also, make sure that Javascript is enabled in your browser.

In the sample screenshot shown on the next page, I signed up with **satish_talim@hotmail.com**. *Please use your own real email address*. The username I typed was **RLGGHC**. *You must use a different username*. The SSH public key is important and is used when you push your code to GitHub which needs to validate who you are. Be sure you pick the public SSH key (the one with "ssh-rsa" as 1st line). If you want multiple GitHub accounts, refer to –
http://github.com/guides/multiple-github-accounts


Sign up (log in)

Username

RLGGHC

Email Address

satish_talim@hotmail.com

Password

●●●●●●●●●

Confirm Password

●●●●●●●●●

SSH Public Key (explain ssh keys)

Please enter one key only. You may add more later. This field is **not required** to sign up.

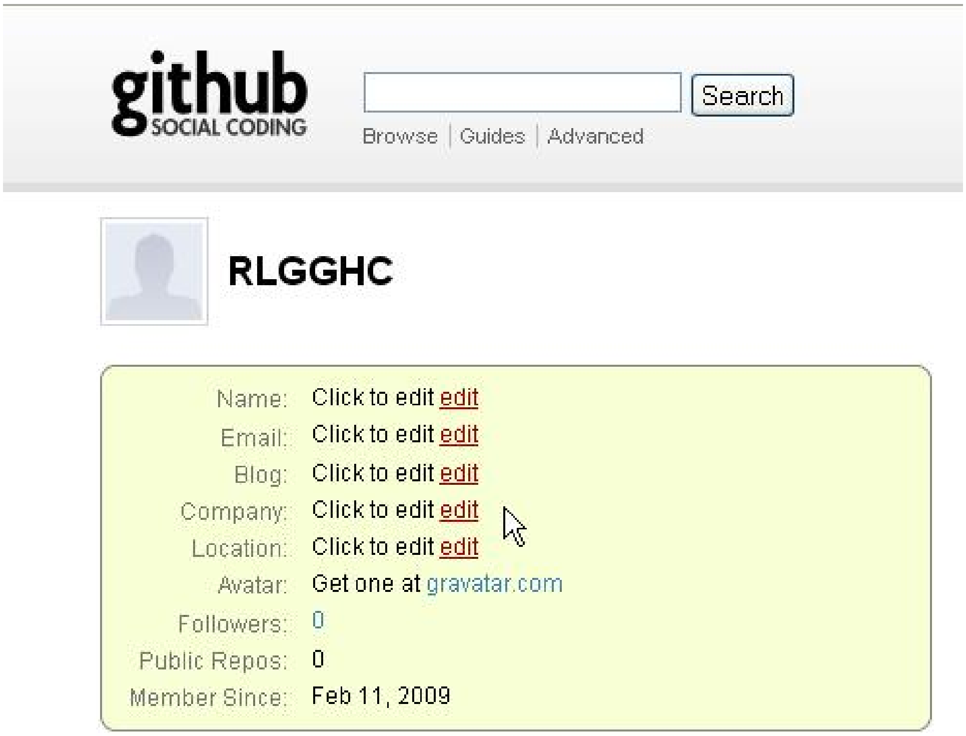
```
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAu/qYTYWxq1ZtyebR36KDwglO3G+OKd7qCgLe
/FX9k4yOgaNTlcWlGEbqyGev+xxHzrUySuz7eHoF1OdDqN8Sge2hgzaW940CGS2D4
```

You're signing up for the **free** plan. If you have any questions please email support.

By signing up, you agree to the Terms of Service, Privacy, and Refund policies.

The GitHub *account* I just created (see next screenshot) is at –

http://github.com/RLGGHC

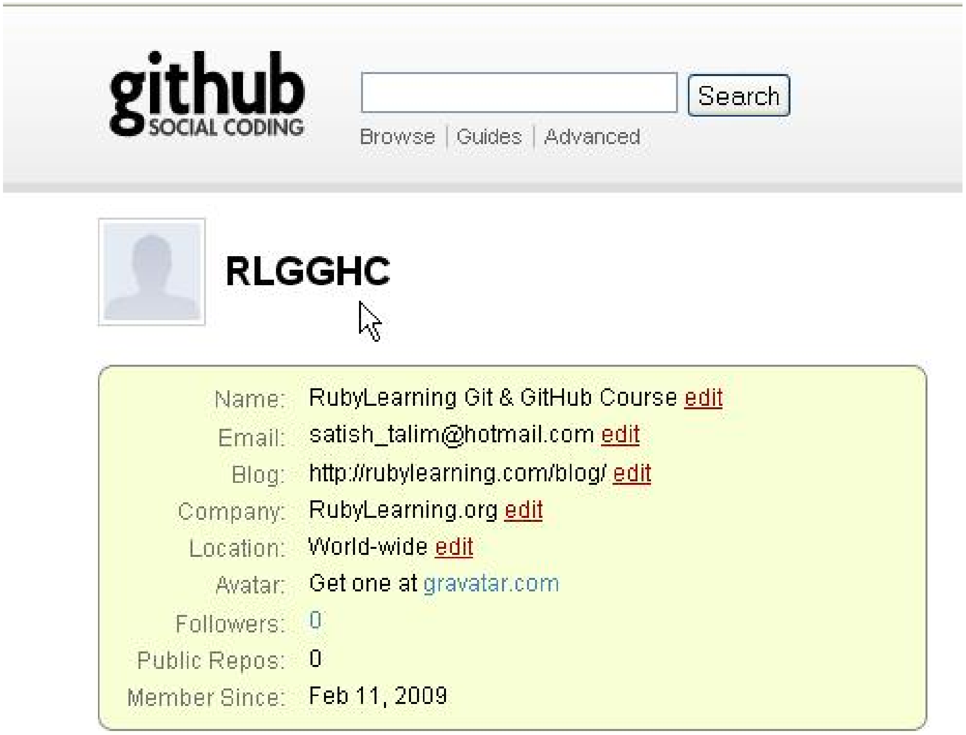


RLGGHC

Name:	Click to edit edit
Email:	Click to edit edit
Blog:	Click to edit edit
Company:	Click to edit edit
Location:	Click to edit edit
Avatar:	Get one at gravatar.com
Followers:	0
Public Repos:	0
Member Since:	Feb 11, 2009

Please click on the edit links (see above screenshot) and fill in the relevant details in *your* GitHub account.

The next screenshot shows what I filled in (i.e. on my 'profile' page):



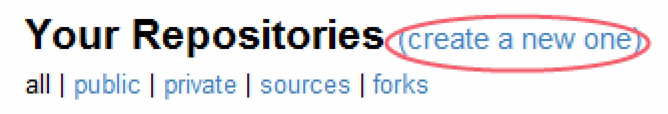
RLGGHC

Name:	RubyLearning Git & GitHub Course edit
Email:	satish_talim@hotmail.com edit
Blog:	http://rubylearning.com/blog/ edit
Company:	RubyLearning.org edit
Location:	World-wide edit
Avatar:	Get one at gravatar.com
Followers:	0
Public Repos:	0
Member Since:	Feb 11, 2009

Creating a new repository

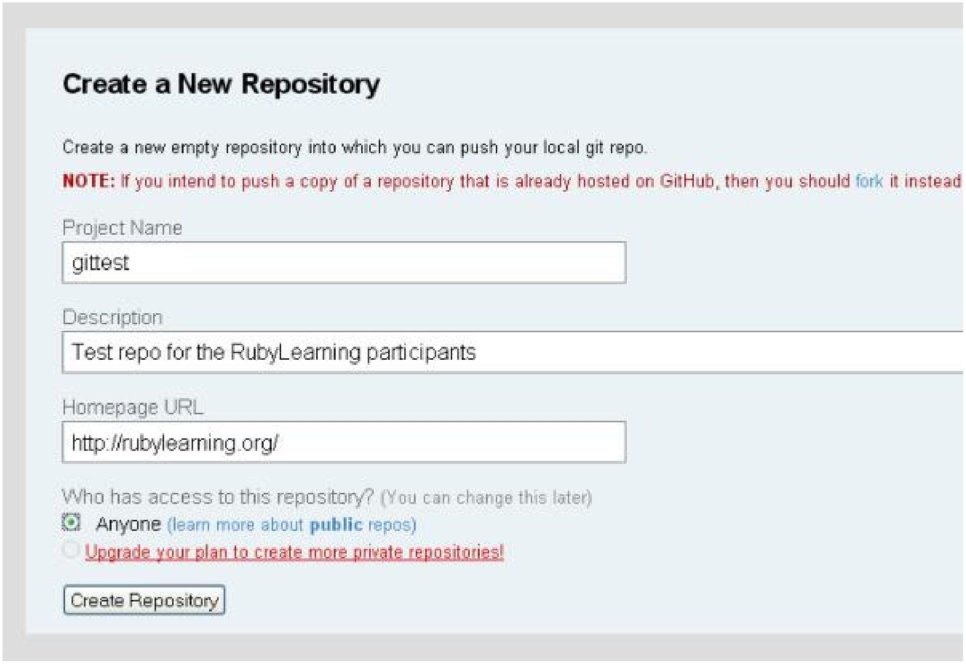
Now, I would like to work on a simple local project called **`gittest`,** both on my local machine and on GitHub. For this, I will need to create a repository. *Please create your own local project with a different name.*

You are already logged into your GitHub account you just created above. If you click on the 'dashboard' link at the top right-hand-side of your GitHub page, you will see something like the following screenshot:


Your Repositories (create a new one)
all | public | private | sources | forks

Click on – “Your Repositories (create a new one)” to create a new repository.

This is shown in the next screenshot:



Create a New Repository

Create a new empty repository into which you can push your local git repo.

NOTE: If you intend to push a copy of a repository that is already hosted on GitHub, then you should fork it instead

Project Name: gittest

Description: Test repo for the RubyLearning participants

Homepage URL: http://rubylearning.org/

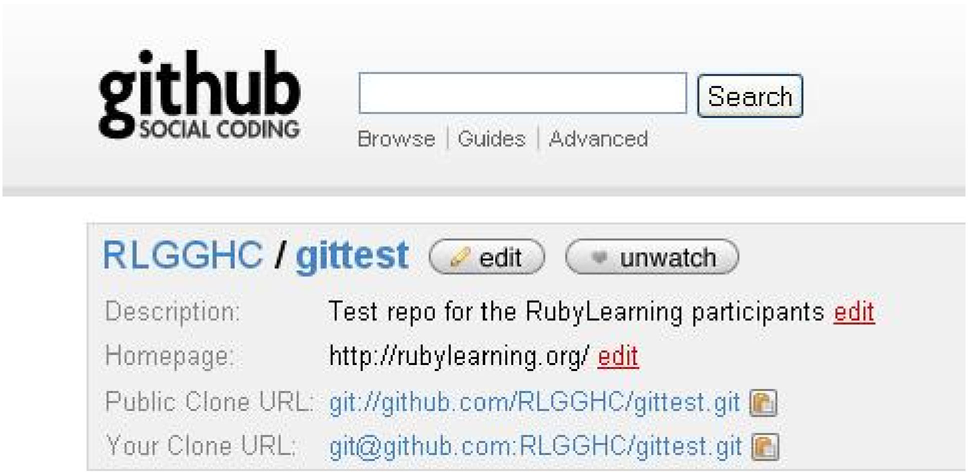
Who has access to this repository? (You can change this later)

[✓ Anyone] (learn more about **public** repos)

Upgrade your plan to create more private repositories!

The next screenshots shows the result:

NOTE: In the examples that follow, a sample repository name of ‘**`gittest`**’ will be used, but it is NOT necessary that you name your repository ‘**`gittest`**’.



RLGGHC / gittest

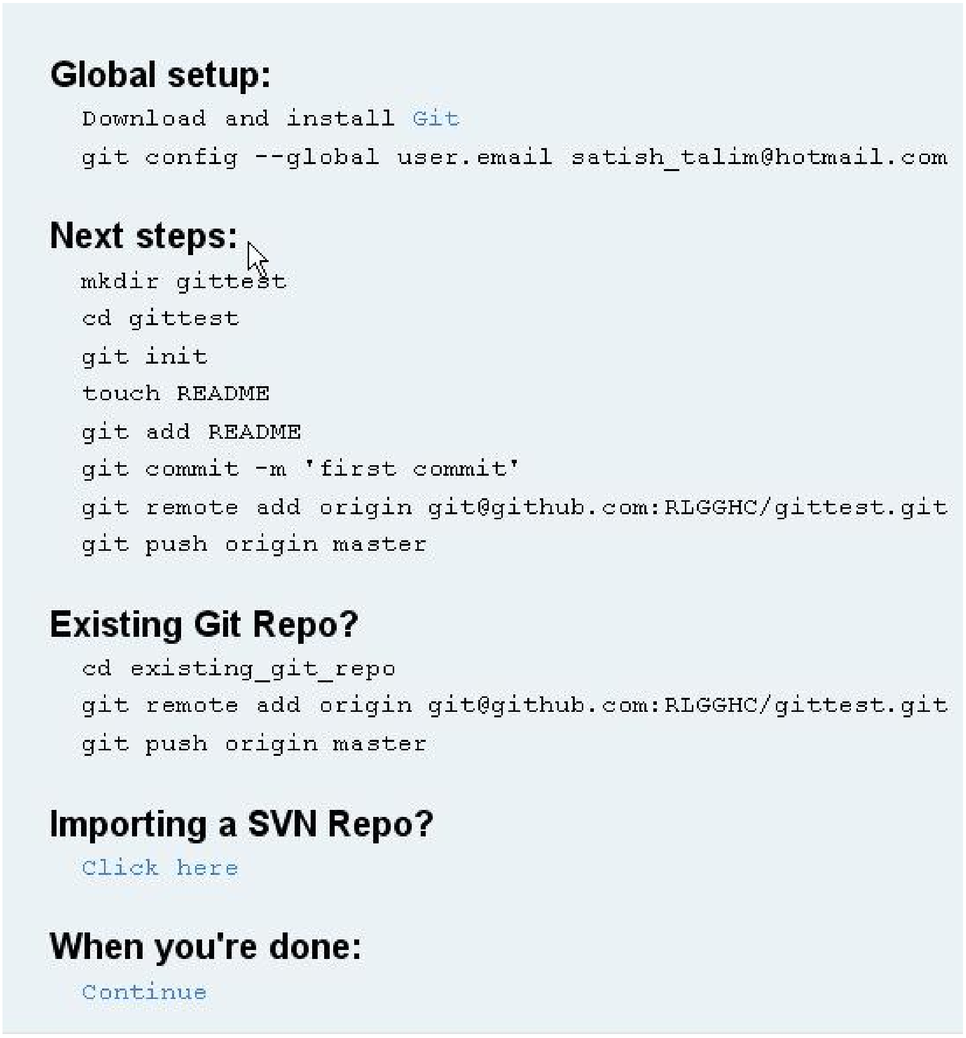
Description: Test repo for the RubyLearning participants edit

Homepage: http://rubylearning.org/ edit

Public Clone URL: git://github.com/RLGGHC/gittest.git

Your Clone URL: git@github.com:RLGGHC/gittest.git

Screenshot of the new repo - gittest



Global setup:

```
Download and install Git
git config --global user.email satish_talim@hotmail.com
```

Next steps:

```
mkdir gittest
cd gittest
git init
touch README
git add README
git commit -m 'first commit'
git remote add origin git@github.com:RLGGHC/gittest.git
git push origin master
```

Existing Git Repo?

```
cd existing_git_repo
git remote add origin git@github.com:RLGGHC/gittest.git
git push origin master
```

Importing a SVN Repo?

Click here

When you're done:

Continue

Screenshot of the screen after a new repo gittest was created

Back to your local gitlocalrepo folder

Note: In the steps shown below, please remember to replace **`gittest`** with the name of your repository on GitHub.

Refer –
http://github.com/guides/local-github-config

As mentioned in the above url, go to your GitHub 'account' page and click on "Global Git Config". A window pops up that says –

```
Copy and paste this into your terminal:

git config --global github.user RLGGHC
git config --global github.token 605966357…
```

With the bash shell still open, type in the following:

```
A@COMP /e/gitlocalrepo
$ git config --global github.user RLGGHC
$ git config --global github.token 605966357…
```

*Replace **RLGGHC** with your account name and **605966357…** with your token.*

The above two commands tell GitHub aware apps like GitNub or Command Line Gist, who you are. *Keep Your Token A Secret*. It's like your password - please keep it secret. Also: whenever you change your password, your token will also change. So if your token gets out, change your GitHub password and you will be fine.

Exercise 1

Try and find out the public clone url of mattetti / merb-book. What is it?

Normal workflow

As we will soon see, once you have a Git repository and everything setup, the workflow is not going to be much different from working with any other source control system, the only real difference should be the staging process. The workflow will generally go something like this:

- synchronize with remote repository
- modify files locally
- see what you've changed locally
- stage the changes you want to commit, locally
- commit your staging area, locally
- rinse, repeat if necessary
- push to remote repository

Create your local repository

Let us first create our local repository (should be with the same name as the name of your repository on GitHub). Type the following:

```
A@COMP /e/gitlocalrepo
$ mkdir gittest
A@COMP /e/gitlocalrepo
$ cd gittest
A@COMP /e/gitlocalrepo/gittest
$
```

Next we shall put our local folder **`gitlocalrepo/gittest`** under Git management. Type:

```
A@COMP /e/gitlocalrepo/gittest
$ git init
Initialized empty Git repository in
e:/gitlocalrepo/gittest/.git/
A@COMP /e/gitlocalrepo/gittest
$
```

The above command needs to be executed only *once.*

The **git init** command creates an empty Git repository or reinitializes an existing one. It has created a repository in the current directory i.e. **gitlocalrepo/gittest**

Next type:

```
A@COMP /e/gitlocalrepo/gittest
$ ls -la
total 0
drwxr-xr-x    3 A        Administ        0 Jan 28 16:47 .
drwxr-xr-x    4 A        Administ        0 Jan 28 16:44 ..
drwxr-xr-x    6 A        Administ        0 Jan 28 16:47 .git
A@COMP /e/gitlocalrepo/gittest
$
```

Git has created a **.git** subdirectory in your directory **gitlocalrepo/gittest** that will hold your changes locally (in compressed form). The new local repo will not contain anything yet.

