Version

**DHIS2**

**Developer Manual**

|  |  |
| --- | --- |
| **Author** | **Mithilesh Kumar Thakur** |
| **Last change:** | **18 December 2017** |
| **Status:** | **Draft** |
| **Version**: | **1.0** |

**Document approved by:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Signature** | **Role** |
| Mithilesh Kumar Thakur |  | Lead Technical System |
|  |  |  |
|  |  |  |

**Version history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Change description** | **Author** |
| 3.4.2017 | 1.0 | Initial version of the document | **Mithilesh Kumar Thakur** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**1** **Introduction**

**1.1 Introduction:**

The document covers the information regarding basic architecture of HIV TRACKER SYSTEM. It provides installation procedures, technology framework, tools and development process for system.

**2** **Installation**

**2.1 Installation of software's**

Following are the steps to configure system for all developers

***Step 1:*** Software required with its flow of installation:

1. JDK/JRE – Install JDK and set the path as shown in step 3.
2. Apache Tomcat – Put this folder in some directory.
3. Database server (postgres) – Install postgres.
4. postgres – Install postgres for database GUI (pgAdmin).
5. Maven – Put Maven folder in some directory.
6. Git – Install git.
7. Tortoise SVN – Install
8. Eclipse – Put eclipse folder in some directory.

Note: Better to put Maven, Eclipse, Apache in one folder or directory (ex- C:)

***Step 2***: Get the database to import.

***Step 3***: Environment variable Setup:

1. Path to reach : My Computer -> Advanced system setting -> Advanced -> Environment Variables -> System Variables -> New
2. System Variable:

Name:  \_HOME

Value: c:\\_HOME

1. JDK/JRE :

**JAVA HOME**

Variable name: JAVA\_HOME

Variablevalue: C:\ProgramFiles\Java\jdk1.8

**JAVA OPTS**

Variable name: JAVA\_OPTS

Variablevalue: -Xms1024m -Xmx1536m -XX:MaxPermSize=512m

1. Apache Tomcat

**CATALINA HOME**

Variable name : CATALINA\_HOME

Variablevalue : C:\tomcat7

1. Maven

**MAVEN OPTS**

Variable name : MAVEN\_OPTS

Variablevalue: -Xms1024m -Xmx1536m-XX:MaxPermSize=512m

1. **Path to set :**

Variable name : Path

Variablevalue:C:\ProgramFiles\Java\jdk1.8\bin;C:\apache-maven-3.0.4\bin

1. **M2 REPO :**

Variable name : M2\_REPO

Variablevalue : c:\users\hisp\.m2

***Step 4***: Source Code Checkout of required modules in Directory:

1. Create directory
2. Write the following command inside directory.

$ git clone  [https://github.com/hispindia/-Core.git](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2Fhispindia%2Fdhis2-Core.git&sa=D&sntz=1&usg=AFQjCNERb56I_WYiJ-sLMHWILCxK2WTtTg)

$ git checkout SAVE-CHILD-2.27

1. Once checkout finished, your source code will be available inside the directory.

***Step 5***: Building HIV TRACKER SYSTEM module

1. Open Git bash
2. Go to the directory location through git bash using CD command where you put source code.
3. Now go inside the directory (**cd dhis-2**)
4. Enter the following commands:

1. Go to the dhis-2 folder inside root folder of source code and run the command

**mvn clean -Dmaven.test.skip=true install**

2. then go to dhis-web inside the folder dhis-2 and run the command

**mvn clean -Dmaven.test.skip=true install**

and after build success go to the folder **dhis-2\dhis-web\dhis-web-portal\target**

and get the war file name dhis.war and put it in tomcat webapp folder.

**OR**

We can directly run the command using pom-full.xml inside dhis-2 folder. (Ex- **mvn clean -f pom-full.xml -Dmaven.test.skip=true install).**

and get the war file name dhis.war and put it in tomcat webapp folder.

***Step 6***: Install and import database

1. Open pgAdmin.
2. Click on create database
3. Enter name of new database
4. Now, either import or restore the database inside the newly created database.

***Step 7***: DHIS2 dhis.conf file configuration setup

1. Go \_HOME folder under c:\\_HOME.
2. Paste the dhis.conf file here, this file contains :

hibernate.show\_sql = false  
hibernate.use\_sql\_comments = false

**# Hibernate SQL dialect**

hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect

**# JDBC driver class**

hibernate.connection.driver\_class = org.postgresql.Driver

**# Database connection URL**

hibernate.connection.url = jdbc:postgresql:name\_of\_db

**# Database username**

hibernate.connection.username = postgres

**# Database password**

hibernate.connection.password = root.

***Step 8***: Procedure to start DHIS2 application

1. Double click on startup (window batch file) in (tomcat7->bin) folder.
2. Go to web browser
3. Enter URL ([http://localhost:8080/dhis](http://localhost:8080/openmrs))
4. Provide the user name and password.
5. And enter the application.

***Steps 9***: Working with eclipse

**Importing source code inside eclipse:**

a. Go to the dhis-2 folder inside root folder of source code and run the command

**mvn clean eclipse:eclipse**

b. then go to dhis-web inside the folder dhis-2 and run the command

**mvn clean eclipse:eclipse**

then create workspace and Open eclipse folder double click on eclipse application file.

File ---> import ---> General ----> Existing projects into Workspace

and go to the source code location and import and click on Finish.

3.Restart the eclipse.

**Technologies Used:**

* Java
* Spring
* Struts2
* Hibernate
* AngularJS
* VM(VelocityMacro)
* HTML
* Javascript
* Jquery

**SOFTWARE INSTALLATION**

* **Install PostgreSQL**

There are three steps to complete the PostgreSQL installation:

1. Download PostgreSQL installer for Windows
2. Install PostgreSQL
3. Verify the installation

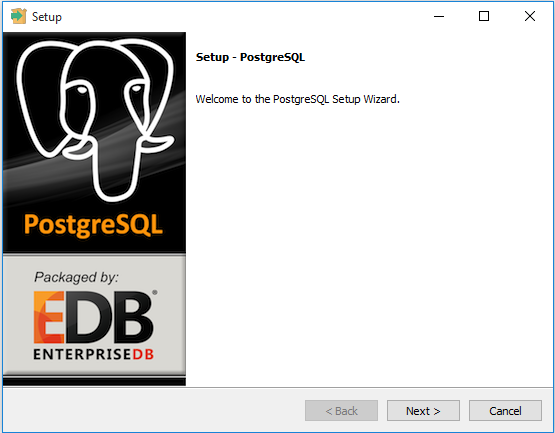
## **Download PostgreSQL Installer for Windows**

You need to download the installer from PostgreSQL Official website.

