**Github**

**Why command line?**

* There is a strong history with the command line in git. It was originally designed as a command line tool and graphical clients came later.
* As a result, new features make it onto the command line well before they are integrated into a graphical client.
* Nearly all online assistance websites, blogs or other tutorials, use the command line as the standard way of communicating how git does something. There are dozens of graphical clients, but everyone has the same commands in git, with only a few fringe exceptions.
* Also the command line has more power. Most graphical clients only implement the core or common commands, but leave of some very powerful options. Only the command line provides all the raw power of git.
* Going command line allows me one set of commands in git, that will work on windows, the Mac OS, and even Linux.

**Why source control?**

* Source control is a type of backup. It is not just a normal backup, that’s difficult to access or restore.
* Instead it is an ongoing backup of every version of your source you have made, creating a history or a trail of changes.
* This allows you to undo changes that you have made or restore from previous known state.
* You could compare various revisions of your code. For example, what has changed in the last two weeks in the process of moving your exclusive residence of your source from your local system to a shared server, your source can allow collaboration and team work on a project.
* while working on a team, sometimes you need to know who to blame for breaking the latest build or other opportunities for learning
* source control supports the concept of branching off to make changes in isolation. This encourages experimentation, but in a safe way, without immediately impacting ongoing new development.
* Finally, some source control systems provide excellent tools. For reviewing code for collaborative learning and improving the quality of the project.

**Who needs source control?**

* Software developers/ engineers/ programmers. They typically need version control for programming source file, like the ones used for developing ruby, java, Objective C and any other programming languages. however, there are several other types of files that need versioning like UML diagrams especially if they store as XML documents, or SQL scripts or text based configuration files.
* Freelancers often don’t bother with source control. Many feel like it is unnecessary since they don’t work on a team or are at a company that requires this overhead. It is more critical or freelancers to use some type of version control system. Since they are the only ones with a copy of their work and they often don’t have the same support network enjoyed by teams at most companies
* Web designers often use tools that save their mockups as HTML, or in a text compatible format that is great for source control.
* Also any original art for websites should be version controlled like web designers, graphic artists should save their original art and assets in version control. In particular, vector graphics like SVG files, are XML documents or other text format files, that are ideal for version control. To a certain degree, I even support versioning original Photoshop files.
* Finally, anyone planning to share their code or contribute to an open source project, needs to be familiar with source control. Since all major open source projects use it heavily.

**Source control options:**

* There are two main types. They are centralized and decentralized or distributed.
* On the centralized side, we have subversion and CVS which are free and open source options. Commercial offerings include ClearCase, perforce and TFS, as well as many others. The main concept with centralized systems is that there is a centralized server that is the ultimate source of truth for a collection of versioned files. This normally means that a network connection is required to the central server for most operations, including just editing a file.
* On the flip side, the decentralized or the distributed approach, doesn’t mandate a central source of truth, and allows most of the operations to be local. For example, you could get some meaningful work done. while on an airplane crossing the Atlantic Ocean completely disconnected from your company network. Source control system that fit this model include mercurial and git.

**What is Git?**

* Git is a decentralized and distributed version control system. While Git is decentralized, most people still choose to use it with a central server to serve as the main repository for project or a team.
* Because of the distributed nature, Git can scale massively. For example, the creator of Git, Linus Torvalds wanted to create a version control system that could handle the requirements of the Linux project, which he also created.
* Today Linux kernel project contains over 15 million lines of code, with over 1200 worldwide developers contributing to the project, in dozens of active branches.
* Another key benefit of being distributed is that most operations in Git are local. There are only a few commands that require a network connection. Otherwise you can work completely disconnected even performing comparisons and commits completely off-network.
* Since most operations are local, Git is very fast with the same operations.
* Also Git is free and open source, which also contributes to its popularity.
* Git has a very active community, and there are many resources available online.
* Also it is very easy to find developers that already have experienced with Git.

Because of all these factors, Git has become the most popular version control system.

Git “de facto” standard status means that git enjoys wide adoption and integration into other tools used by the development community including text editors, bug tracking systems, and build servers.