Making changes

Next, the following commands create an empty **README** and **donottrack.log** files at the root of my project:

```
A@COMP /e/gitlocalrepo/gittest
$ touch README
A@COMP /e/gitlocalrepo/gittest
$ touch donottrack.log
A@COMP /e/gitlocalrepo/gittest
$
```

The above **touch** command creates the two files **README** and **donottrack.log,** if the files do not already exist. If a file already exists, the accessed / modification time is updated for that file.

Ignoring files you do not want to track

Let us assume that you do not want to track the file **donottrack.log**. To achieve this, type:

```
A@COMP /e/gitlocalrepo/gittest
$ touch .gitignore
A@COMP /e/gitlocalrepo/gittest
$
```

This creates a **.gitignore** file at the root of the project. Next edit the **.gitignore** file and type some patterns. For example ***.log** will ignore all log files in any subdirectory also. This file helps us to un-track all .log files in my project. The **.gitignore** file stays with the project. So it will be checked-in and anyone who clones it will get the **.gitignore** file (we shall talk later about check-in and cloning).

For **Mac OS X**, it is recommended to place in the **.gitignore** file the following pattern:

```
.DS_Store
```

The **.DS_Store** files are always ignored. On Mac, the **.DS_Store** files are created by the system (always - those files are created after some time in any folder on the files system store the settings of the Finder Windows).

Also, the file **e:/gitlocalrepo/gittest/.git/info/exclude** also contains certain excludes. This file is your own personal copy and it is not going to be checked in. You are going to use it if there are certain files you want to be excluded from the repo. **Mac users** need to declare it in the .gitconfig file.

```
git config --global core.excludesfile ~/.gitexcludes
```

Staging

Note: In Git, you "stage" things before you commit them. You do this with the **git add** command (for example: **git add README**). This adds specific content to the ‘stage’.

To *stage all changes* and new files, type:

```
A@COMP /e/gitlocalrepo/gittest
$ git add .
A@COMP /e/gitlocalrepo/gittest
$
```

The **`git add .`** command will take the working directory and all sub-directories and every single file, i.e. it adds all content to the ‘stage’ (this snapshot is now stored in a temporary staging area which Git calls the "index").

If you make any changes to a file after staging (but before committing), you’ll need to **`git add`** the file again.

You can see what has and has not been staged and/or committed by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   .gitignore
#       new file:   README
#
A@COMP /e/gitlocalrepo/gittest
$
```

Note: You can remove a file from staging with **`git reset HEAD <filename>`**.

Commit

When you get your changes just the way you want them added to the current revision, then you need to commit that revision to your local repository (this permanently stores the contents of the index in the repository).

We already have something staged and ready and it is time to *commit* it. We do it as follows:

```
git commit -m 'brief description of commit'
```

Whenever you make a significant change while working on a project, get into the habit of committing it, even if it is only a line or two. It is quick and easy, and you will thank yourself later. Replace '**brief description of commit**' with a comment like **'First commit'**

```
A@COMP /e/gitlocalrepo/gittest
$ git commit -m 'First commit'
[master (root-commit)]: created 9311786: "First commit"
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore
 create mode 100644 README
A@COMP /e/gitlocalrepo/gittest
$
```

To check which files have been committed to the local repo, type:

```
A@COMP /e/gitlocalrepo/gittest
$ git ls-files
.gitignore
README
A@COMP /e/gitlocalrepo/gittest
$
```

Note: To *revert* the most recent commit, type **git revert HEAD**. This will create a new commit which undoes the change in HEAD. You will be given a chance to edit the commit message for the new commit. Also, we *cannot* revert the first commit of a file.

Push changes to your repository on GitHub

Refer to the following url for details of the **git remote** command:
http://www.kernel.org/pub/software/scm/git/docs/git-remote.html

Type in the following:

```
A@COMP /e/gitlocalrepo/gittest
$ git remote add origin git@github.com:RLGGHC/gittest.git
A@COMP /e/gitlocalrepo/gittest
$
```

(The *above* command needs to be executed only once in case you are not going to change the remote repository. **Note**: Substitute **RLGGHC** with your **GitHubName** and **gittest** with *your* repository name on GitHub).

You can also confirm that this command has successfully executed by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git remote show origin
* remote origin
  URL: git@github.com:RLGGHC/gittest.git
A@COMP /e/gitlocalrepo/gittest
$
```

Please note that: When we type:

```
$ git remote show origin
```

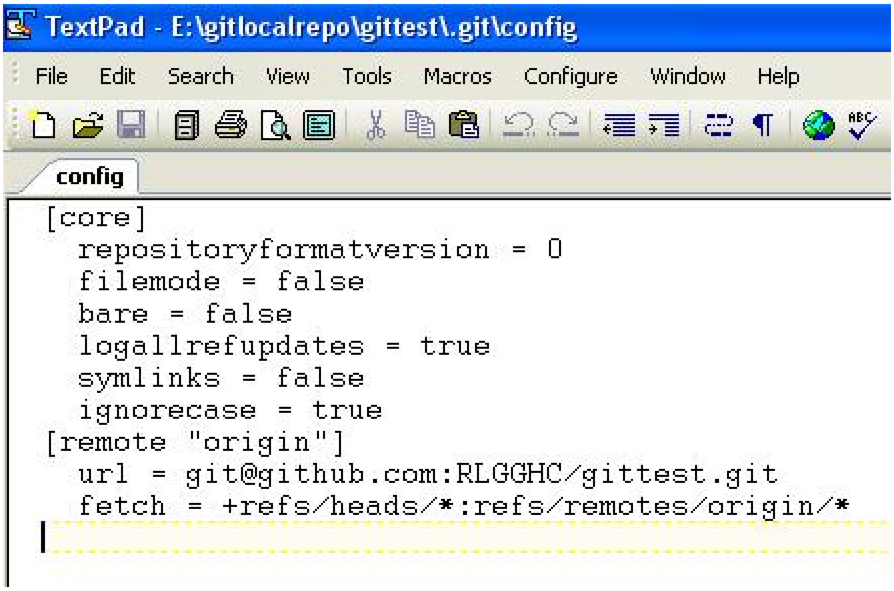
You may be requested to enter a passphrase (in case you use the passphrase), because the command queries the remote repository.

On the contrary, if you type:

```
$ git remote show -n origin
```

You would not be asked for your passphrase, because in this case the command queries the local cache.

You can also open the **config** file in the **e:/gitlocalrepo/gittest/.git** folder to see the remote being set, as shown in the following screenshot:


TextPad - E:\gitlocalrepo\gittest\.git\config
File Edit Search View Tools Macros Configure Window Help
config

```
[core]
  repositoryformatversion = 0
  filemode = false
  bare = false
  logallrefupdates = true
  symlinks = false
  ignorecase = true
[remote "origin"]
  url = git@github.com:RLGGHC/gittest.git
  fetch = +refs/heads/*:refs/remotes/origin/*
```

Mac OS X users would have to use Cmd-Shift-G on the folder name in the Finder and type .git in the dialog window to get access to the Git folder.

Once your changes are committed to your local repository, you need to *push* them to the remote repository for others to get at. To do that, you need to execute **git push**, which will push all the changes from your local repository up to the remote repository.

Git push takes several arguments:
git push <repository> <branch>

In this case, we want to push changes back to the original repository, which is aliased as *origin*, to the *master* branch.

Now type:

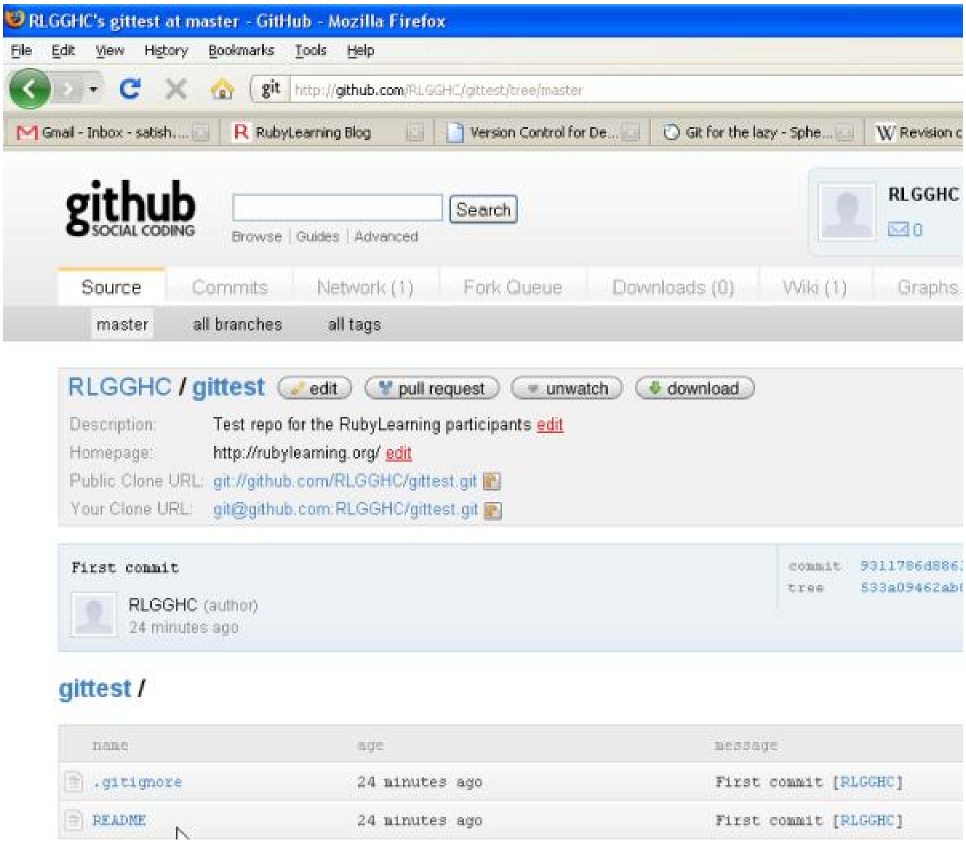
```
A@COMP /e/gitlocalrepo/gittest
$ git push origin master
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 262 bytes, done.
Total 4 (delta 0), reused 0 (delta 0)
To git@github.com:RLGGHC/gittest.git
 * [new branch]      master -> master
A@COMP /e/gitlocalrepo/gittest
$
```

This command tells Git that you have a remote repository there and the **`git push`** command uploads your (committed) changes to Github.

Note: In case Git tells you that it failed to push some references, then refer to this blog post –

http://edspencer.net/2008/04/when-git-tells-you-it-failed-to-push.html

After pushing, here's how our repo on GitHub looks:


RLGGHC's gittest at master - GitHub - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://github.com/RLGGHC/gittest/tree/master
Gmail - Inbox - satish....
RubyLearning Blog
Version Control for De...
Git for the lazy - Sphe...
Revision c
github SOCIAL CODING
Search
Browse | Guides | Advanced
RLGGHC
0
Source
Commits
Network (1)
Fork Queue
Downloads (0)
Wiki (1)
Graphs
master
all branches
all tags
RLGGHC / gittest
edit
pull request
unwatch
download

Description: Test repo for the RubyLearning participants edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/gittest.git
Your Clone URL: git@github.com:RLGGHC/gittest.git

First commit
RLGGHC (author)
24 minutes ago

commit 9311786d886
tree 533a09462ab

gittest /

name	age	message
.gitignore	24 minutes ago	First commit [RLGGHC]
README	24 minutes ago	First commit [RLGGHC]

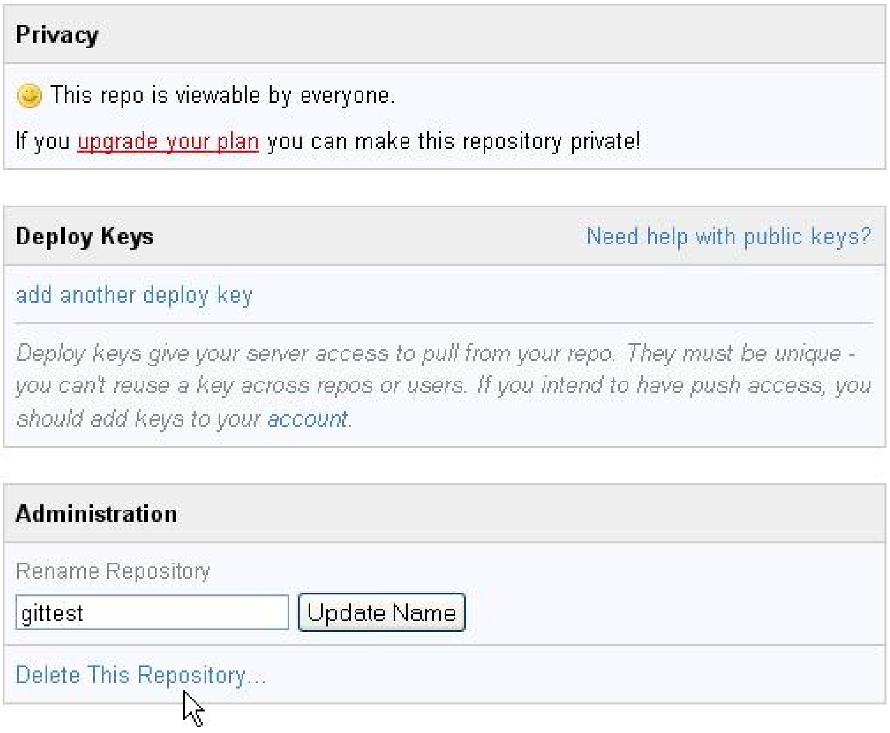
Note: Whenever you have committed changes that you wish to put into your Github repository in the future, just type:

```
$ git push origin master
```

Deleting a repository

You need not delete the repository that you just created. I am just showing you how a repository can be deleted, if required.

Click on the "Admin" link of the repo you want to delete and scroll down to the bottom of the screen. Click on "Delete This Repository…" as shown in the following screenshot.


Privacy

This repo is viewable by everyone.

If you upgrade your plan you can make this repository private!

Deploy Keys Need help with public keys?

add another deploy key

Deploy keys give your server access to pull from your repo. They must be unique - you can't reuse a key across repos or users. If you intend to have push access, you should add keys to your account.

Administration

Rename Repository

gittest [Update Name]

Delete This Repository…

You will get a message "**Once you delete a repository, there is no going back. Please be certain.**" Click on the "Delete repository" button. Confirm on the OK button and that's it. The repo has been deleted.

Exercise 2

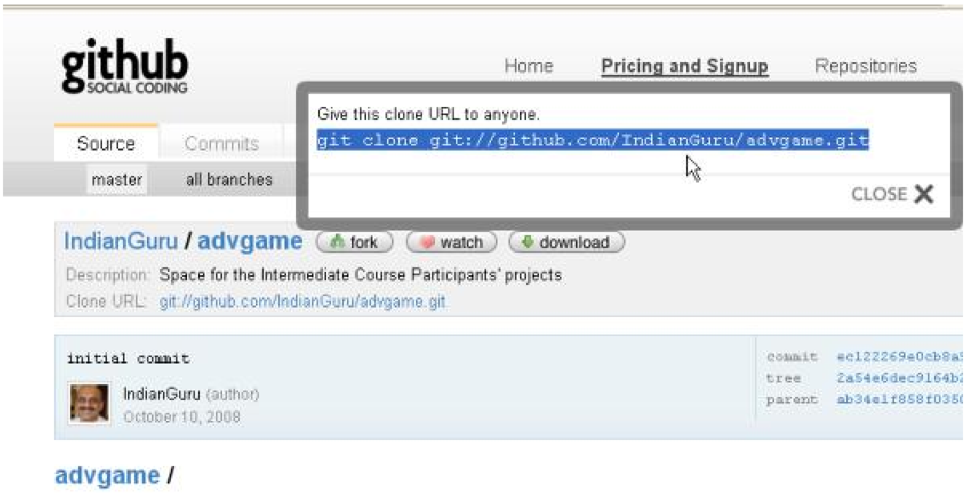
Post the URL (http://) of *your* newly created GitHub repo in the relevant thread at the Course Technical forum.

Cloning a public project on GitHub to your local repo

To get a working copy of an open source project on GitHub, you need to *clone* a remote repository to your local machine. Cloning creates the repository and checks out the latest version, which is referred to as **HEAD**.

You can clone any of the public repos on GitHub.

Let's say we want to clone the project (this no longer exists) here – http://github.com/IndianGuru/advgame/tree/master



Give this clone URL to anyone.

```
git clone git://github.com/IndianGuru/advgame.git
```

IndianGuru / advgame

Description: Space for the Intermediate Course Participants' projects

Clone URL: git://github.com/IndianGuru/advgame.git

```
initial commit
```

IndianGuru (author)
October 10, 2008

```
commit  ec122269e0cb8a
tree    2a54e6dec9164b
parent  ab34e1f858f035
```

The public clone url which you can use is:

```
git://github.com/IndianGuru/advgame.git
```

An **important** thing to remember is:

The clone url -
`git clone git://github.com/project_name/repo_name.git`
is used by anyone (public) and they can only *pull* objects from the repository.