* Go to the PostgreSQL official website, download section for Windows <http://www.postgresql.org/download/windows/>
* Click on the download installer
* Choose the latest version to download. It takes few minutes to complete the download.

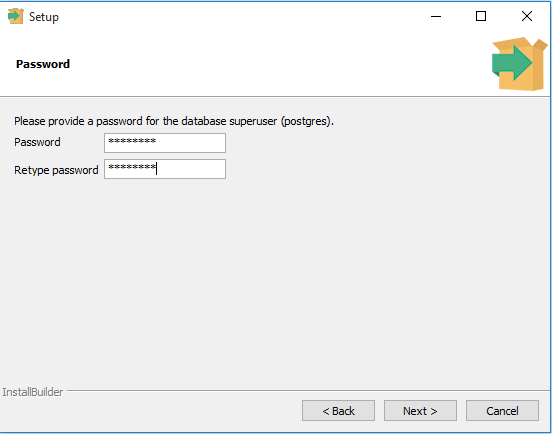
Double click on the installer file; an installation wizard will appear and guide you through multiple steps where you can choose different options that you would like to have in PostgreSQL.

Start Installing PostgreSQL

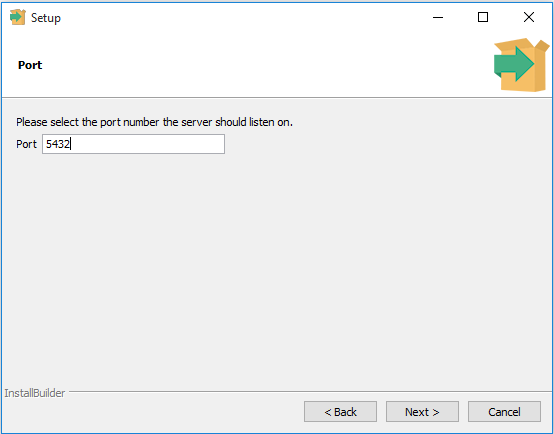


Specify installation folder, choose your own or keep the default folder suggested by PostgreSQL installer.

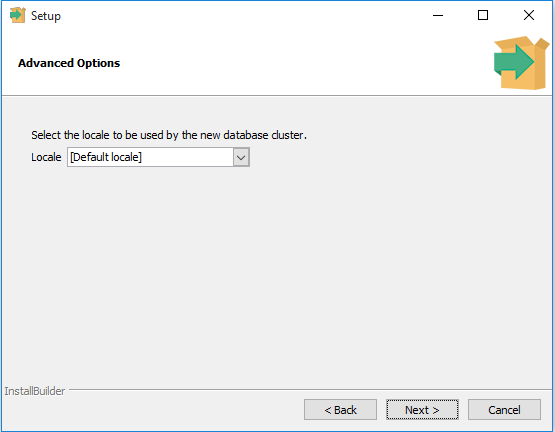
Enter the password for the database superuser and service account



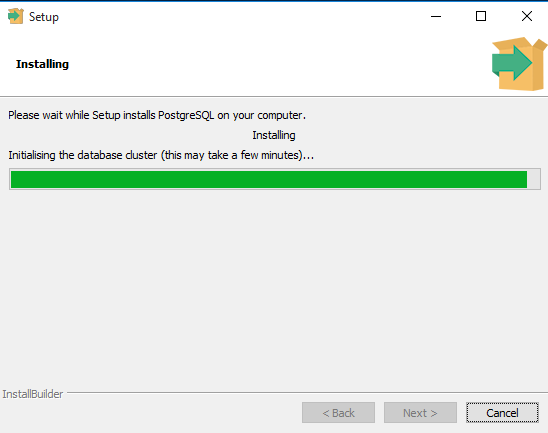
Enter the port for PostgreSQL. Make sure that no other applications are using this port. Leave it as default if you are unsure.



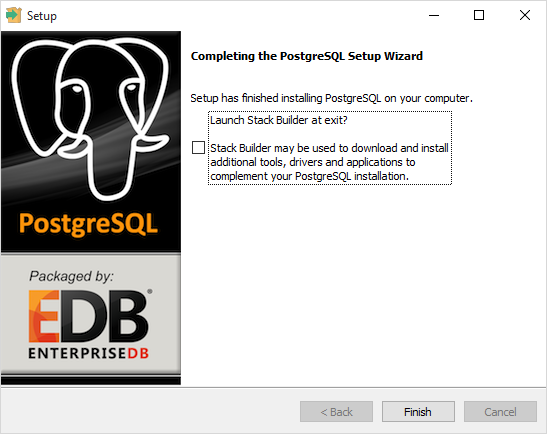
Choose the default locale used by the database.



You’ve completed providing information for the PostgreSQL installer. Click the Next button to install PostgreSQL



The installation may take few minutes to complete.



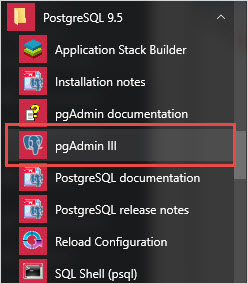
Click the Finish button to complete the PostgreSQL installation.

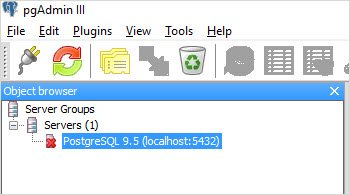
## **Verify the Installation**

There are several ways to verify the installation. You can try to [connect to the PostgreSQL](http://www.postgresqltutorial.com/connect-to-postgresql-database/) database server from any client application e.g.,  psql, pgAdmin, etc.