Whereas, the clone url –
`git clone git@github.com:project_name/repo_name.git`
can be used to *push* and *pull* from the repository (over SSH) by the owner of the project and collaborators.

Now, with the bash shell still open, you can download a copy of the above mentioned project from GitHub to your local repo at **`gitlocalrepo`**.

(**Remember**: We need to be inside the **`gitlocalrepo`** folder).

You clone it locally by typing the following in the bash shell:

```
A@COMP /e/gitlocalrepo
$ git clone git://github.com/IndianGuru/advgame.git
Initialized empty Git repository in E:/gitlocalrepo/advgame/.git/
remote: Counting objects: 12, done.←[K
remote: Compressing objects: 100% (12/12), done.←[K
remote: Total 12 (delta 2), reused 0 (delta 0)←[K
Receiving objects: 100% (12/12), 10.37 KiB, done.
Resolving deltas: 100% (2/2), done.
A@COMP /e/gitlocalrepo
$
```

Note that the command is **`git clone`**.
The location you are cloning from is –

`git://github.com/IndianGuru/advgame.git`

Congratulations, you just cloned your first repository. The **`clone`** command sets up a few convenience items for you; it keeps the address of the original repository, and aliases it as *origin*, so you can easily send back changes (if you have authorization) to the remote repository.

You will now have a folder **`advgame`** created under your folder **`gitlocalrepo`**. Type the following in the bash shell:

```
$ ls -la
total 0
drwxr-xr-x    3 A       Administ        0 Jan 28 16:24 .
drwxr-xr-x   38 A       Administ        0 Jan 28 16:15 ..
drwxr-xr-x    3 A       Administ        0 Jan 28 16:24 advgame
A@COMP /e/gitlocalrepo
```

```
$
```

Next change the folder to **advgame** and list out the contents:

```
$ cd advgame/
A@COMP /e/gitlocalrepo/advgame
$ ls
README      advgame_spec.rb  test_advgame.rb
advgame.rb  loogink.png      test_advgame_on_shoes.rb
A@COMP /e/gitlocalrepo/advgame
$
```

You can see the full history of this project, by typing:

```
A@COMP /e/gitlocalrepo/advgame
$ git log
commit ec122269e0cb8a9db3f1f7b298ee63d9c17f7892
Author: IndianGuru <satish.talim@gmail.com>
Date:   Fri Oct 10 10:47:02 2008 +0530
    initial commit
commit ab34e1f858f03508d4046228370b7e4c3267406f
Author: IndianGuru <satish.talim@gmail.com>
Date:   Fri Sep 26 07:54:33 2008 +0530
    initial commit
commit 748443155bd50a9e5508155e2e74dd2c15bd43f1
Author: IndianGuru <satish.talim@gmail.com>
Date:   Sun Sep 21 06:47:18 2008 +0530
    initial commit
A@COMP /e/gitlocalrepo/advgame
$
```

This log has amongst other things the email addresses of the people involved in the project.

A detailed note on `git log` is available in Appendix A.

Switch back to the **gitlocalrepo** folder by typing:

```
$ cd ..
```

Please note that we are not going to use this *clone* anywhere in our course.

Note: You can get all the info on Git commands from the Git manpage:

http://www.kernel.org/pub/software/scm/git/docs/

Using gitk tool

Git comes along with a gitk – a tree visualizer tool. The gitk application allows you to navigate through the tree of changes, view diffs, search old revisions, and more.

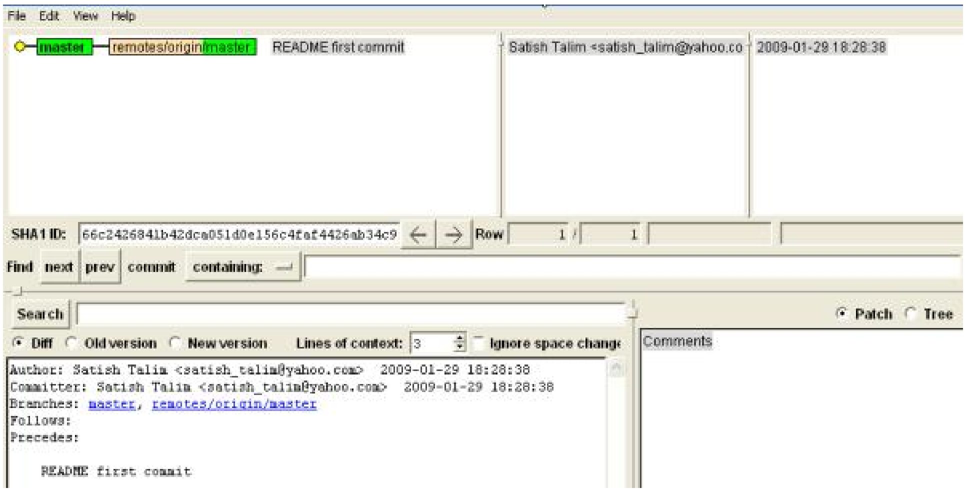
(**Remember**: We need to be inside the **`gitlocalrepo/gittest`** folder).

```
A@COMP /e/gitlocalrepo
$ cd gittest/
A@COMP /e/gitlocalrepo/gittest
$
```

Next type:

```
$ gitk --all
```

This loads a screen as shown:


File Edit View Help
master
remotes/origin/master
SHA1 ID:
Row
Find next prev commit containing:
Search
Patch Tree
Diff Old version New version Lines of context: Ignore space change
Comments

```
README first commit    Satish Talim <satish_talim@yahoo.co    2009-01-29 18:28:38
SHA1 ID: 66c2426841b42dca051d0e156c4faf4426ab34c9    Row 1 / 1
Lines of context: 3

Author: Satish Talim <satish_talim@yahoo.com>  2009-01-29 18:28:38
Committer: Satish Talim <satish_talim@yahoo.com>  2009-01-29 18:28:38
Branches: master, remotes/origin/master
Follows:
Precedes:

    README first commit
```

Note by Michèle Garoche: To increase the font size just use cmd-= on the Mac (probably ctrl-= on other systems).

(Maybe, that it does not work on non English keyboard on some system - it was a bug not sure if it has been corrected -, if you use the = sign in the letter pad, but it works using the = sign on the numeric pad. It works on the Mac both ways).

Another way for Mac users is:

```
mate ~/.gitk
```

Edit **`.gitk`** file in your home directory and add –

```
set mainfont {Monaco 12}
set textfont {Monaco 12}
set uifont {"Monaco Bold" 12}
```

Finally close this window, when you are thro' looking at.

Forking a repository

Next, let us see an example of forking a public repository. Ensure that you are logged into your GitHub account.

We shall fork our mentor Satoshi Asakawa's repository here –

http://github.com/ashbb/ruby_learning_participants/tree/master

Go to the above url and click on the "fork" button. This creates a copy or fork of **`ashbb/ruby_learning_participants`**. It copies all the information over but you get a different "Your Clone URL" as in –

git@github.com:RLGGHC/ruby_learning_participants.git

(For your clone url, you will see your account name in place of **`RLGGHC`**).

Whenever you fork a repository, you automatically watch that repository in your GitHub account. However, fork does not automatically create a local folder.

Now you need to *clone* the fork.

Make sure you use the "Your Clone URL" and **not** the "Public Clone URL".

Remember you should be in your **`gitlocalrepo`** folder.

Type:

```
A@COMP /e/gitlocalrepo
$ git clone git@github.com:RLGGHC/ruby_learning_participants.git
Initialized empty Git repository in
e:/gitlocalrepo/ruby_learning_participants/.
git/
remote: Counting objects: 67, done.←[K
remote: Compressing objects: 100% (67/67), done.←[K
remote: Total 67 (delta 32), reused 0 (delta 0)←[KiB/s
Receiving objects: 100% (67/67), 409.18 KiB | 16 KiB/s, done.
Resolving deltas: 100% (32/32), done.
A@COMP /e/gitlocalrepo
$
```

Remember that your clone url will be different.

Once the clone is complete your repo will have a remote named "origin" that points to your fork on github. Don't let the name confuse you, this **does not** point to the original repo you forked from.

Let's confirm that, by typing:

```
A@COMP /e/gitlocalrepo/ruby_learning_participants
$ git remote show origin
* remote origin
  URL: git@github.com:RLGGHC/ruby_learning_participants.git
  Remote branch merged with 'git pull' while on branch master
    master
  Tracked remote branch
    master
```

```
A@COMP /e/gitlocalrepo/ruby_learning_participants
$
```

Pushing our changes

Now that we've got our fork, we need to make a few changes and commit them locally.

I added some text at the end of the file README.txt, namely:

```
Cheers,
ashbb (Satoshi Asakawa)
```

I then typed:

```
A@COMP /e/gitlocalrepo/ruby_learning_participants
$ git add README.txt
A@COMP /e/gitlocalrepo/ruby_learning_participants
$ git commit -m 'added full name in README.txt'
[master]: created 3a29ccd: "added full name in README.txt"
 1 files changed, 4 insertions(+), 4 deletions(-)
A@COMP /e/gitlocalrepo/ruby_learning_participants
$
```

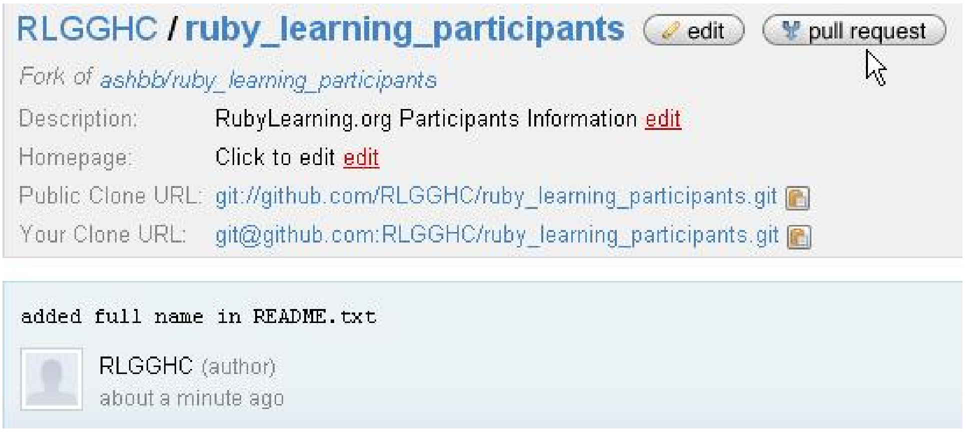
Once you've done this, it's time to push your updated branch.

```
A@COMP /e/gitlocalrepo/ruby_learning_participants
$ git push origin master
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 443 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@github.com:RLGGHC/ruby_learning_participants.git
   20bf241..3a29ccd  master -> master
A@COMP /e/gitlocalrepo/ruby_learning_participants
$
```

Sending a pull request

After you've pushed your commit(s) you need to inform the project owner (ashbb) of the changes so they can pull them into their repo.

From your project's page, click the “pull request” button (as shown in the next screenshot).



RLGGHC / ruby_learning_participants

Fork of ashbb/ruby_learning_participants

Description: RubyLearning.org Participants Information edit

Homepage: Click to edit edit

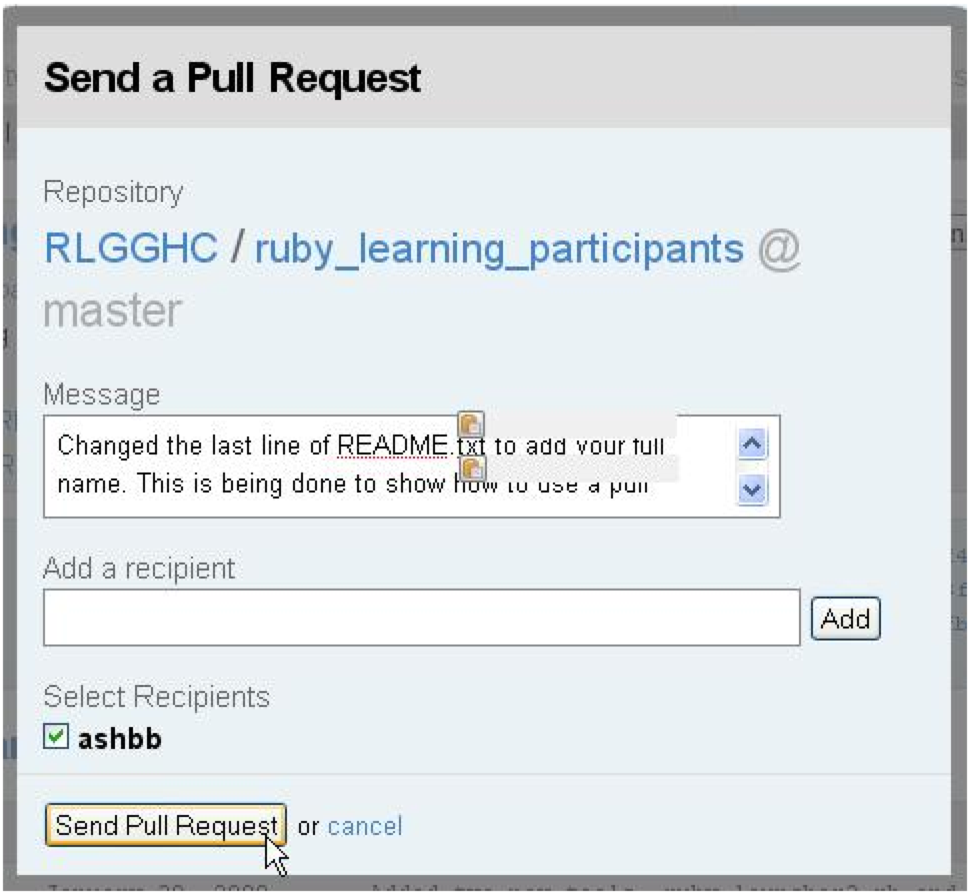
Public Clone URL: git://github.com/RLGGHC/ruby_learning_participants.git

Your Clone URL: git@github.com:RLGGHC/ruby_learning_participants.git

added full name in README.txt

RLGGHC (author)
about a minute ago

Fill in a note and pick who to send the request to, as can be seen in the next screenshot –


Send a Pull Request

Repository

RLGGHC / ruby_learning_participants @ master

Message

Changed the last line of README.txt to add your full name. This is being done to show how to use a pull

Add a recipient

Select Recipients

✓ ashbb

Please note: It is okay to send **ashbb** your pull request. However, for obvious reasons, it'll be ignored.

Keeping track of original repository

To help you keep track of the original repo **ashbb/ruby_learning_participants**, we will add another remote named "**ashbbrlp**". Type:

```
A@COMP /e/gitlocalrepo
$ cd ruby_learning_participants/
A@COMP /e/gitlocalrepo/ruby_learning_participants
$ git remote add ashbbrlp
git://github.com/ashbb/ruby_learning_participants.git
A@COMP /e/gitlocalrepo/ruby_learning_participants
$ git remote show ashbbrlp
* remote ashbbrlp
  URL: git://github.com/ashbb/ruby_learning_participants.git
  New remote branch (next fetch will store in remotes/ashbbrlp)
    master
A@COMP /e/gitlocalrepo/ruby_learning_participants
$ git fetch ashbbrlp
From git://github.com/ashbb/ruby_learning_participants
 * [new branch]      master     -> ashbbrlp/master
A@COMP /e/gitlocalrepo/ruby_learning_participants
$
```

Note that we used the public clone URL for **ashbbrlp**, so we can't push changes directly to it. We probably don't have permission to do that anyway, which is why we're creating a fork in the first place.

Also, **git fetch** fetches all the objects from **ashbbrlp** and stores them locally.

Pulling in ashbbrlp changes

Some time has passed, the **ashbbrlp** repo has changed and you want to update your fork before you submit a new patch.

To update your local repository and working copy to the latest revision committed to the remote repository – **git://github.com/ashbb/ruby_learning_participants.git**

You need to execute **`git pull`**. This pulls all of the change-sets down from the remote repository and merges them with your current changes (if there are any).

```
$ git pull ashbbrlp master
```

This command does two things, **`git fetch ashbbrlp master`** to update your local tracking branch, then **`git merge ashbbrlp/master`** to bring the changes into your currently checked out branch. If you have local commits that are not in the **`ashbbrlp`** branch, a normal merge will occur. If your local commits are in the **`ashbbrlp`** branch, a fast-forward merge will be done, moving your local branch to the same commit as **`ashbbrlp/master`**. You may have to resolve merge conflicts if there are any.

Now that your local branch has been updated, you can commit, push, and send a pull request.

You may wish to do the fetch and merge manually, instead of letting **`git pull`** do it for you. This can sometimes help avoid headaches caused by mysterious merge conflicts.

Exercise 3

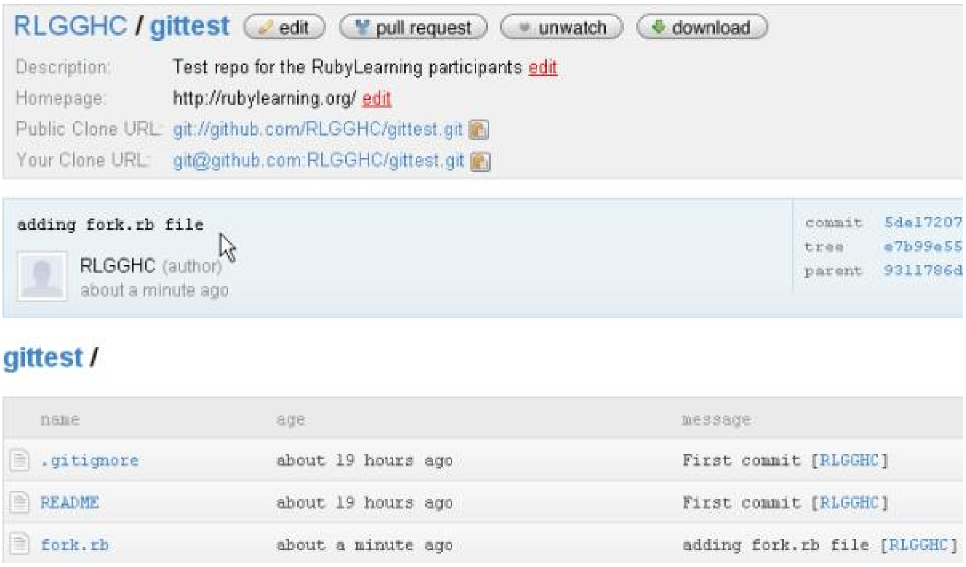
I have added a simple, single line program called **`fork.rb`** to my **`gittest`** public repository here –

http://github.com/RLGGHC/gittest/tree/master

The program **`fork.rb`** contains, to begin with, one line of code –

```
puts "This line added by project RLGGHC"
```

Please see screenshot -


RLGGHC / gittest

Description: Test repo for the RubyLearning participants edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/gittest.git
Your Clone URL: git@github.com:RLGGHC/gittest.git

adding fork.rb file
RLGGHC (author)
about a minute ago

commit 5de17207
tree e7b99e55
parent 9311786d

gittest /

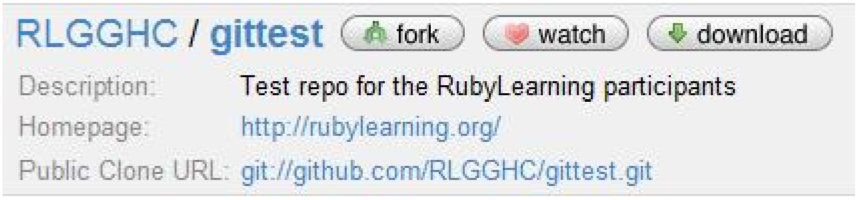
name	age	message
.gitignore	about 19 hours ago	First commit [RLGGHC]
README	about 19 hours ago	First commit [RLGGHC]
fork.rb	about a minute ago	adding fork.rb file [RLGGHC]

Just for your information, this is how I added the **fork.rb** to my GitHub repo (copy **fork.rb** to local **gittest** folder, *stage*, *commit* and *push*):

```
A@COMP /e/gitlocalrepo/gittest
$ git add fork.rb
A@COMP /e/gitlocalrepo/gittest
$ git commit -m 'adding fork.rb file'
[master]: created 5de1720: "adding fork.rb file"
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 fork.rb
A@COMP /e/gitlocalrepo/gittest
$ git push origin master
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 357 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:RLGGHC/gittest.git
   9311786..5de1720  master -> master
A@COMP /e/gitlocalrepo/gittest
$
```

You now fork my **gittest** repo at (click on Fork button) –

http://github.com/RLGGHC/gittest/tree/master



Description: Test repo for the RubyLearning participants
Homepage: http://rubylearning.org/
Public Clone URL: git://github.com/RLGGHC/gittest.git

Note: If you already have a repo by the name of **`gittest`**, then you will not see the "Fork" button on my **`gittest`** repo at –
`http://github.com/RLGGHC/gittest/tree/master`
Please delete your repo **`gitttest`** before proceeding.

After forking, **`clone`** it to your local repo (make sure you use the "Your Clone URL" and **not** the "Public Clone URL").

Next, modify your local copy of **`fork.rb`** file and add one line of code to it as:
`puts "Added this line by name of participant"`

substitute "name of participant" with your actual name.

Next **`add`** and **`commit`** to your local repository and then **`push`** it to *your* forked **`gittest`** repository.

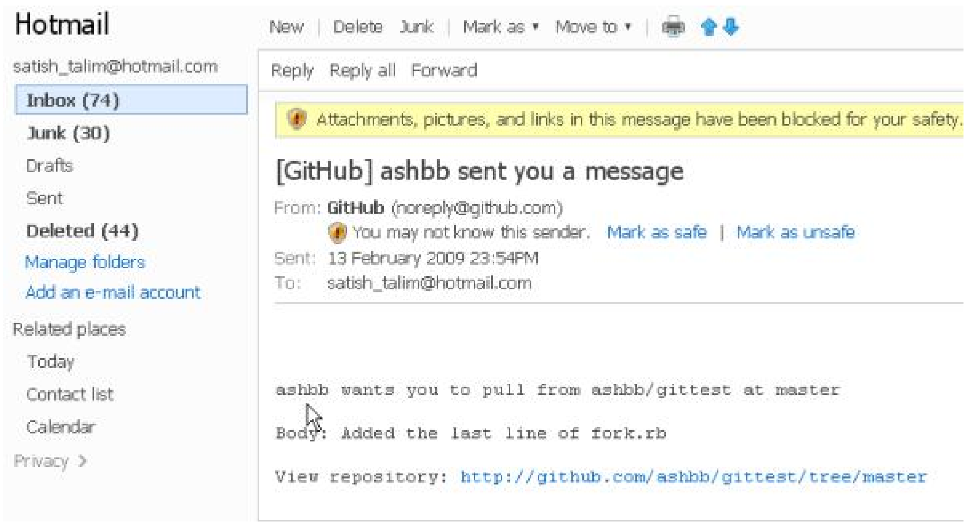
Finally, inform the project owner (i.e. me) of the changes so that I can pull them into the repo – using a *pull* request.

Please note: It is okay to send me your pull requests. However, for obvious reasons, it'll be ignored.

Merging changes from a pull request

Let us assume that **ashbb** has made a pull request to me for my project at –
`http://github.com/RLGGHC`

I shall receive an email at my satish_talim@hotmail.com a/c, as shown:



Attachments, pictures, and links in this message have been blocked for your safety.

[GitHub] ashbb sent you a message

From: GitHub (noreply@github.com)
You may not know this sender. Mark as safe | Mark as unsafe
Sent: 13 February 2009 23:54PM
To: satish_talim@hotmail.com

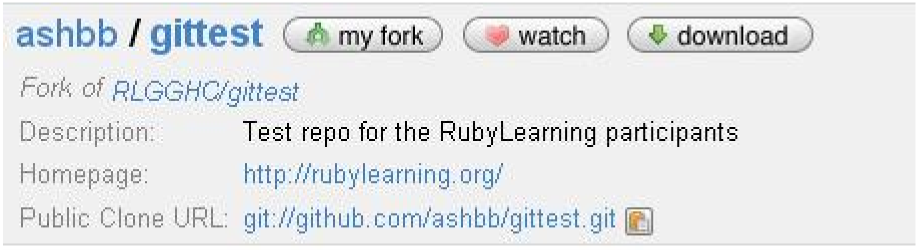
```
ashbb wants you to pull from ashbb/gittest at master

Body: Added the last line of fork.rb

View repository: http://github.com/ashbb/gittest/tree/master
```

Let me log into my account at – http://github.com/RLGGHC

go to the url in the email, namely – http://github.com/ashbb/gittest/tree/master

and check out the commit list at – http://github.com/ashbb/gittest/commits/master

In the next screenshot, click on “Added one line”

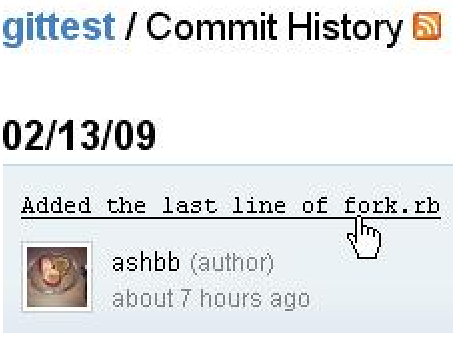


Fork of RLGGHC/gittest

Description: Test repo for the RubyLearning participants

Homepage: http://rubylearning.org/

Public Clone URL: git://github.com/ashbb/gittest.git

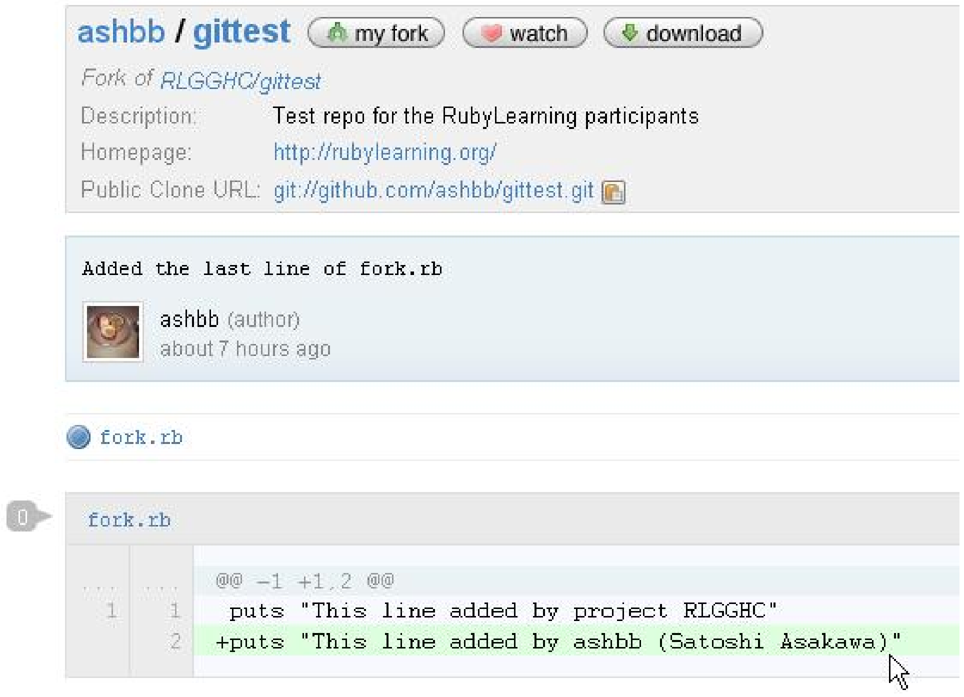


02/13/09

Added the last line of fork.rb

ashbb (author)
about 7 hours ago

This will take you to the next screenshot, where you will see that ashbb has added one line of code in **`fork.rb`**



Fork of RLGGHC/gittest
Description: Test repo for the RubyLearning participants
Homepage: http://rubylearning.org/
Public Clone URL: git://github.com/ashbb/gittest.git

Added the last line of fork.rb

ashbb (author)
about 7 hours ago

fork.rb

fork.rb

```
@@ -1 +1,2 @@
 puts "This line added by project RLGGHC"
+puts "This line added by ashbb (Satoshi Asakawa)"
```

Now let us pull the change made by ashbb, i.e. let us add ashbb's remote repository by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git remote add ashbb git://github.com/ashbb/gittest.git
A@COMP /e/gitlocalrepo/gittest
$
```

Let's confirm this by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git remote show ashbb
* remote ashbb
  URL: git://github.com/ashbb/gittest.git
  New remote branch (next fetch will store in remotes/ashbb)
    master
A@COMP /e/gitlocalrepo/gittest
$
```

Now let us fetch all the objects from ashbb and store them locally, by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git fetch ashbb
remote: Counting objects: 5, done.←[K
remote: Compressing objects: 100% (3/3), done.←[K
remote: Total 3 (delta 1), reused 0 (delta 0)←[K
Unpacking objects: 100% (3/3), done.
From git://github.com/ashbb/gittest
 * [new branch]      master     -> ashbb/master
A@COMP /e/gitlocalrepo/gittest
$
```

Let's see the log, by typing:

```
$ git log ashbb/master
commit 7428b16ff87a75c7b73b5234157f699db1e5bcf7
Author: ashbb <ashbbb@gmail.com>
Date:   Sat Feb 14 03:20:34 2009 +0900
    Added the last line of fork.rb
commit 5de17207fb95580a47abdc46dfa5933ee89bc5e0
Author: Satish Talim <satish_talim@hotmail.com>
Date:   Thu Feb 12 13:39:06 2009 +0530
    adding fork.rb file
commit 9311786d88613f0abd28e53280e554d2dbd89ff2
Author: Satish Talim <satish_talim@hotmail.com>
Date:   Wed Feb 11 18:06:44 2009 +0530
    First commit
A@COMP /e/gitlocalrepo/gittest
$
```

Let's do a merge on my local Git database, by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git merge ashbb/master
Updating 5de1720..7428b16
Fast forward
 fork.rb |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
A@COMP /e/gitlocalrepo/gittest
$
```

At this stage, you may or may not get a conflict in **`fork.rb`** file and you would need to resolve the conflict manually. If there's a conflict, open **`fork.rb`**, and you would see the following:

```
puts "This line added by project RLGGHC"
<<<<<<< HEAD:fork.rb
=======
puts "This line added by ashbb (Satoshi Asakawa)"
>>>>>>> ashbb/master:fork.rb
```

In such a case, edit the **fork.rb** file and delete the extra lines manually. The **fork.rb** file now looks like:

```
puts "This line added by project RLGGHC"
puts "This line added by ashbb (Satoshi Asakawa)"
```

Note however that I did not get a conflict when I executed the command – **git merge ashbb/master**

Next we add the **fork.rb** file manually by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git add fork.rb
A@COMP /e/gitlocalrepo/gittest
$
```

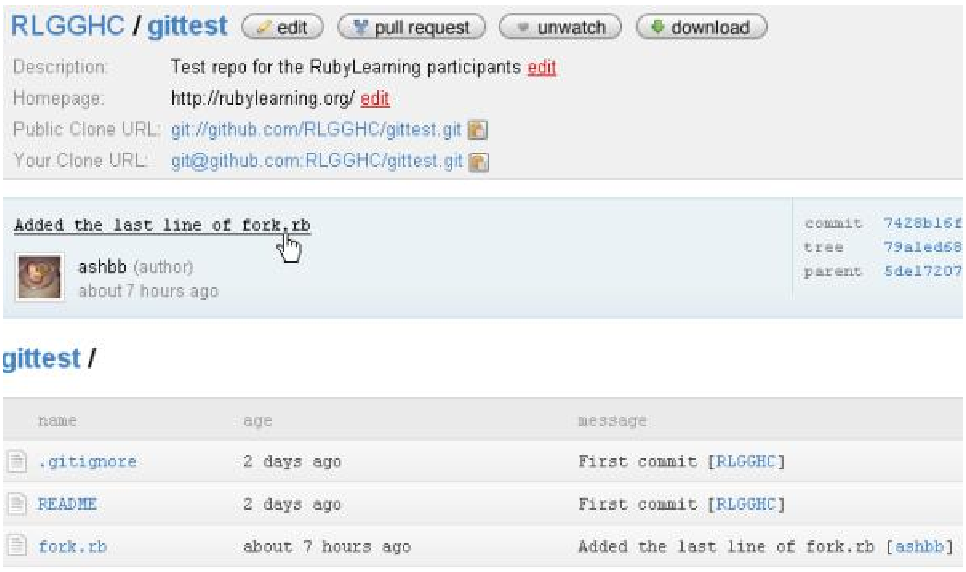
Let us commit our changes by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git commit -m 'fork.rb commit with ashbb changes'
# On branch master
nothing to commit (working directory clean)
A@COMP /e/gitlocalrepo/gittest
$
```

Finally let us push the merged changes to my master by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git push origin master
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 324 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@github.com:RLGGHC/gittest.git
   5de1720..7428b16  master -> master
A@COMP /e/gitlocalrepo/gittest
$
```

This is the screenshot:



Description:	Test repo for the RubyLearning participants edit
Homepage:	http://rubylearning.org/ edit
Public Clone URL:	git://github.com/RLGGHC/gittest.git
Your Clone URL:	git@github.com:RLGGHC/gittest.git

Added the last line of fork.rb

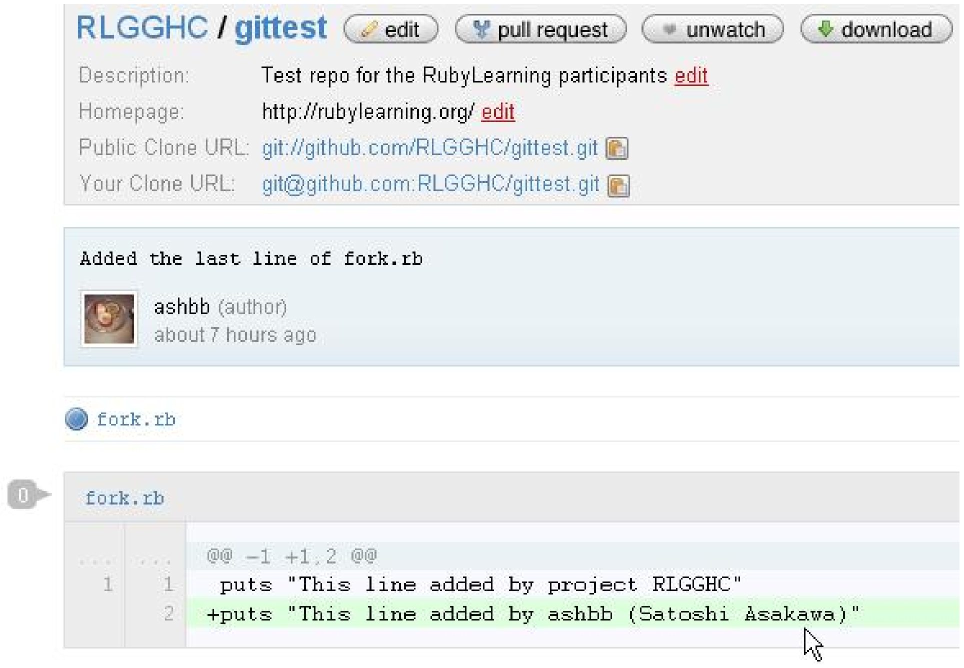
ashbb (author)
about 7 hours ago