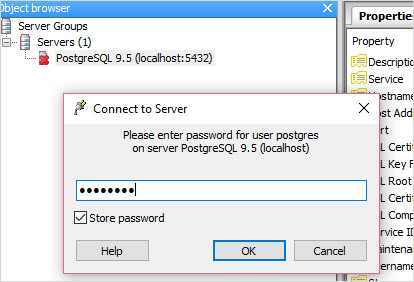
The quick way to verify the installation is through the pgAdmin application.

First, click on pgAdmin III to launch it. The pgAdmin III GUI will display.

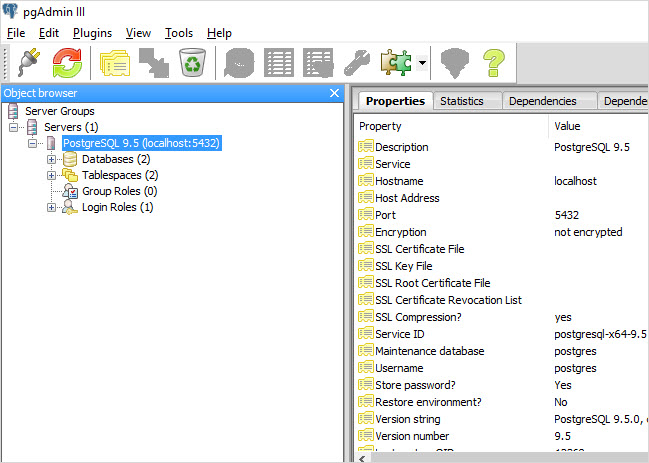




Second, double click on**PostgreSQL** on the object browser. It will ask you for the admin password. Just enter the password you’ve used in the installation step.



Third, if everything is fine, the pgAdmin will display all the objects that belong to the server.



Now, you’ve successfully installed PostgreSQL database server in your local system.

**GitHub**

## **What Is Git?**

A [version control](http://git-scm.com/video/what-is-version-control) application like Git keeps that from happening. You and your coworker can each upload your revisions to the same page, and Git will save two copies. Later, you can merge your changes together without losing any work along the way. You can even revert to an earlier version at any time, because Git keeps a “snapshot” of every change ever made.

## **Words People Use When They Talk About Git**

**Command Line:** The computer program we use to input Git commands. On a Mac, it’s called Terminal. On a PC, it’s a non-native program that you download when you download Git for the first time (we’ll do that in the next section). In both cases, you type text-based commands, known as prompts, into the screen, instead of using a mouse.

**Repository:** A directory or storage space where your projects can live. Sometimes GitHub users shorten this to “repo.” It can be local to a folder on your computer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, image files, you name it, inside a repository.

**Version Control:** Basically, the purpose Git was designed to serve. When you have a Microsoft Word file, you either overwrite every saved file with a new save, or you save multiple versions. With Git, you don’t have to. It keeps “snapshots” of every point in time in the project’s history, so you can never lose or overwrite it.

**Commit:** This is the command that gives Git its power. When you commit, you are taking a “snapshot” of your repository at that point in time, giving you a checkpoint to which you can reevaluate or restore your project to any previous state.

**Branch:** How do multiple people work on a project at the same time without Git getting them confused? Usually, they “branch off” of the main project with their own versions full of changes they themselves have made. After they’re done, it’s time to “merge” that branch back with the “master,” the main directory of the project.

## **Git-Specific Commands**

git init**:** Initializes a new Git repository. Until you run this command inside a repository or directory, it’s just a regular folder. Only after you input this does it accept further Git commands.

git config**:** Short for “configure,” this is most useful when you’re setting up Git for the first time.

git help**:** Forgot a command? Type this into the command line to bring up the 21 most common git commands. You can also be more specific and type “git help i2nit” or another term to figure out how to use and configure a specific git command.

git status**:** Check the status of your repository. See which files are inside it, which changes still need to be committed, and which branch of the repository you’re currently working on.

git add**:** This does *not* add new files to your repository. Instead, it brings new files to Git’s attention. After you add files, they’re included in Git’s “snapshots” of the repository.

git commit**:** Git’s most important command. After you make any sort of change, you input this in order to take a “snapshot” of the repository. Usually it goes git commit -m “Message here.” The -m indicates that the following section of the command should be read as a message.

git branch**:** Working with multiple collaborators and want to make changes on your own? This command will let you build a new branch, or timeline of commits, of changes and file additions that are completely your own. Your title goes after the command. If you wanted a new branch called “cats,” you’d type git branch cats.

git checkout**:** Literally allows you to “check out” a repository that you are not currently inside. This is a navigational command that lets you move to the repository you want to check. You can use this command as git checkout master to look at the master branch, or git checkout cats to look at another branch.

git merge**:** When you’re done working on a branch, you can merge your changes back to the master branch, which is visible to all collaborators. git merge cats would take all the changes you made to the “cats” branch and add them to the master.

git push**:** If you’re working on your local computer, and want your commits to be visible online on GitHub as well, you “push” the changes up to GitHub with this command.

git pull**:** If you’re working on your local computer and want the most up-to-date version of your repository to work with, you “pull” the changes down from GitHub with this command.

## **Setting Up GitHub And Git for The First Time**

First, you’ll need to [sign up for an account](https://github.com/) on GitHub.com. It’s as simple as signing up for any other social network.

You could stop there and GitHub would work fine. But if you want to work on your project on your local computer, you need to have Git installed. In fact, GitHub won’t work on your local computer if you don’t install Git.

http://git-scm.com/, from where you download Git.

Now it’s time to go over to the command line. On Windows, that means starting the Git Bash app you just installed, it’s regular old Terminal. It’s time to introduce yourself to Git. Type in the following code:

git config --global user.name "Your Name Here"

Of course, you’ll need to replace “Your Name Here” with your own name in quotations. It can be your legal name, your online handle, anything. Git doesn’t care, it just needs to know to whom to credit commits and future projects.

Next, tell it your email and make sure it’s the same email you used when you signed up for a GitHub.com account just a moment ago. Do it like this:

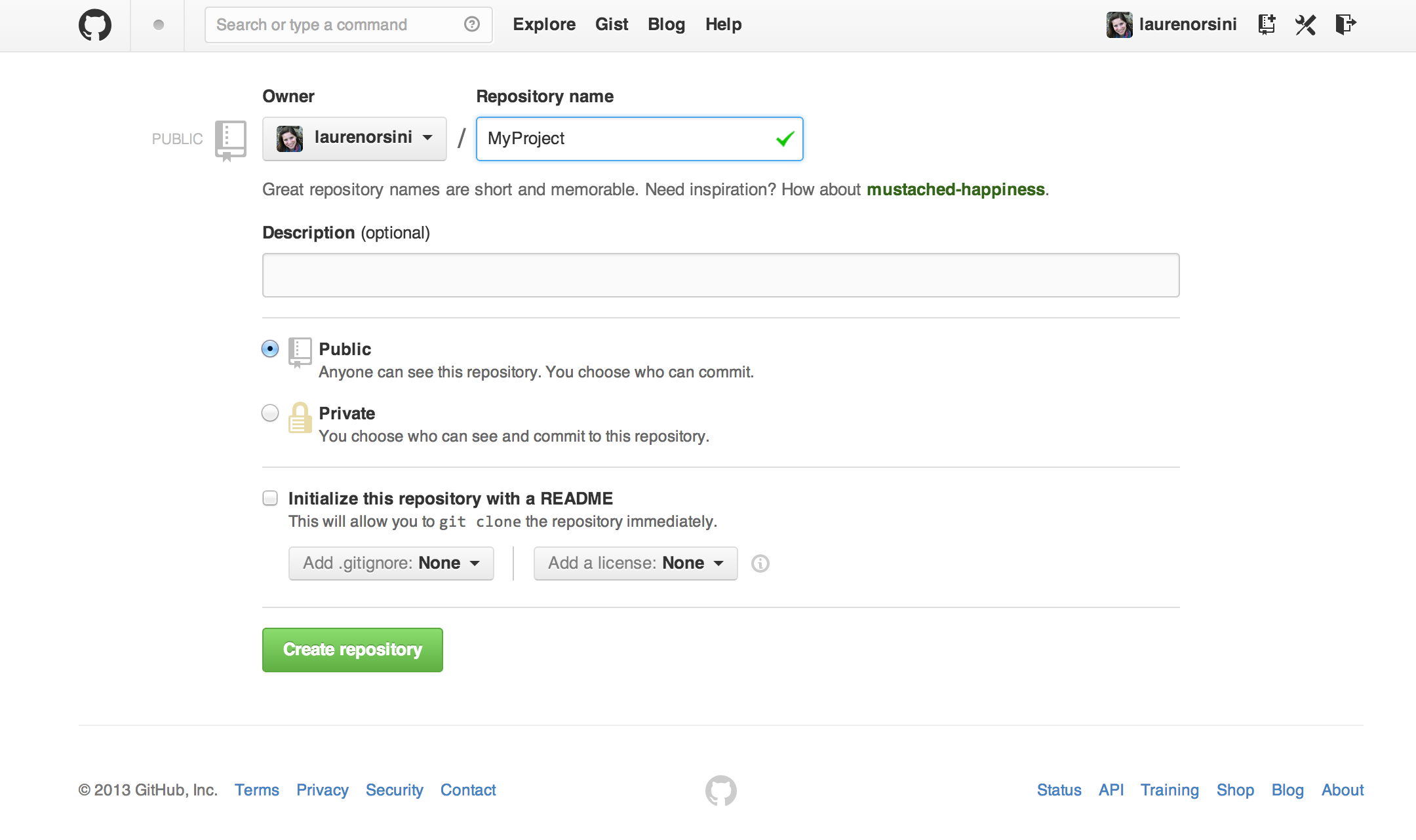
git config --global user.email "your\_email@youremail.com"

That’s all you need to do to get started using Git on your computer. However, since you did set up a GitHub.com account, it’s likely you don’t just want to manage your project locally, but also online. If you want you can also set up Git so it doesn’t ask you to log in to your GitHub.com account every time you want to talk to it. For the purposes of this tutorial, it isn’t a big deal since we’ll only be talking to it once.

## **Creating Your Online Repository**

Now that you’re all set up, it’s time to create a place for your project to live. Both Git and GitHub refer to this as a repository, or “repo” for short, a digital directory or storage space where you can access your project, its files, and all the versions of its files that Git saves.

Go back to GitHub.com and click the tiny book icon next to your username. Or, go to the [new repository page](https://github.com/new) if all the icons look the same. Give your repository a short, memorable name.



Don’t worry about clicking the checkbox next to “Initialize this repository with a README.” A Readme file is usually a text file that explains a bit about the project. But we can make our own Readme file locally for practice.

Click the green “Create Repository” button and you’re set. You now have an online space for your project to live in.

## **Creating Your Local Repository**

So, we just made a space for your project to live online, but that’s not where you’ll be working on it. The bulk of your work is going to be done on your computer. So we need to actually mirror that repository we just made as a local directory.

First type:

mkdir ~/MyProject

mkdir is short for make directory. It’s not actually a Git command, but a general navigational command from the time before visual computer interfaces. The ~/ ensures that we’re building the repository at the top level of your computer’s file structure, instead of stuck inside some other directory that would be hard to find later. If you type ~/ into your browser window, it’ll bring up your local computer’s top-level directory.

I called it MyProject, the very same name I called my GitHub repository that we made earlier. Keep your name consistent, too.

Next, type:

cd ~/MyProject

cd stands for change directory, and it’s also a navigational command. We just made a directory, and now we want to switch over to that directory and go inside it. Once we type this command, we are transported inside MyProject.

Now we’re finally using a Git command. For your next line, type:

git init

You know you’re using a Git command because it always begins with git. init stands for “initialize.” Remember how the previous two commands we typed were general command-line terms? When we type this code in, it tells the computer to recognize this directory as a local Git repository. If you open up the folder, it won’t look any different, because this new Git directory is a hidden file inside the dedicated repository.