```
commit  7428b16f
tree    79a1ed68
parent  5de17207
```

gittest /

name	age	message
.gitignore	2 days ago	First commit [RLGGHC]
README	2 days ago	First commit [RLGGHC]
fork.rb	about 7 hours ago	Added the last line of fork.rb [ashbb]

Clicking on “Added the last line at fork.rb” shows (see screenshot):



Description: Test repo for the RubyLearning participants edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/gittest.git
Your Clone URL: git@github.com:RLGGHC/gittest.git

Added the last line of fork.rb

ashbb (author)
about 7 hours ago

fork.rb

fork.rb

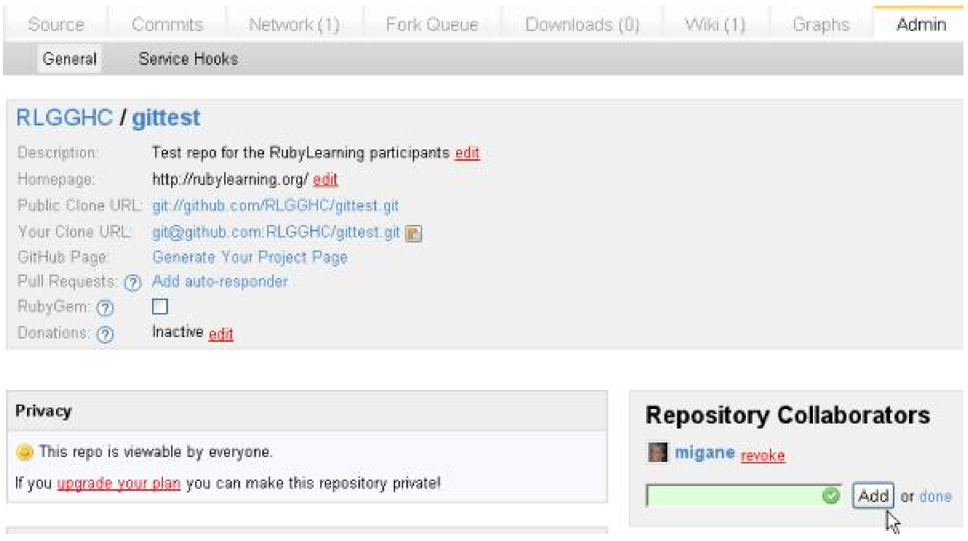
```
@@ -1 +1,2 @@
 puts "This line added by project RLGGHC"
+puts "This line added by ashbb (Satoshi Asakawa)"
```

Exercise 4

As an additional exercise, you can fork, pull, push changes of one of the course participants' repo. The participant whose repo you have forked, could in turn merge the changes. This participant can return the favour by doing exactly the same, as you did.

Managing collaborators on GitHub

Open your repositories admin page (**`gittest`** in my case on GitHub). To add a collaborator, in the repo admin page under "Repository Collaborators" click on the link "Add another collaborator", type in your collaborator username and click the "Add" button. To remove a collaborator, open the repo admin page, find the collaborator you wish to remove, and click "revoke". I have added Michèle Garoche (migane) as collaborator in my **`gittest`** repo on GitHub. This is shown in the screenshot:



RLGGHC / gittest

Description: Test repo for the RubyLearning participants edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/gittest.git
Your Clone URL: git@github.com:RLGGHC/gittest.git
GitHub Page: Generate Your Project Page
Pull Requests: Add auto-responder
RubyGem: ☐
Donations: Inactive edit

Privacy

This repo is viewable by everyone.

If you upgrade your plan you can make this repository private!

Repository Collaborators

migane revoke

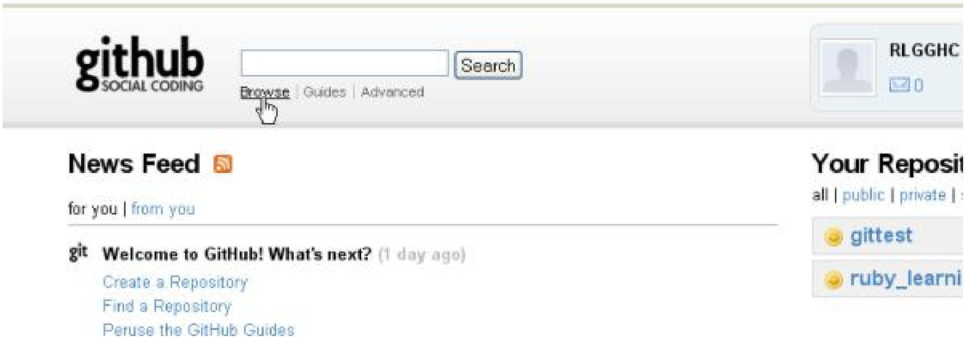
You add as collaborators only the users that you trust because they will have push privileges to your repository. If I am a collaborator in a project then there is no need to fork it. I just have to clone it and push my changes.

Collaboration between collaborators

Now when say **`migane`** logs into her GitHub account, she will be able to see in her news feed –"RLGGHC added you to **`gittest`**". If **`migane`** clicks on **`gittest`** she will be able to see the **`gittest`** repo. Now, **`migane`** can *clone* the **`gittest`** repo and make changes locally and push her changes to my **`gittest`** repo.

Watch projects and people

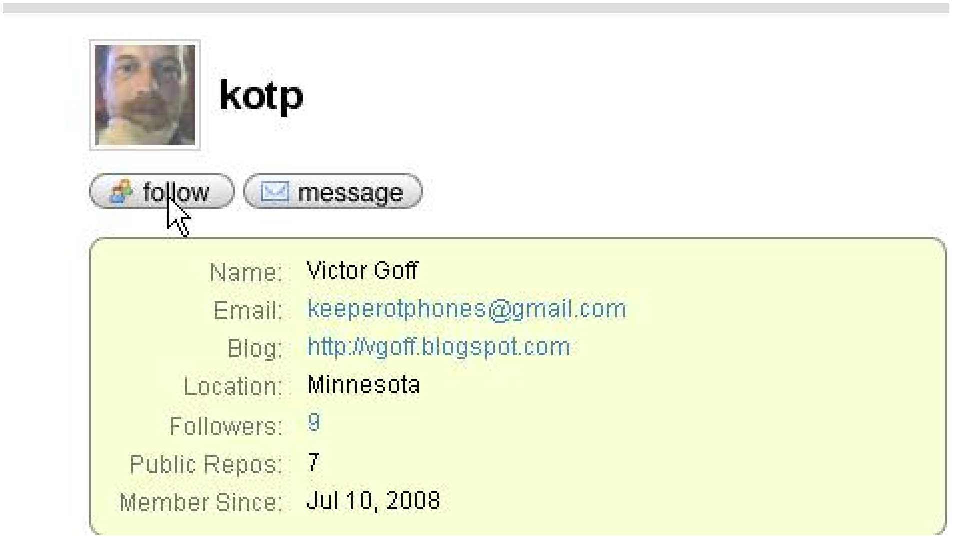
Assuming that you are still logged-in to your GitHub account, click on the “dashboard” link at the top right side of your GitHub account page. A new page will load. Click on the “Browse” link as shown below:



You will be taken to a screen which has links to “Recent”, “Popular Forked” and “Popular Watched” repos. Select and click on the repo you are interested in. You will be taken to that repo screen. Next, click on the “watch” button. This repo will now appear in your “Watched Repositories”. Once you start watching a repo, then anytime anything is committed to this or the wiki is changed or something, you will be able to see that in your personal News Feed (see screenshot above) or you will be notified if you have subscribed to this RSS feed.

Let us go to the following url –
`http://github.com/kotp`

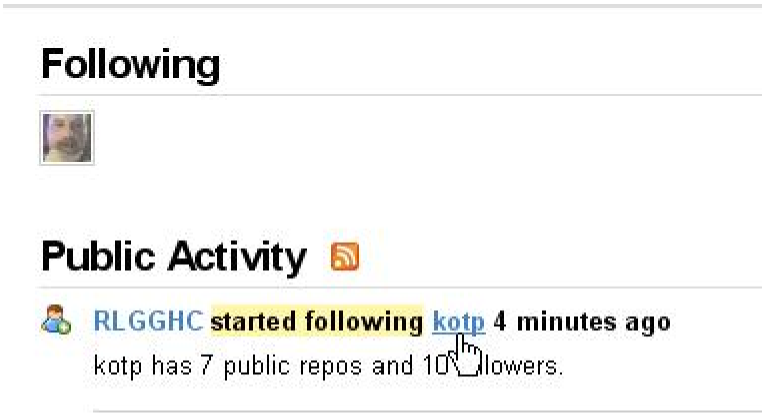
This leads us to the following page (see screenshot) -



kotp

Name:	Victor Goff
Email:	keeperotphones@gmail.com
Blog:	http://vgoff.blogspot.com
Location:	Minnesota
Followers:	9
Public Repos:	7
Member Since:	Jul 10, 2008

Click on the “follow” button as shown above. Click on your GitHub account’s “profile” link. In the new page that comes up, you can now see that you are following **`kotp`** (refer the next screenshot) -



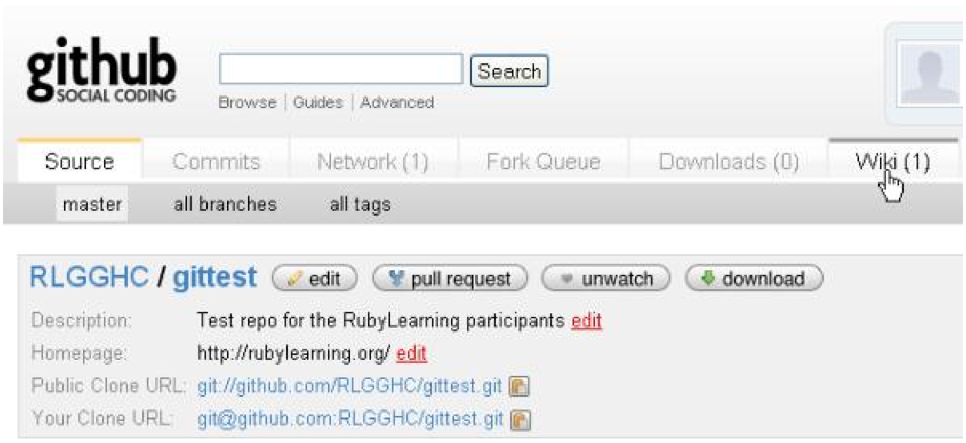
Following

Public Activity

RLGGHC started following kotp 4 minutes ago
kotp has 7 public repos and 10 followers.

Using the project wiki

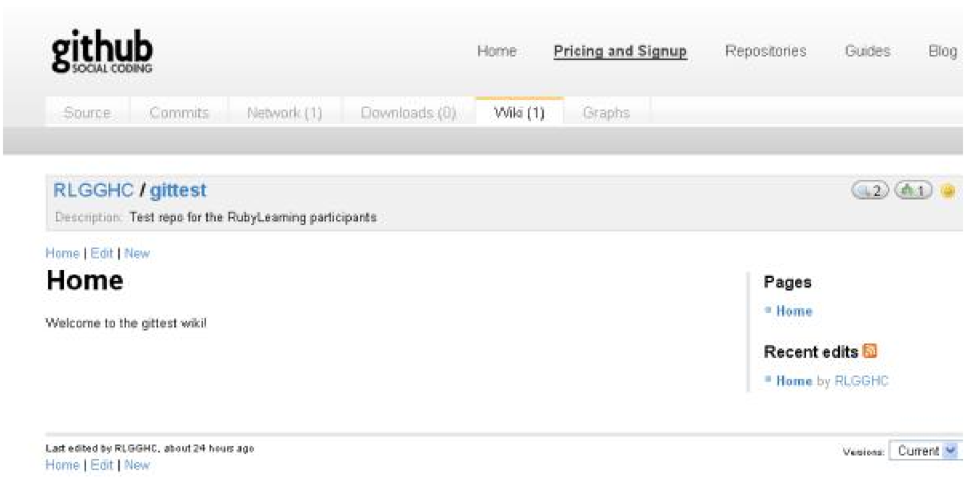
On your GitHub repo page, click on the “Wiki” link as shown:



RLGGHC / gittest

Description: Test repo for the RubyLearning participants edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/gittest.git
Your Clone URL: git@github.com:RLGGHC/gittest.git

This is how the Wiki looks for now:



RLGGHC / gittest

Description: Test repo for the RubyLearning participants

Home

Welcome to the gittest wiki!

Home by RLGGHC

Last edited by RLGGHC, about 24 hours ago

You can add new pages, edit existing pages on this wiki. The wiki can be used to keep your notes here. The wiki can be edited using Textile editing –
http://hobix.com/textile/quick.html

I created a new wiki page here –
http://wiki.github.com/RLGGHC/gittest/proposed-courses-at-rubylearning

Exercise 5

Create your own wiki page for your GitHub account. You decide what you would like to show on your wiki page, but remember that this would be a public page – viewable by all.

Branches

Branches fulfil the same role as drafts when writing an email. You work on the draft, saving it frequently until it is complete; then, when it's done, you send the email, and the draft is deleted. In this case, the outbox is not polluted by your frequent changes, until you hit "send".

Branching is useful when developing new features, because it allows the *master* branch – the outbox – to be always working and deployable. There may be any number of drafts – experimental branches – in active development. Branches are easy to create and switch between.

Once the code in a branch is finished and the branch passes its tests, the changes are *merged* into the master branch and the branch is removed, just like the email draft. And if someone commits code to the master branch, it's easy to update the branch to the latest master code.

You should always create a branch before starting work on a feature. This way, the master will always be in a working state, and you'll be able to work in isolation of other's changes. Creating a branch allows you to take the master branch, "clone" it, and make commits to that clone. Then when you're ready, you can merge the branch back into master; or, if there are changes made to master while you're working, you can merge those changes. It's just like pushing and pulling, but it all happens in the same directory.
Reference: http://hoth.entp.com/output/git_for_designers.html

The true power of Git comes out when you start using branches. The ability to easily and quickly create and manipulate branches is one of Git's major selling points.

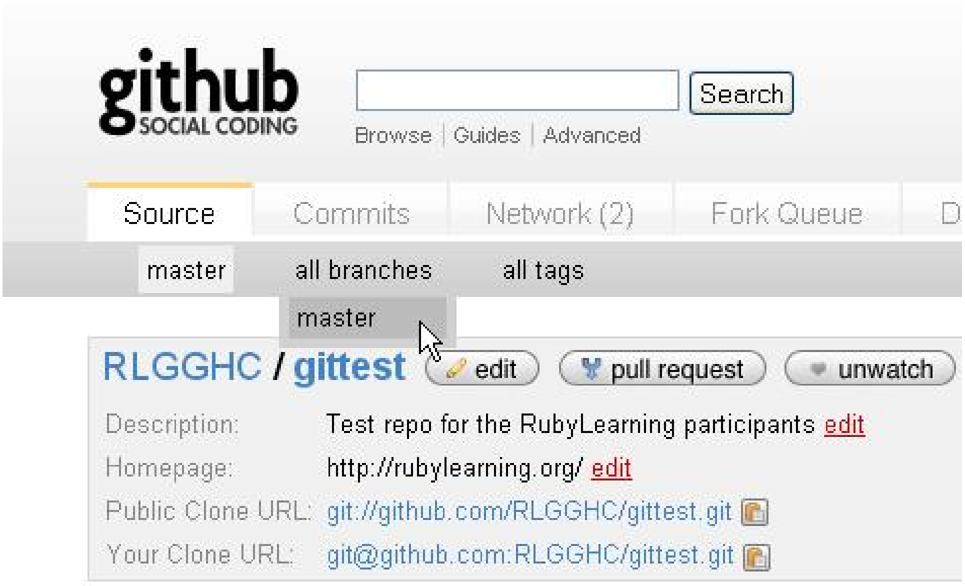
So far we have been working with a branch - **master**, as can be seen here:

```
A@COMP /e/gitlocalrepo/gittest
$ git branch
* master
A@COMP /e/gitlocalrepo/gittest
$
```

If you go to my GitHub repository here –

http://github.com/RLGGHC/gittest/tree/master

you will see (as in the screenshot) that I have only one branch called master.



Description: Test repo for the RubyLearning participants edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/gittest.git
Your Clone URL: git@github.com:RLGGHC/gittest.git

Creating a branch

Let us now create a new branch called “rubylearning” where we would work on a new feature in a file **`student.rb`**

To create the “rubylearning” branch, type in your local **`gittest`** repo:

```
$ git branch rubylearning
```

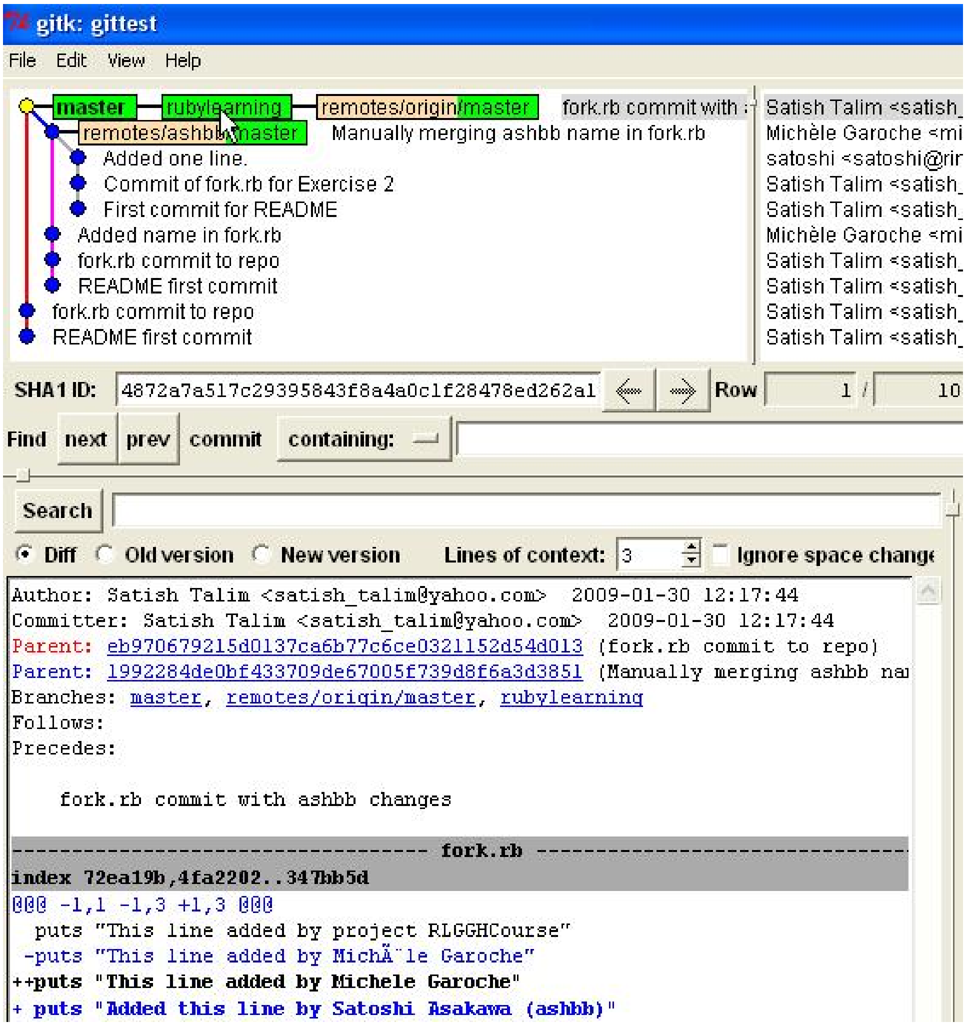
To confirm, type:

```
A@COMP /e/gitlocalrepo/gittest
$ git branch
* master
  rubylearning
A@COMP /e/gitlocalrepo/gittest
$
```

Notice, that the branch “rubylearning” has been created but we are in the “master” branch still. Next type:

```
A@COMP /e/gitlocalrepo/gittest
$ gitk
```

The next screenshot shows that “rubylearning” has the same contents as the “master” branch.