However, your computer now realizes this directory is Git-ready, and you can start inputting Git commands. Now you’ve got both an online and a local repo for your project.

**Some Git Commands:**

// checkout branch  
$ git clone “copy the link of your repository”   
git checkout “name of repository”  
git branch  
git branch --all  // to show all branches  
git status // for status  
git add main.js package.json // add files  
git commit -a -m "initial for node" // -a used for commit modified files  
git push origin master // after commit push to that branch  
git pull // for updating branch which are forked forks  
$ git branch  
$ git -b branch node  // for create branch  
$ git checkout -b node // for checkout branch

**SourceTree**

### **What is SourceTree?**

So far you probably used GIT only through a **command line** (Terminal or GIT Shell/Bash). SourceTree is a tool that helps you use GIT **visually**, through a user interface.

### **Download SourceTree**

You already have GIT installed. So, all you need now is to install SourceTree. Go here and **download** it: <https://www.sourcetreeapp.com/>. Install SourceTree.

### **Set-up SourceTree**

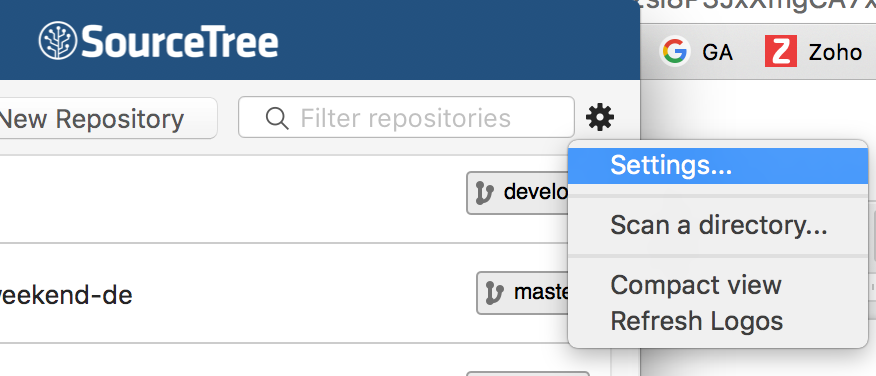
After successful installation, a small window will open - it’s a **repository browser**. Currently it’s empty, but in the future, you’ll have a list of your repositories here.

#### **Set-up your name, email and projects folder**

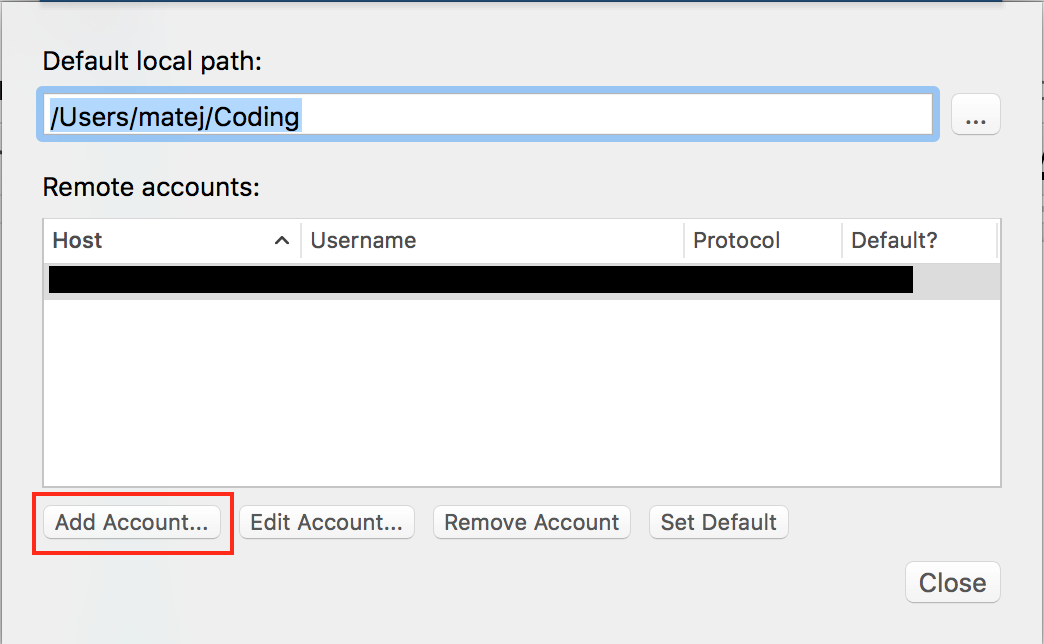
As you already know, GIT**needs your name and email** to sign every commit.

A new window will open. In this window, you can **add your name and email**. Also **select the folder** where you have **all your coding projects** (if you don’t have one already, create it).

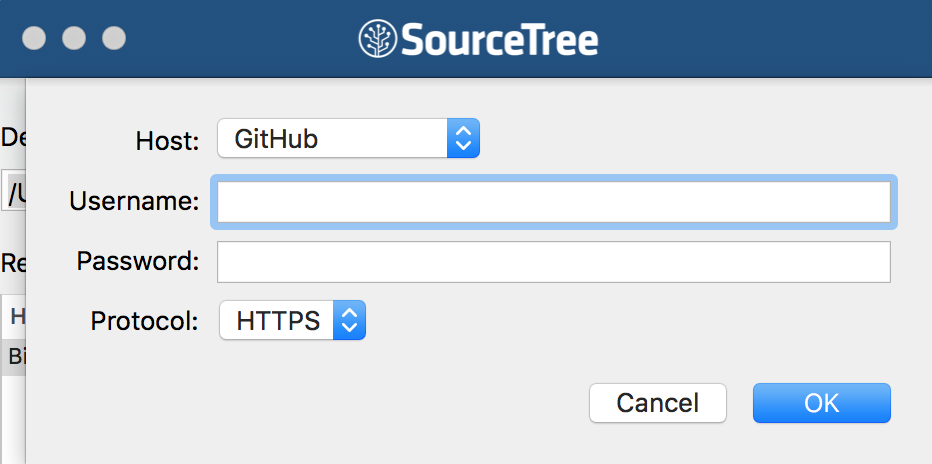
Now you have to connect your GitHub account to SourceTree. Click on the **Settings icon** on the right and select **Settings**.



A new window will appear. Click on **Add account**:

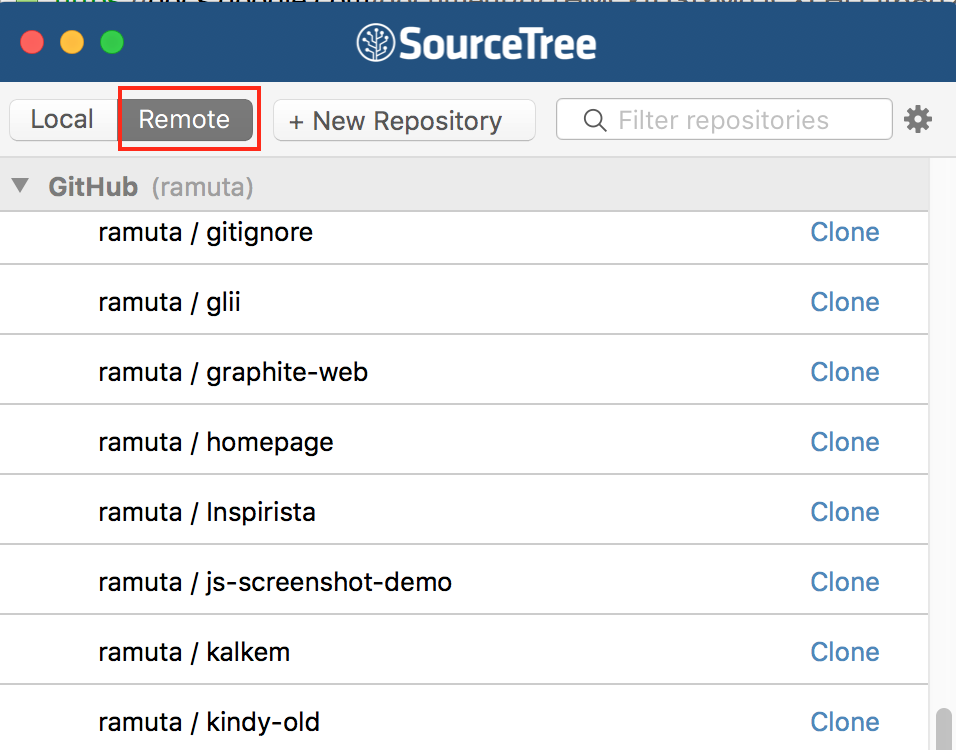


In the next window select **GitHub** and put in your **GitHub username** and **password**:

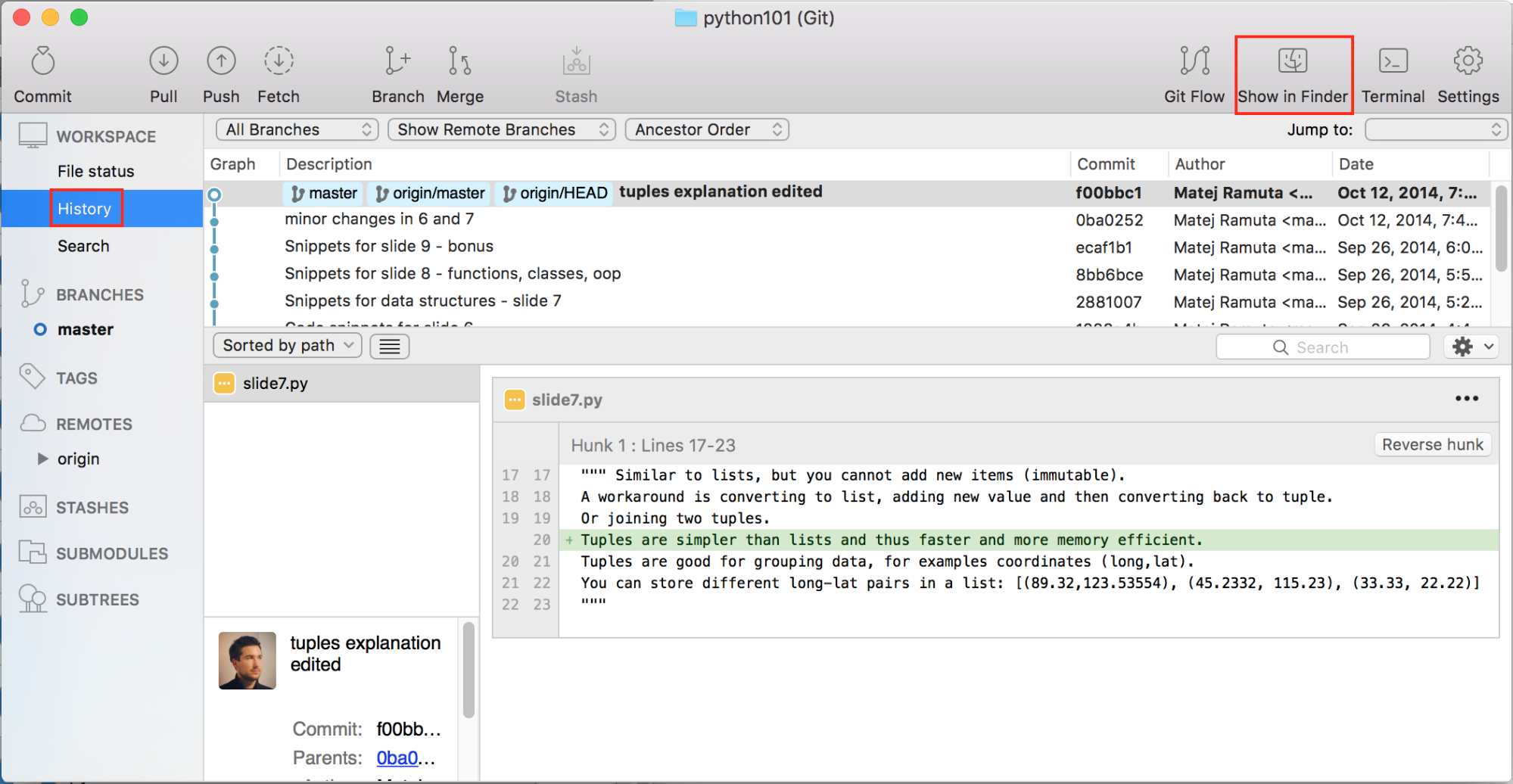


### **Clone a project**

If you already have some projects on GitHub, click on **Remote**and you’ll see a list of your GitHub repositories (if you don’t have any, first create a new repository on GitHub). Next to one of the projects click on **Clone**. This will clone the repository into your projects folder on your computer.