```
fork.rb commit with      Satish Talim <satish_
Manually merging ashbb name in fork.rb      Michèle Garoche <mi
Added one line.      satoshi <satoshi@rir
Commit of fork.rb for Exercise 2      Satish Talim <satish_
First commit for README      Satish Talim <satish_
Added name in fork.rb      Michèle Garoche <mi
fork.rb commit to repo      Satish Talim <satish_
README first commit      Satish Talim <satish_
fork.rb commit to repo      Satish Talim <satish_
README first commit      Satish Talim <satish_
```

```
4872a7a517c29395843f8a4a0c1f28478ed262a1
```

```
Author: Satish Talim <satish_talim@yahoo.com>  2009-01-30 12:17:44
Committer: Satish Talim <satish_talim@yahoo.com>  2009-01-30 12:17:44
Parent: eb970679215d0137ca6b77c6ce0321152d54d013 (fork.rb commit to repo)
Parent: 1992284de0bf433709de67005f739d8f6a3d3851 (Manually merging ashbb na
Branches: master, remotes/origin/master, rubylearning
Follows:
Precedes:

    fork.rb commit with ashbb changes

------------------------------------ fork.rb ---------------------------------
index 72ea19b,4fa2202..347bb5d
@@@ -1,1 -1,3 +1,3 @@@
  puts "This line added by project RLGGHCourse"
 -puts "This line added by MichÃ¨le Garoche"
++puts "This line added by Michele Garoche"
+ puts "Added this line by Satoshi Asakawa (ashbb)"
```

Close **`gitk`** and let us now switch to the “rubylearning” branch by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git checkout rubylearning
Switched to branch "rubylearning"
A@COMP /e/gitlocalrepo/gittest
$
```

To confirm, type:

```
A@COMP /e/gitlocalrepo/gittest
$ git branch
  master
* rubylearning
A@COMP /e/gitlocalrepo/gittest
$
```

Let’s create a simple **student.rb** file in the folder **e/gitlocalrepo/gittest** as follows:

```
# student.rb
class Student

end
```

Now let us type:

```
A@COMP /e/gitlocalrepo/gittest
$ git add student.rb
A@COMP /e/gitlocalrepo/gittest
$ git status
# On branch rubylearning
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   student.rb
#
A@COMP /e/gitlocalrepo/gittest
$
A@COMP /e/gitlocalrepo/gittest
$ git commit -m 'student class first commit'
[rubylearning]: created 0c9e1be: "student class first
commit"
 1 files changed, 4 insertions(+), 0 deletions(-)
 create mode 100644 student.rb

A@COMP /e/gitlocalrepo/gittest
$
```

Let’s now *push* this to our repo on GitHub. The syntax is:

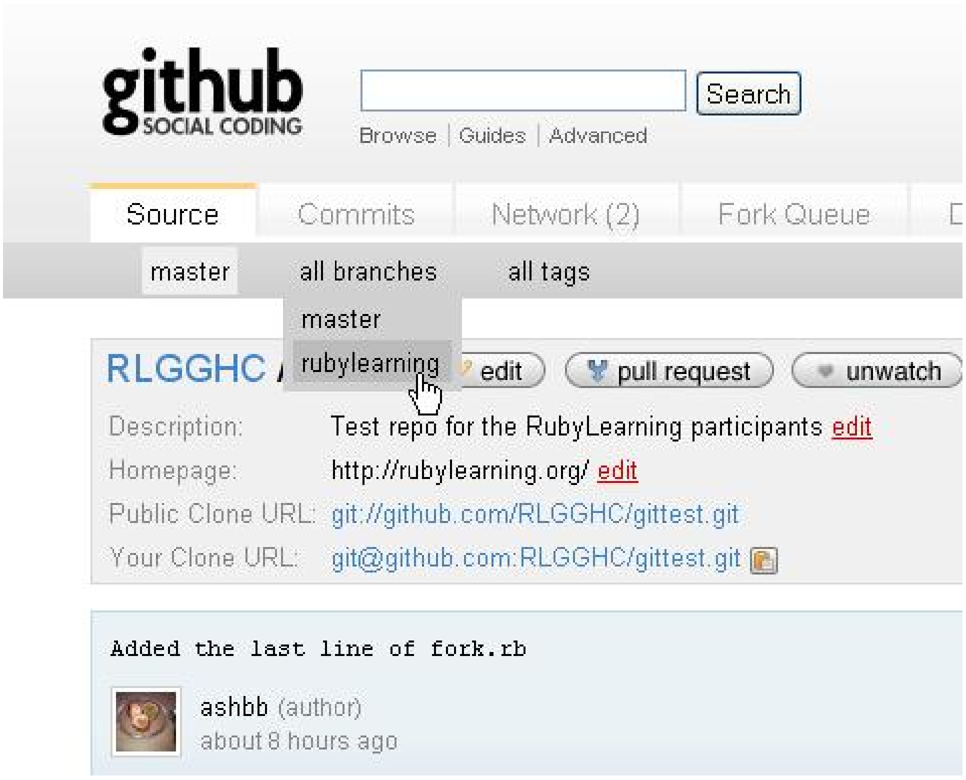
```
git push origin <localbranch>:<remotebranch>
```

And so this operation is really "pushing to the remote rubylearning branch."

Type:

```
A@COMP /e/gitlocalrepo/gittest
$ git push origin rubylearning
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 310 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@github.com:RLGGHC/gittest.git
 * [new branch]      rubylearning -> rubylearning
A@COMP /e/gitlocalrepo/gittest
$
```

See the next snapshot of my repo on GitHub. Notice that the “rubylearning” branch has been added:



RLGGHC /

Description: Test repo for the RubyLearning participants edit

Homepage: http://rubylearning.org/ edit

Public Clone URL: git://github.com/RLGGHC/gittest.git

Your Clone URL: git@github.com:RLGGHC/gittest.git

```
Added the last line of fork.rb
```

ashbb (author)
about 8 hours ago

In our local repo, let us switch back to the “master” branch and check what the *difference* is between the “master” and “rubylearning” branches.

```
A@COMP /e/gitlocalrepo/gittest
$ git checkout master
Switched to branch "master"
A@COMP /e/gitlocalrepo/gittest
$ git branch
* master
  rubylearning
A@COMP /e/gitlocalrepo/gittest
$ git diff master rubylearning
diff --git a/student.rb b/student.rb
new file mode 100644
index 0000000..cdba213
--- /dev/null
+++ b/student.rb
@@ -0,0 +1,4 @@
+# student.rb
+class Student
+
+end
\ No newline at end of file
A@COMP /e/gitlocalrepo/gittest
$
```

As you can see it says that a new file **`student.rb`** exists in the “rubylearning” branch.

Merging a branch

Let us assume that you have finished working in your “rubylearning” branch and would like to merge the changes made in “rubylearning” into “master”.

To achieve this, type:

```
A@COMP /e/gitlocalrepo/gittest
$ git checkout master
Already on "master"
A@COMP /e/gitlocalrepo/gittest
$ git merge rubylearning
Updating 4872a7a..0c9e1be
Fast forward
 student.rb |    4 ++++
 1 files changed, 4 insertions(+), 0 deletions(-)
 create mode 100644 student.rb
A@COMP /e/gitlocalrepo/gittest
$
```

If the changes don't conflict, you're done. If there are conflicts, markers will be left in the problematic files showing the conflict. Once you've edited the files to resolve the conflicts, type:

```
$ git add .
$ git commit –m ‘Commit after rubylearning branch merge’
```

will commit the result of the merge.

Deleting a branch

At this point you could delete the “rubylearning” branch with:

```
A@COMP /e/gitlocalrepo/gittest
$ git branch -d rubylearning
Deleted branch rubylearning (0c9e1be)
A@COMP /e/gitlocalrepo/gittest
$
```

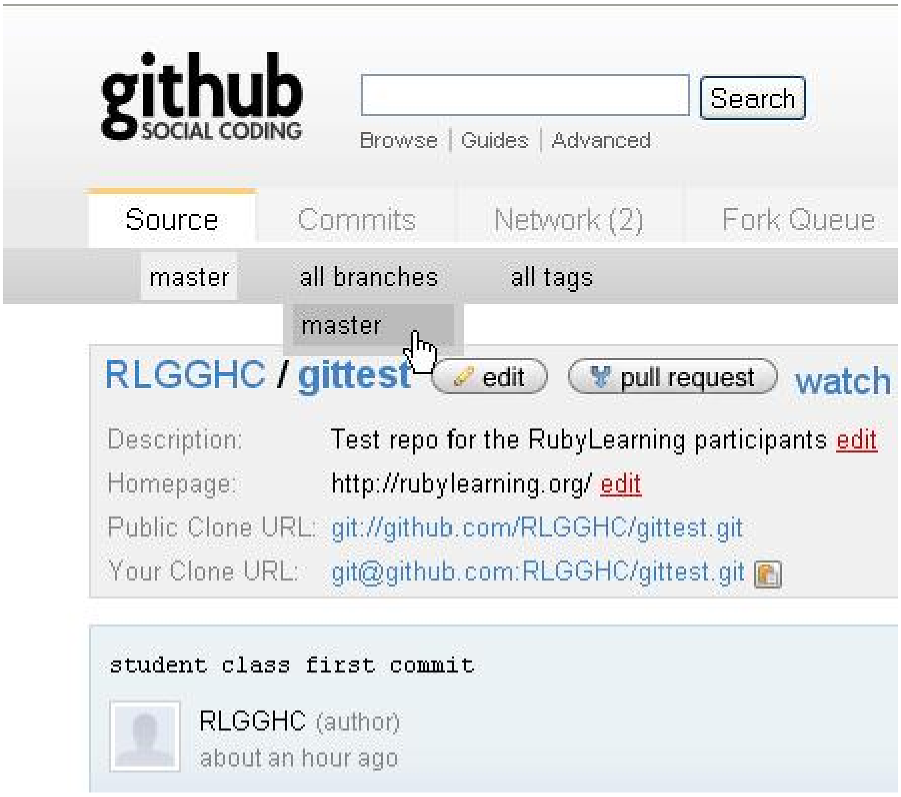
Now push the changes to the master repo on GitHub by typing:

```
$ git push origin master
```

Now we can safely remove the “rubylearning” branch from our repo on GitHub by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git push origin :rubylearning
To git@github.com:RLGGHC/gittest.git
 - [deleted]         rubylearning
A@COMP /e/gitlocalrepo/gittest
$
```

We confirm this by checking out the screenshot:



Description: Test repo for the RubyLearning participants edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/gittest.git
Your Clone URL: git@github.com:RLGGHC/gittest.git

student class first commit

RLGGHC (author)
about an hour ago

Reversing a merge

(Don’t actually execute ‘reversing a merge’; it is just for your information).

You can undo your last commit or merge by typing:

```
$ git reset --hard ORIG_HEAD
```

This will return master to the status before you attempted the merge. You can then change master and/or other branch to make them compatible and try the merge again.

Exercise 6

In the master branch of your local repo **gittest**, start writing a Ruby program called **motorcycle.rb** as shown:

```
class MotorCycle
  def initialize(make, color)
    # Instance variables
    @make = make
    @color = color
    end
  def startEngine
    if (@engineState)
      puts 'Engine is already Running'
    else
      @engineState = true
      puts 'Engine Idle'
    end
  end
end
```

Stage this file, *commit* it and *push* it to your **gittest** repo on GitHub.

Create a local "display" branch and *in this branch*, write **display.rb** as follows:

```
class Display < MotorCycle
  def dispAttr
    puts 'Color of MotorCycle is ' + @color
    puts 'Make  of MotorCycle is ' + @make
  end
end
```

Add this file to the '*stage*'. Next *commit* it to your local repo.

Push the **`display.rb`** file in branch **`display`** to the **`gittest`** repo and branch **`display`** on GitHub.

Now we have finished working in our “display” branch. Merge the changes made in “display” branch into “master” branch and delete the “display” branch for your local repo. Add all files for staging and commit to your local repo.

Finally, push the changes to the master repo on GitHub and delete the display branch on GitHub.

Git tag

Appendix A shows us that:

```
$ git log -2
commit 15dcc6fc3d43505ce830818f4ed02d6cdbe24b95
Author: Satish Talim <satish_talim@hotmail.com>
Date:   Sat Feb 14 10:00:42 2009 +0530
    first commit of display.rb in branch display
commit 16dd5bea05713a26be224018007e497c6a0f73f6
Author: Satish Talim <satish_talim@hotmail.com>
Date:   Sat Feb 14 09:53:10 2009 +0530
    first commit motorcycle.rb
A@COMP /e/gitlocalrepo/gittest
$
```

Displays the last two commits.

A tag is a special kind of object - it has a date, tagger (author) and its own ID.

We can create a tag to refer to a particular commit (**`15dcc6fc3d43505ce830818f4ed02d6cdbe24b95`**) by running **`git tag`**, as shown:

```
A@COMP /e/gitlocalrepo/gittest
$ git tag –a display_commit
15dcc6fc3d43505ce830818f4ed02d6cdbe24b95
A@COMP /e/gitlocalrepo/gittest
$
```

Here, **`display_commit`** is the tag name we have given. Now we can use **`display_commit`** to refer to the commit **`15dcc6fc3d43505ce830818f4ed02d6cdbe24b95`**.

Type the following to confirm the tag creation:

```
$ git tag
display_commit
A@COMP /e/gitlocalrepo/gittest
$
```

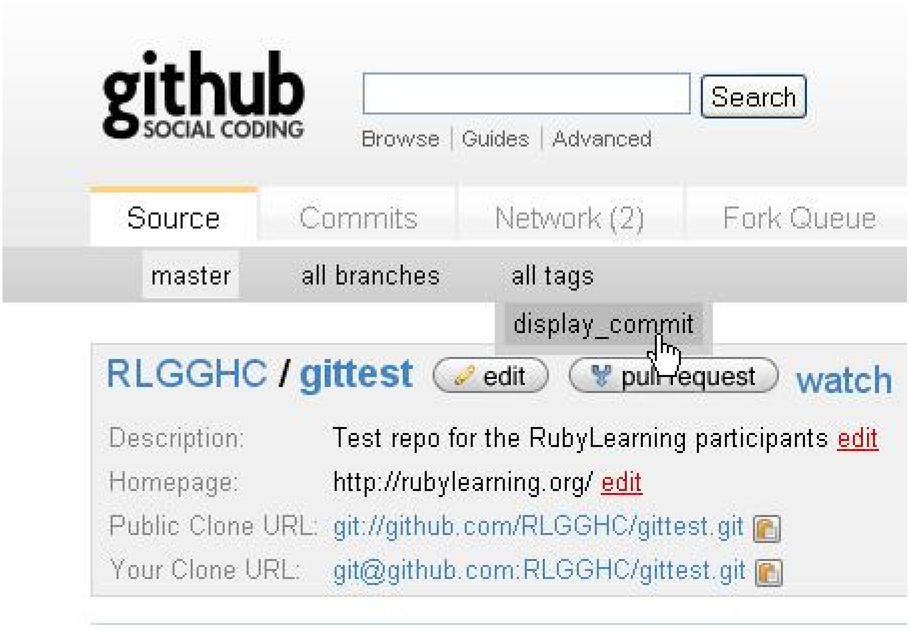
Next, we will push this tag to our repo on GitHub. We can either push an individual tag by typing:

```
A@COMP /e/gitlocalrepo/gittest
$ git push origin display_commit
Total 0 (delta 0), reused 0 (delta 0)
To git@github.com:RLGGHC/gittest.git
 * [new tag]         display_commit -> display_commit
A@COMP /e/gitlocalrepo/gittest
$
```

Or we can push all tags by typing:

```
$ git push origin --tags
```

Let us reload our repo page on GitHub and check whether the tag **display_commit** is available:



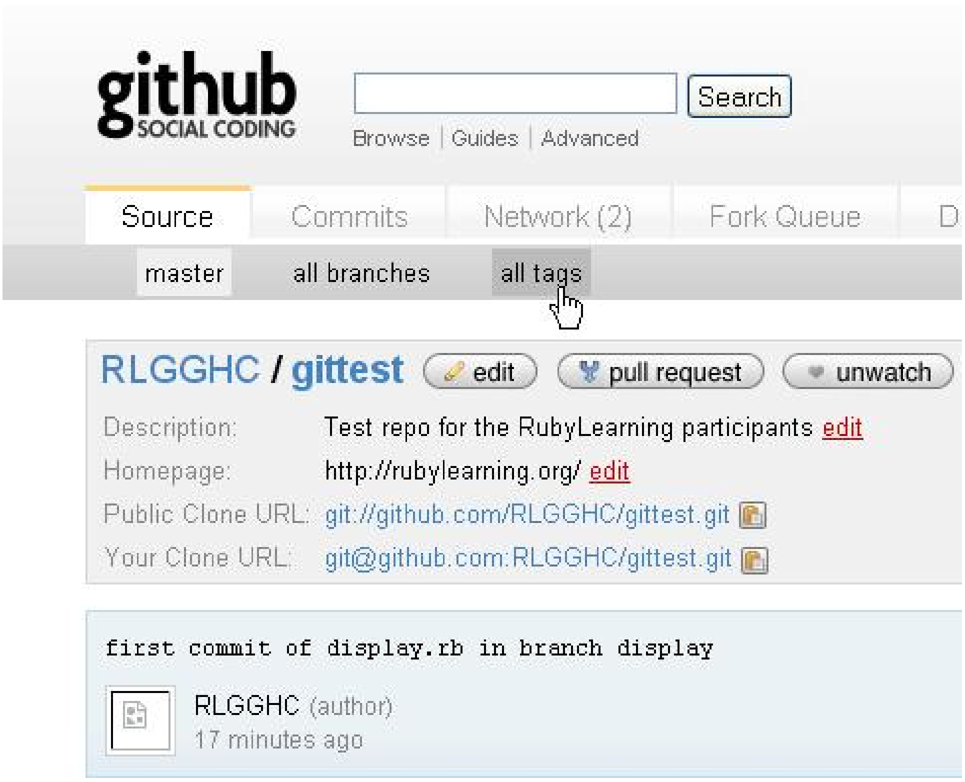
RLGGHC / gittest

Description: Test repo for the RubyLearning participants edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/gittest.git
Your Clone URL: git@github.com:RLGGHC/gittest.git

To delete the tag which we just pushed to GItHub, type:

```
A@COMP /e/gitlocalrepo/gittest
$ git push origin :display_commit
To git@github.com:RLGGHC/gittest.git
 - [deleted]         display_commit
A@COMP /e/gitlocalrepo/gittest
$
```

Confirm whether the tag has been deleted by reloading the repo page on GitHub. See screenshot:



Description: Test repo for the RubyLearning participants edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/gittest.git
Your Clone URL: git@github.com:RLGGHC/gittest.git

first commit of display.rb in branch display

RLGGHC (author)
17 minutes ago

You can see that the tag is no longer available.

Distributing and Releasing Ruby Libraries As Gems

To solve various problems with Ruby, you might develop your own libraries. Also, you might want to open-source your libraries to get help from the Ruby community and have many developers working on the same.

A gem is a packaged Ruby application or library. It has a name and a version. Gems are managed on your computer using the **gem** command. You can install, remove, and query (among other things) gem packages using the **gem** command.

RubyGems is the name of the project that developed the gem packaging system and the gem command. With RubyGems it is easy to create “gems” of your own from your own code. RubyGems is now part of the standard library from Ruby version 1.9.

Installing RubyGems

Since we are using Ruby 1.8+ we will need to install RubyGems on our local PC. For a detailed explanation of the installation process, please refer to the following -
http://docs.rubygems.org/read/chapter/3

If you already have RubyGems installed, make sure that you are running RubyGems 1.3.0 or higher. To check, open a command window and type:

```
gem –v
```

If it is not 1.3.0 or higher, you need to upgrade your gem (The page - http://www.ruby-forum.com/topic/166853#732427 is very useful for doing that).

Hosting a RubyGem on GitHub

We have not yet created our gem but nevertheless we shall do the following first:

Set up our GitHub account

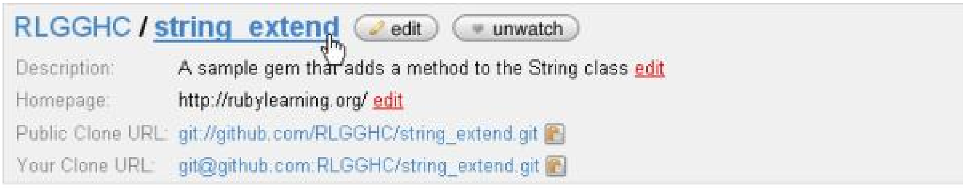
We have already done this before.

I have set up my account here – http://github.com/RLGGHC

I now log into this account. You too should log into your respective account on GitHub.

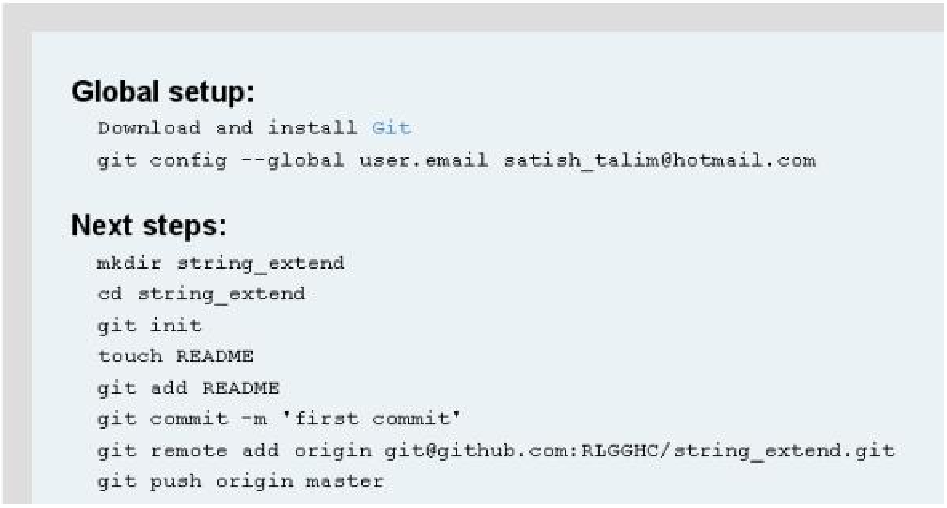
Create a new repository

We have already learnt how to set up our own repository. I have created a new repo by the name **`string_extend`** as shown:



RLGGHC / string_extend

Description: A sample gem that adds a method to the String class edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/string_extend.git
Your Clone URL: git@github.com:RLGGHC/string_extend.git



Global setup:

```
Download and install Git
git config --global user.email satish_talim@hotmail.com
```

Next steps:

```
mkdir string_extend
cd string_extend
git init
touch README
git add README
git commit -m 'first commit'
git remote add origin git@github.com:RLGGHC/string_extend.git
git push origin master
```

Important: Next go to your project's edit page and check the 'RubyGem' box – *this is an important step*.

Creating a Gem

Now, open a bash shell in your local repository folder (I have created and use **`gitlocalrepo`** folder on E: of my hard disk) and type:

```
A@COMP /e/gitlocalrepo
$ mkdir string_extend
A@COMP /e/gitlocalrepo
$ cd string_extend
A@COMP /e/gitlocalrepo/string_extend
$ mkdir lib
A@COMP /e/gitlocalrepo/string_extend
$
```

We will create a simple library that opens up the **`String`** class and adds a method **`writesize`** (which returns the size of the string). Let's store this program in a file called **`string_extend.rb`** in the folder **`e:/gitlocalrepo/string_extend/lib`**

```
# string_extend.rb
class String
  def writesize
    self.size
  end
end
```

Let us now turn it into a gem, so that you can use it anywhere.

Structuring Your Files

First, it's necessary to create a folder that holds everything. We have already created a folder called **`string_extend`**. Under this folder you can create several other folders as follows:

lib: This directory will contain the Ruby code related to the library.
test: This directory will contain any unit tests or other testing scripts related to the library.
doc: This is an optional directory that could contain documentation about the library, particularly documentation created with or by rdoc.

In this example, we have placed **`string_extend.rb`** within the **`string_extend/lib`** directory.

Creating a Specification File

Reference:
http://www.rubygems.org/read/chapter/20

In order to create a gem, you need to define a gem specification, commonly called a "gemspec". A gemspec consists of several *attributes*. Some of these are required; most of them are optional.

Create a text file called **`string_extend.gemspec`** in the main **`string_extend`** folder, as follows:

```
Gem::Specification.new do |s|
  s.name = %q{string_extend}
  s.version = '0.0.3'
  s.summary = %q{string_extend adds useful features to the
String class}
  s.require_paths = ["lib"]
  s.files = ['lib/string_extend.rb']
  s.authors = ["Satish Talim"]
  s.email = %q{mail@satishtalim.com}
  s.homepage = %q{http://satishtalim.com/}
  s.has_rdoc = false
  s.rubygems_version = %q{1.3.0}
end
```

This is a basic specification file. Let’s look at a few key areas.

First you define the name of the gem, setting it to 'string_extend'. This attribute is *required*. The name does not include the version number:
`s.name = %q{string_extend}`

Next, you define the version number and this attribute is *required*. Typically, version numbers for Ruby projects (and for Ruby itself) contain three parts in order of significance. Early versions of software, before an official release, perhaps, often begin with 0, as in 0.0.3 here:
`s.version = '0.0.3'`

The summary line (this attribute is *required*) is displayed by gem list, and can be useful to people prior to installing the gem. Simply put together a short description of your library/gem here:
`s.summary = %q{string_extend adds useful features to the String class}`

The require_paths attribute is *required* and is ["lib"] by default.
`s.require_paths = ["lib"]`

The files attribute is optional (but mandatory for GitHub) and is an array of file names to be contained in the gem.

```
s.files = ['lib/string_extend.rb']
```

The homepage attribute is optional and if given it is the URL of the project or author.

```
s.homepage = %q{http://satishtalim.com/}
```

Last, the rubygems_version is optional and is the version of RubyGems used to create this gem.

```
s.rubygems_version = %q{1.3.0}
```

Back to our local repo

Type the following:

```
A@COMP /e/gitlocalrepo/string_extend
$ git init
Initialized empty Git repository in
e:/gitlocalrepo/string_extend/.git/
```

The above command needs to be typed only once. Next type:

```
A@COMP /e/gitlocalrepo/string_extend
$ touch string_extend.gemspec
```

Now type:

```
A@COMP /e/gitlocalrepo/string_extend
$ git add .

A@COMP /e/gitlocalrepo/string_extend
$ git commit -m 'commit of gem 0.0.3 version'
[master (root-commit)]: created 94b78af: "first commit of gemspec"
 2 files changed, 18 insertions(+), 0 deletions(-)
 create mode 100644 lib/string_more.rb
 create mode 100644 string_extend.gemspec
A@COMP /e/gitlocalrepo/string_extend
$

A@COMP /e/gitlocalrepo/string_extend
$ git remote add origin git@github.com:RLGGHC/string_extend.git
A@COMP /e/gitlocalrepo/string_extend
$
```

The above **`git remote add origin`** command needs to be executed only once. Next type,

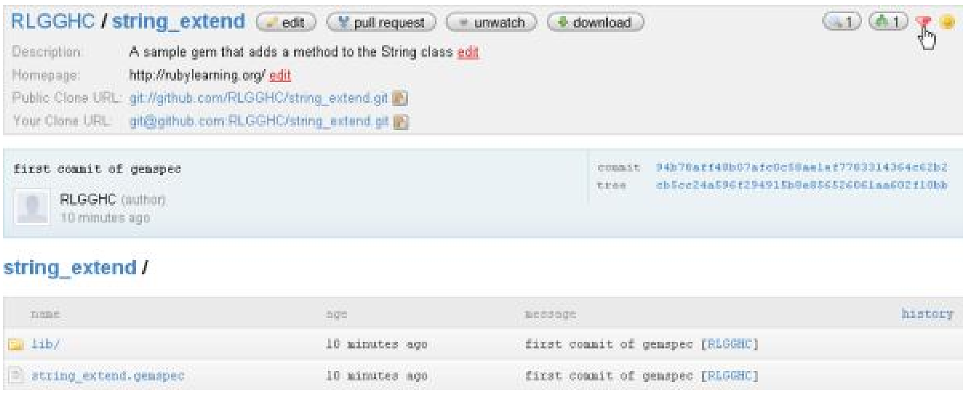
```
A@COMP /e/gitlocalrepo/string_extend
$ git push origin master
Counting objects: 5, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 620 bytes, done.
Total 5 (delta 0), reused 0 (delta 0)
To git@github.com:RLGGHC/string_extend.git
 * [new branch]      master -> master
A@COMP /e/gitlocalrepo/string_extend
$
```

We have pushed our gemspec to GitHub. GitHub will build a gem (prefixed with your username) on the fly. Note that once we have pushed our gemspec to GitHub, it may take up to 15 minutes for our new gem to find its way into GitHub's index.

Any time you push a gemspec with a bumped version number, GitHub will build a new gem for you.

If the gem build failed or is a success, you will get a message from GitHub, in your inbox.

Also, you should see a small red diamond, which means that your gem is ready. See the following screenshot:



Description: A sample gem that adds a method to the String class edit
Homepage: http://rubylearning.org/ edit
Public Clone URL: git://github.com/RLGGHC/string_extend.git
Your Clone URL: git@github.com:RLGGHC/string_extend.git

first commit of gemspec
RLGGHC (author)
10 minutes ago
commit 94b70aff40b07afc0c58ae1ef7703314364c62b2
tree cb5cc24a596f294915b0e8565260610a602f10bb

string_extend /

name	age	message	history
lib/	10 minutes ago	first commit of gemspec [RLGGHC]	
string_extend.gemspec	10 minutes ago	first commit of gemspec [RLGGHC]	

Installing our RubyGem from GitHub

You should be able to see the created gem **RLGGHC-string_extend** here – http://gems.github.com/list.html

In a new command window, first check your gem sources by typing:

```
e:\> gem sources --list
*** CURRENT SOURCES ***
http://gems.rubyforge.org/
http://gems.github.com
e:\>
```

If http://gems.github.com is not listed as a source, then type the following:

```
e:\> gem sources -a http://gems.github.com
```

Install the gem by typing:

```
e:\> gem install RLGGHC-string_extend
```

Once you have locally installed our gem, you can use it in your program as follows:

```
# use it as follows
require ‘string_extend’
size_writer = "Tell me my size!"
puts size_writer.writesize
```

References

Building a Ruby Gem –
http://blog.bogojoker.com/2008/05/building-a-ruby-gem/

RubyGems User Guide –
http://docs.rubygems.org/read/book/1

GitHub Ruby Gems –
http://gems.github.com/

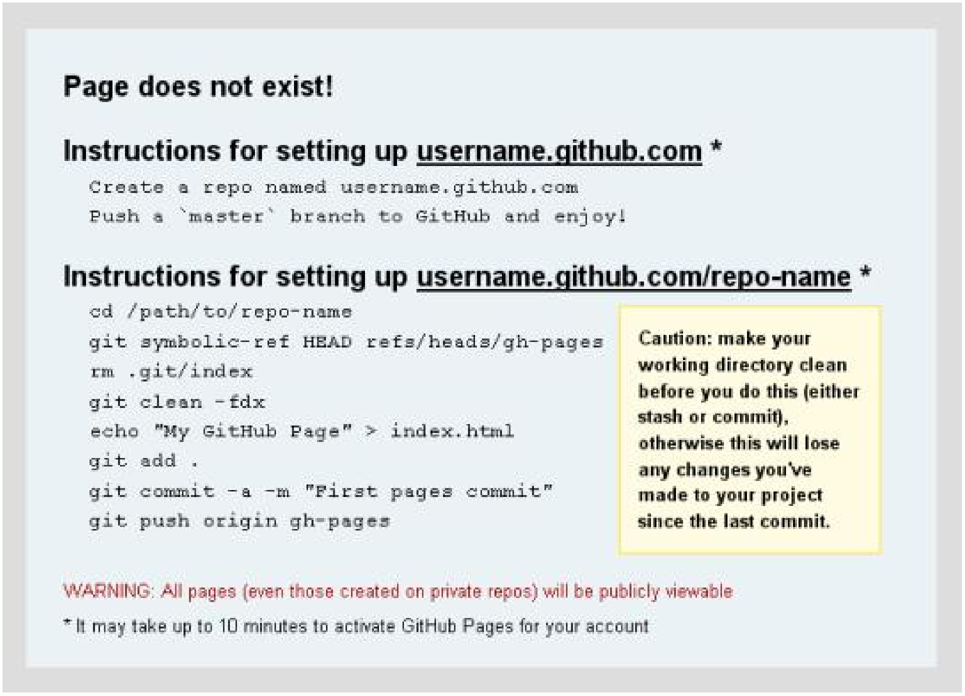
Library Development in Ruby –
http://devlib.wikidot.com/ruby

GitHub Pages

GitHub Pages allow you to publish web content to a github.com subdomain named after your username. With Pages, publishing web content becomes as easy as pushing to your GitHub repository.

If you create a repository named you.github.com, where you is your username, and push content to it, GitHub will automatically publish that to http://you.github.com. No FTP, no scp, no rsync, nothing. Just a simple **`git push`** and you're done. You can put anything here you like. Use it as a customizable home for your Git repos. Create a blog and spread your ideas. Whatever you want!

If you create a gh-pages branch on any regular repository and push content there, GitHub will automatically publish that to http://you.github.com/your-repo. This allows you to create instant documentation sites that are as easy to collaborate on as your code. Since you'll want a blank slate for your Pages branch, you can use a little Git trick to create a new branch that has no parents. Just follow the instructions at pages.github.com and you'll be up and running in a few seconds.



Page does not exist!

Instructions for setting up username.github.com *