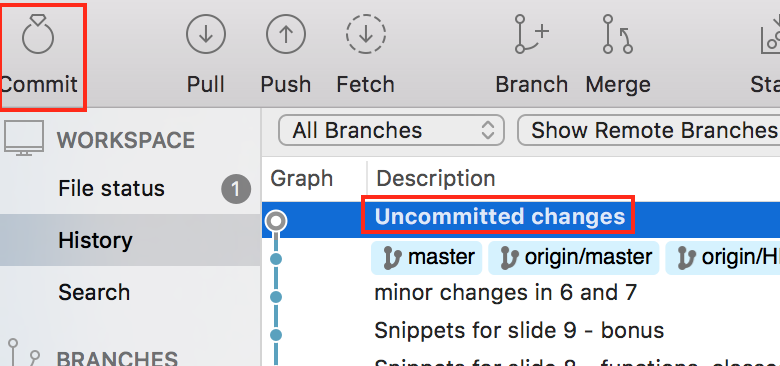


A new window will appear. This window contains a **history** of all the commits that were made for this project so far:

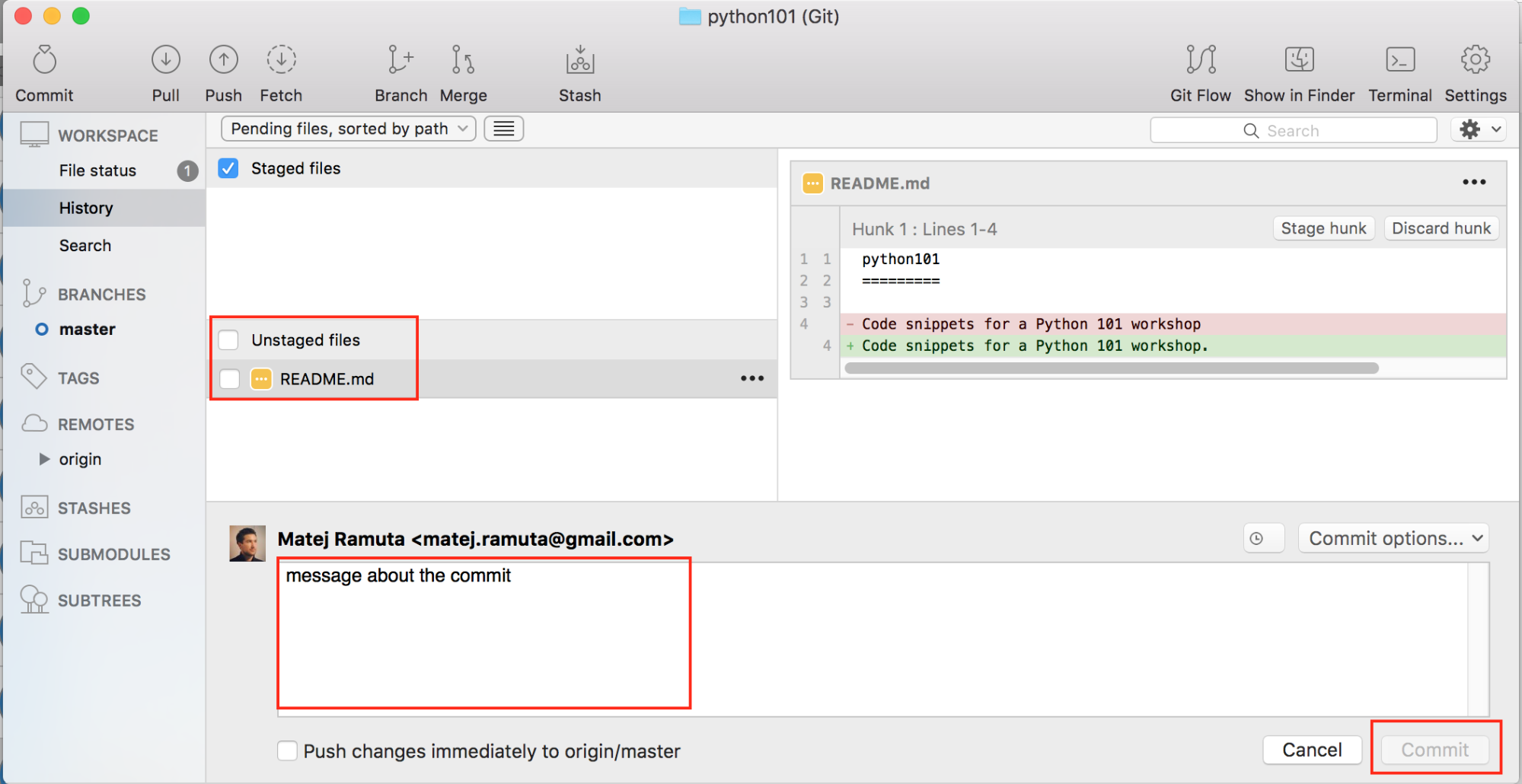


### **Make some changes**

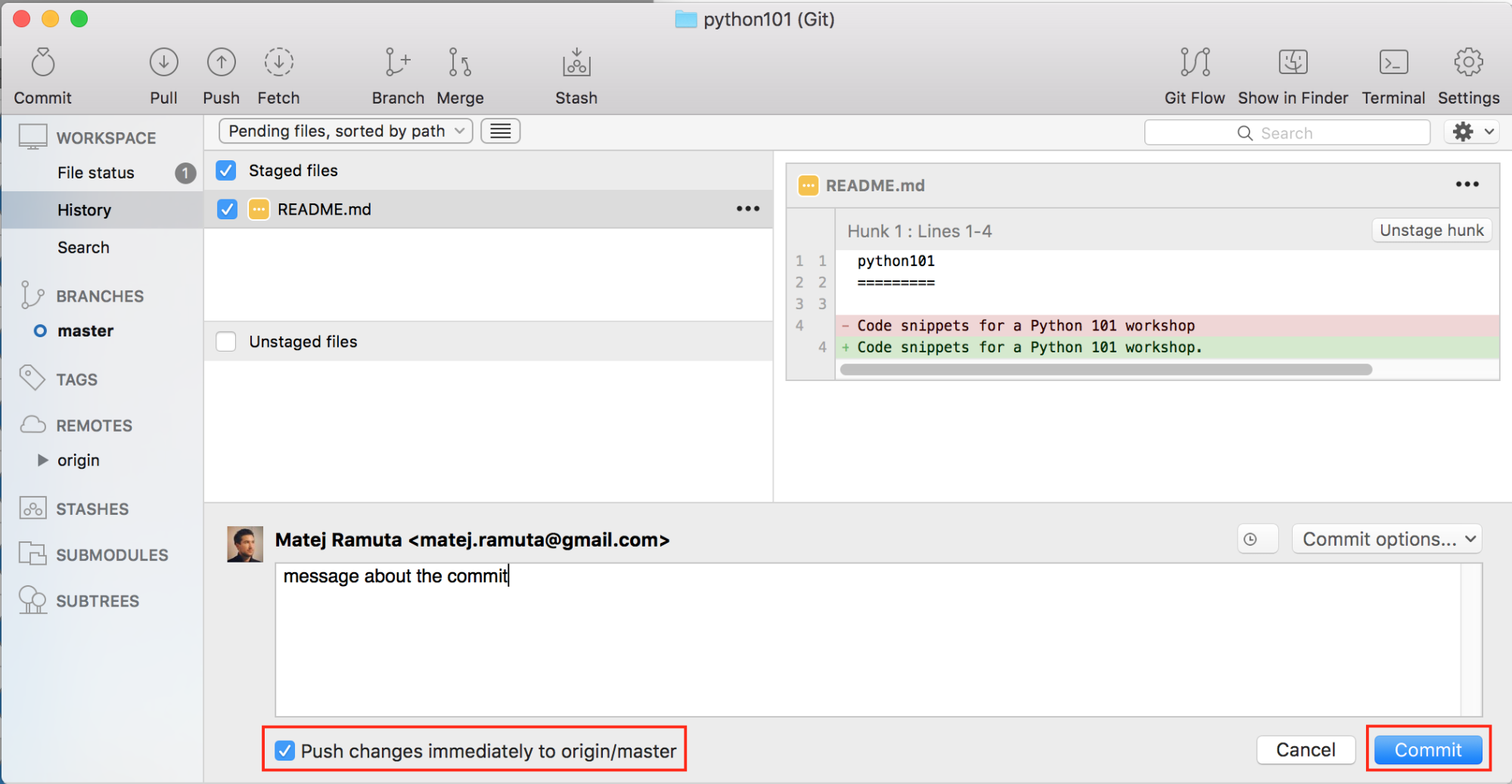
Click on **Show in** Folder. This will open the project folder. Make some smaller change in your project. Then open SourceTree window again:



As you can see, changes were **automatically detected**. Now click on the **Commit**button.



You can **add a Commit message**, but you **can’t click** on a Commit button. Because you have**unstaged files**. You must **select** files that you want to add to your commit (this is basically a git add command). When you select the unstaged files (and make them “staged”), the commit button will become active. But before you click on it, also select **Push changes immediately to origin/master.**



If you forgot to select “Push changes immediately to origin/master”, you’ll have to push the commit to GitHub via the **Push button** (top left).

**Introduction to Biometrics**

**1 Introduction:**

The document covers the information regarding Biometrics Software Integration with Dhis2. It provides installation procedures, technology framework, tools and development process for system.

**2** **Installation**

Two applications need to be set up, Server side application will be installed in server with WAR and config file (biometrics specific) and client side application will be installed on client PC.

**2.1 Server Application Setup:**

***Step 1:*** Software required with its flow of installation:

* JDK/JRE
* Apache Tomcat
* Fingerprint reader driver software

***Step 2:*** Install driver software of fingerprint reader

***Step 3:*** Environment variable Setup:

* Path to reach : My Computer -> Advanced system setting -> Advanced -> Environment Variables -> System Variables -> New
* Fingerprint Server Variable:

Name:  DHF\_HOME

Value: c:\DHF\_HOME

***Step 4:*** Fingerprint Server and dhis.config file configuration setup**:**

* Put WAR file (fingerprint specific) in webapps folder under tomcat folder.
* Go DHF\_HOME folder under c:\DHF\_HOME
* Paste the dhis.config file here containing details for:
* host (ip or dhis domain name)
* port (default is 80)
* dhisUrl (dhis url)

***Step 5:*** Starting Fingerprint server.

* Connect fingerprint reader with machine.
* Double click on startup (window batch file) in (tomcat->bin) folder.
* Initialize server with “identify” request through client application.

Note: Check for log in tomcat logs for fingerprint server. It will initialize with “db-init-success” log.

**2.2 Client Application Setup:**

***Step 1:*** Software required with its flow of installation:

* JDK/JRE
* Fingerprint reader driver software.

***Step 2:*** Install driver software of fingerprint reader.

***Step 3:*** Fingerprint Client cdhis.config file configuration setup

* cdhis.config file should be in same folder as where the jar file of client application.
* host (ip or dhis domain name)
* port (default is 80)
* dhisUrl (dhis url)
* fingerprintUrl (url of finger print server)

***Step 4:*** Starting Fingerprint Client application.

* Connect fingerprint reader with machine.
* Double click on jar file in fingerprint client application folder.
* Login with same username password being used for DHIS2 login