```
Create a repo named username.github.com
Push a `master` branch to GitHub and enjoy!
```

Instructions for setting up username.github.com/repo-name *

```
cd /path/to/repo-name
git symbolic-ref HEAD refs/heads/gh-pages
rm .git/index
git clean -fdx
echo "My GitHub Page" > index.html
git add .
git commit -a -m "First pages commit"
git push origin gh-pages
```

Caution: make your working directory clean before you do this (either stash or commit), otherwise this will lose any changes you've made to your project since the last commit.

WARNING: All pages (even those created on private repos) will be publicly viewable

* It may take up to 10 minutes to activate GitHub Pages for your account

First, create a repo in your account on GitHub as **`rlgghc.github.com`** (Note: your repo name could be different than **`rlgghc`**).

Next, this is what I did in my local folder **`gitlocalrepo`**. Type:

```
$ mkdir rlgghc.github.com
$ cd rlgghc.github.com
$ git init
$ touch README
$ git add README
$ git commit -m 'first commit README'
$ git remote add origin
git@github.com:RLGGHC/rlgghc.github.com.git
$ git push origin master
```

Next I typed:

```
$ cd rlgghc.github.com/
$ git symbolic-ref HEAD refs/heads/gh-pages
$ rm .git/index
$ git clean -fdx
$ echo "My GitHub Page" > index.html
$ git add .
$ git commit -a -m "First pages commit"
$ git push origin gh-pages
```

My new page comes up at - `http://rlgghc.github.com/`.

Now I want to enhance my **`index.html`** page and for that I first type the following in my local folder – **`e/gitlocalrepo/rlgghc.github.com`**

```
$ git symbolic-ref HEAD refs/heads/gh-pages
$ rm .git/index
$ git clean –fdx
```

Next, I add a new **`index.html`**, **`style.css`** and an **`images`** folder in the folder **`e/gitlocalrepo/rlgghc.github.com`** (you can see these files here – `http://github.com/RLGGHC/rlgghc.github.com/tree/gh-pages`)

Finally I type:

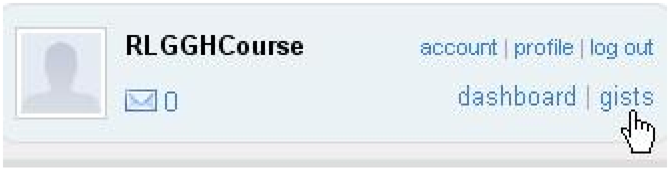
```
$ git add .
$ git commit -a -m "First pages commit"
$ git push origin gh-pages
```

My new page comes up at - `http://rlgghc.github.com/`.

Here's an example of another page: `http://mojombo.github.com/`

Gist

Gist is a cool feature of GitHub. This is available as shown in the following screenshot:



Gist is basically a paste tool, similar to Pastie (http://pastie.org/). Paste tools make it easy to share text or code - the user pastes the text into a text field on the site which returns a URL under which this text is accessible.

Gist works the same way - except with a twist: the pasted text ends up in a Git repository, which is accessible via a Git clone URL - it can be checked out with a Git client program. Not just that, the connection works both ways: pushing a commit back to the repository makes the changes (and added files) accessible in the Gist web interface, which also allows to edit text files. Gist allows you to start off from a pasted bit of text or code and collaborate over the net, either using a web interface or Git tools.

---- The End ----

Appendix A

(Thanks for this note to Michèle Garoche).

The **`git log`** command allows you to show commit history and has some powerful flags. Here are some:

All the commands should be executed in your **`gittest`** local tree.

For more details, see the **`git log`** and **`git show`** man pages.

1 - **`git log -2`**

Displays the last two commits:

```
git log -2
commit 1992284de0bf433709de67005f739d8f6a3d3851
Merge: a2b0175... ca1e574...
Author: Michèle Garoche <migatine@gmail.com>
Date:   Thu Jan 29 08:57:38 2009 +0100

    Manually merging ashbb name in fork.rb

commit a2b01758b40950d3dd8e3b40f8d233d92e5a06bd
Author: Michèle Garoche <migatine@gmail.com>
Date:   Thu Jan 29 05:19:32 2009 +0100

    Added name in fork.rb
```

2 - **`git log --stat`**

Displays details on the commits:

```
git log --stat
commit 1992284de0bf433709de67005f739d8f6a3d3851
Merge: a2b0175... ca1e574...
Author: Michèle Garoche <migatine@gmail.com>
Date:   Thu Jan 29 08:57:38 2009 +0100

    Manually merging ashbb name in fork.rb

commit a2b01758b40950d3dd8e3b40f8d233d92e5a06bd
```

```
Author: Michèle Garoche <migatine@gmail.com>
Date:   Thu Jan 29 05:19:32 2009 +0100

    Added name in fork.rb

 fork.rb |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)

commit 69596793be76527130517508d640868d2fd35c84
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Wed Jan 28 17:36:05 2009 +0530

    fork.rb commit to repo

 fork.rb |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)

commit 7373fe28a5e96600b2c82b2f97613d81e36cbfae
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Wed Jan 28 16:57:49 2009 +0530

    README first commit

 0 files changed, 0 insertions(+), 0 deletions(-)

commit ca1e574a91c3cbf120dfa4570e4a487e69a3ba20
Author: satoshi <satoshi@rin-shun.com>
Date:   Wed Jan 28 01:23:49 2009 +0900

    Added one line.

 fork.rb |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)

commit b8774a90d359ea9a1739ae8707e3f982da32283c
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Mon Jan 26 09:54:31 2009 +0530

    Commit of fork.rb for Exercise 2

 fork.rb |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)

commit 17415059958f22fe41f10020678129bd3ec84723
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Sun Jan 25 12:33:36 2009 +0530

    First commit for README
 0 files changed, 0 insertions(+), 0 deletions(-)
```

3 - **git log --shortstat**

Same as --stat but reduces the details of the message.

```
git log --shortstat
commit 1992284de0bf433709de67005f739d8f6a3d3851
Merge: a2b0175... ca1e574...
Author: Michèle Garoche <migatine@gmail.com>
Date:   Thu Jan 29 08:57:38 2009 +0100

    Manually merging ashbb name in fork.rb

commit a2b01758b40950d3dd8e3b40f8d233d92e5a06bd
Author: Michèle Garoche <migatine@gmail.com>
Date:   Thu Jan 29 05:19:32 2009 +0100

    Added name in fork.rb

 1 files changed, 1 insertions(+), 0 deletions(-)

commit 69596793be76527130517508d640868d2fd35c84
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Wed Jan 28 17:36:05 2009 +0530

    fork.rb commit to repo

 1 files changed, 1 insertions(+), 0 deletions(-)

commit 7373fe28a5e96600b2c82b2f97613d81e36cbfae
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Wed Jan 28 16:57:49 2009 +0530

    README first commit

 0 files changed, 0 insertions(+), 0 deletions(-)

commit ca1e574a91c3cbf120dfa4570e4a487e69a3ba20
Author: satoshi <satoshi@rin-shun.com>
Date:   Wed Jan 28 01:23:49 2009 +0900

    Added one line.

 1 files changed, 1 insertions(+), 0 deletions(-)

commit b8774a90d359ea9a1739ae8707e3f982da32283c
```

```
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Mon Jan 26 09:54:31 2009 +0530

    Commit of fork.rb for Exercise 2

 1 files changed, 1 insertions(+), 0 deletions(-)

commit 17415059958f22fe41f10020678129bd3ec84723
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Sun Jan 25 12:33:36 2009 +0530

    First commit for README

 0 files changed, 0 insertions(+), 0 deletions(-)
```

4 - **git log --name-status**

Displays M (for modified files), A (for added files), D (for delete files)

git log --name-status

```
commit 1992284de0bf433709de67005f739d8f6a3d3851
Merge: a2b0175... ca1e574...
Author: Michèle Garoche <migatine@gmail.com>
Date:   Thu Jan 29 08:57:38 2009 +0100

    Manually merging ashbb name in fork.rb

commit a2b01758b40950d3dd8e3b40f8d233d92e5a06bd
Author: Michèle Garoche <migatine@gmail.com>
Date:   Thu Jan 29 05:19:32 2009 +0100

    Added name in fork.rb

M       fork.rb

commit 69596793be76527130517508d640868d2fd35c84
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Wed Jan 28 17:36:05 2009 +0530

    fork.rb commit to repo

A       fork.rb

commit 7373fe28a5e96600b2c82b2f97613d81e36cbfae
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Wed Jan 28 16:57:49 2009 +0530
```

```
    README first commit

A       README

commit ca1e574a91c3cbf120dfa4570e4a487e69a3ba20
Author: satoshi <satoshi@rin-shun.com>
Date:   Wed Jan 28 01:23:49 2009 +0900

    Added one line.

M       fork.rb

commit b8774a90d359ea9a1739ae8707e3f982da32283c
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Mon Jan 26 09:54:31 2009 +0530

    Commit of fork.rb for Exercise 2

A       fork.rb

commit 17415059958f22fe41f10020678129bd3ec84723
Author: Satish Talim <satish_talim@thotmail.com>
Date:   Sun Jan 25 12:33:36 2009 +0530

    First commit for README

A       README
```

5 - **`git log --pretty=fuller --stat`**

Displays the full stat in a nice format.

```
git log --pretty=fuller --stat
commit 1992284de0bf433709de67005f739d8f6a3d3851
Merge: a2b0175... ca1e574...
Author:     Michèle Garoche <migatine@gmail.com>
AuthorDate: Thu Jan 29 08:57:38 2009 +0100
Commit:     Michèle Garoche <migatine@gmail.com>
CommitDate: Thu Jan 29 08:57:38 2009 +0100

    Manually merging ashbb name in fork.rb

commit a2b01758b40950d3dd8e3b40f8d233d92e5a06bd
Author:     Michèle Garoche <migatine@gmail.com>
AuthorDate: Thu Jan 29 05:19:32 2009 +0100
```

```
Commit:     Michèle Garoche <migatine@gmail.com>
CommitDate: Thu Jan 29 05:19:32 2009 +0100

    Added name in fork.rb

 fork.rb |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)

commit 69596793be76527130517508d640868d2fd35c84
Author:     Satish Talim <satish_talim@thotmail.com>
AuthorDate: Wed Jan 28 17:36:05 2009 +0530
Commit:     Satish Talim <satish_talim@thotmail.com>
CommitDate: Wed Jan 28 17:36:05 2009 +0530

    fork.rb commit to repo

 fork.rb |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)

commit 7373fe28a5e96600b2c82b2f97613d81e36cbfae
Author:     Satish Talim <satish_talim@thotmail.com>
AuthorDate: Wed Jan 28 16:57:49 2009 +0530
Commit:     Satish Talim <satish_talim@thotmail.com>
CommitDate: Wed Jan 28 16:57:49 2009 +0530

    README first commit

 0 files changed, 0 insertions(+), 0 deletions(-)

commit ca1e574a91c3cbf120dfa4570e4a487e69a3ba20
Author:     satoshi <satoshi@rin-shun.com>
AuthorDate: Wed Jan 28 01:23:49 2009 +0900
Commit:     satoshi <satoshi@rin-shun.com>
CommitDate: Wed Jan 28 01:23:49 2009 +0900

    Added one line.

 fork.rb |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)

commit b8774a90d359ea9a1739ae8707e3f982da32283c
Author:     Satish Talim <satish_talim@thotmail.com>
AuthorDate: Mon Jan 26 09:54:31 2009 +0530
Commit:     Satish Talim <satish_talim@thotmail.com>
CommitDate: Mon Jan 26 09:54:31 2009 +0530

    Commit of fork.rb for Exercise 2

 fork.rb |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)

commit 17415059958f22fe41f10020678129bd3ec84723
Author:     Satish Talim <satish_talim@thotmail.com>
AuthorDate: Sun Jan 25 12:33:36 2009 +0530
Commit:     Satish Talim <satish_talim@thotmail.com>
CommitDate: Sun Jan 25 12:33:36 2009 +0530

    First commit for README

 0 files changed, 0 insertions(+), 0 deletions(-)
[594] Samedi 31/01/2009 07:59:56 CET +0100
```

6 - **`git log --pretty=format:"%h by %an - %ar" --author="Michèle"`**

Displays the objects, the author's first name and last name, today with a relative date for a given author.

```
1992284 by Michèle Garoche - 2 days ago
a2b0175 by Michèle Garoche - 2 days ago
```

7 - **`git show a2b0175`**

Displays the commit together with its diff.

```
git show a2b0175
commit a2b01758b40950d3dd8e3b40f8d233d92e5a06bd
Author: Michèle Garoche <migatine@gmail.com>
Date:   Thu Jan 29 05:19:32 2009 +0100

    Added name in fork.rb

diff --git a/fork.rb b/fork.rb
index 72ea19b..5506f14 100644
--- a/fork.rb
+++ b/fork.rb
@@ -1 +1,2 @@
 puts "This line added by project RLGGHC"
+puts "This line added by MichÃšle Garoche"
[598] Samedi 31/01/2009 08:02:46 CET +0100
[toto@tata:/RubyLearning/github-repositories/gittest
$ git show -p a2b0175
commit a2b01758b40950d3dd8e3b40f8d233d92e5a06bd
Author: Michèle Garoche <migatine@gmail.com>
```

```
Date:   Thu Jan 29 05:19:32 2009 +0100

    Added name in fork.rb

diff --git a/fork.rb b/fork.rb
index 72ea19b..5506f14 100644
--- a/fork.rb
+++ b/fork.rb
@@ -1 +1,2 @@
 puts "This line added by project RLGGHC"
+puts "This line added by MichÃšle Garoche"
```

8 - **git log --pretty=oneline -p fork.rb**

Displays the commits and the diff for a given file from its creation to now.

```
git log --pretty=oneline -p fork.rb
1992284de0bf433709de67005f739d8f6a3d3851 Manually merging ashbb
name in fork.rb
a2b01758b40950d3dd8e3b40f8d233d92e5a06bd Added name in fork.rb
diff --git a/fork.rb b/fork.rb
index 72ea19b..5506f14 100644
--- a/fork.rb
+++ b/fork.rb
@@ -1 +1,2 @@
 puts "This line added by project RLGGHC"
+puts "This line added by MichÃšle Garoche"
69596793be76527130517508d640868d2fd35c84 fork.rb commit to repo
diff --git a/fork.rb b/fork.rb
new file mode 100644
index 0000000..72ea19b
--- /dev/null
+++ b/fork.rb
@@ -0,0 +1 @@
+puts "This line added by project RLGGHC"
ca1e574a91c3cbf120dfa4570e4a487e69a3ba20 Added one line.
diff --git a/fork.rb b/fork.rb
index 72ea19b..472a6d9 100644
--- a/fork.rb
+++ b/fork.rb
@@ -1 +1,2 @@
 puts "This line added by project RLGGHC"
+puts "Added this line by Satoshi Asakawa (ashbb)"
\ No newline at end of file
```

```
b8774a90d359ea9a1739ae8707e3f982da32283c Commit of fork.rb for
Exercise 2
diff --git a/fork.rb b/fork.rb
new file mode 100644
index 0000000..72ea19b
--- /dev/null
+++ b/fork.rb
@@ -0,0 +1 @@
+puts "This line added by project RLGGHC"
```

About the Author

Satish Talim (http://satishtalim.com/) is a senior software consultant based in Pune, India with over 30+ years of I.T. experience. His experience lies in developing and executing business for high technology and manufacturing industry customers. Personally his strengths lie in Business Development and Business Networking apart from new product and solution ideas. Good experience of organization development. Excellent cross disciplinary background in engineering, computer science and management.

He -

- Has helped start subsidiaries for many US based software companies like **Infonox** based in San Jose, CA
http://www.infonox.com/default.shtml
Maybole Technologies Pvt. Ltd.
http://servient.com/
subsidiary of Servient Inc. (based in Houston, Texas) in Pune, India.

- Has been associated with Java / J2EE since 1995 and involved with Ruby and Ruby on Rails since 2005.
also started and manages two very active Java and Ruby User Groups in Pune, India - **PuneJava**
http://tech.groups.hotmail.com/group/pune-java/
and **PuneRuby**
http://tech.groups.hotmail.com/group/puneruby/

- Is a **Ruby Mentor**
http://rubymentor.rubyforge.org/wiki/wiki.pl?AvailablePureRubyMentors
on rubyforge.org, helping people with Ruby programming.

He lives in Pune, India, with his wife, son and his Labrador Benzy. In his limited spare time he enjoys traveling, photography and playing online chess.