**Key concepts:**

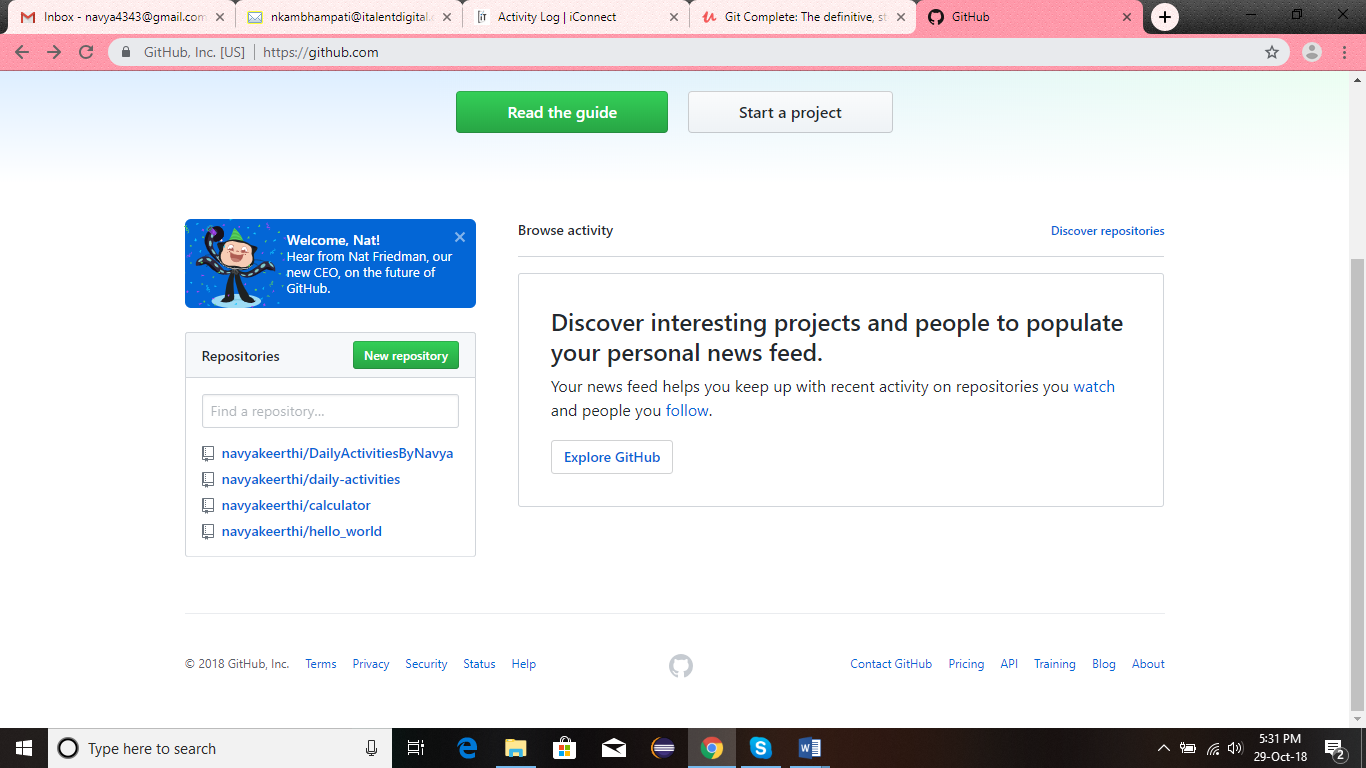
* In Git, collections of version controlled files are kept together in a repository. The repository also contains the history of changes and any special configuration. Generally speaking, a repository would contain all the files related to a specific project or application.
* Next is the three states of Git. The three states related to files being managed by Git. They are the working directory, staging area and the Git repository or the commit history.
* The working directory is the directory or the folder on your computer that holds all the project or application files. Files within the working directory, may or may not be managed by Git. However, Git is aware of them, normally within the working directory is a hidden folder called the “.git” folder that contains the actual git repository.
* The Git repository manages the Git commit history, that is all the changes that are finalized and permanently part of the Git repository.
* In-between is the Git-staging area, often referred to as the Git index, that is a holding area for queueing up changes for the next commit. Since files in the staging area haven’t been committed yet, you can move the files in and out of the staging area without impacting the Git repository and its history of changes.
* The three states of Git are specific to the local Git repository.
* But the fourth state is the remote state. Although the remote repository is just another repository with its own three states internally. Conceptually, I think of the remote repository as a fourth state since it is the last step in the basic Git workflow. And since few people use Git without a corresponding remote repository.
* The last concept is the master branch. Conceptually, branches work like they do in other source control systems. They are timelines that contain your changes. In Git branches contain commits. When we start off Git provides us with a default branch named master.

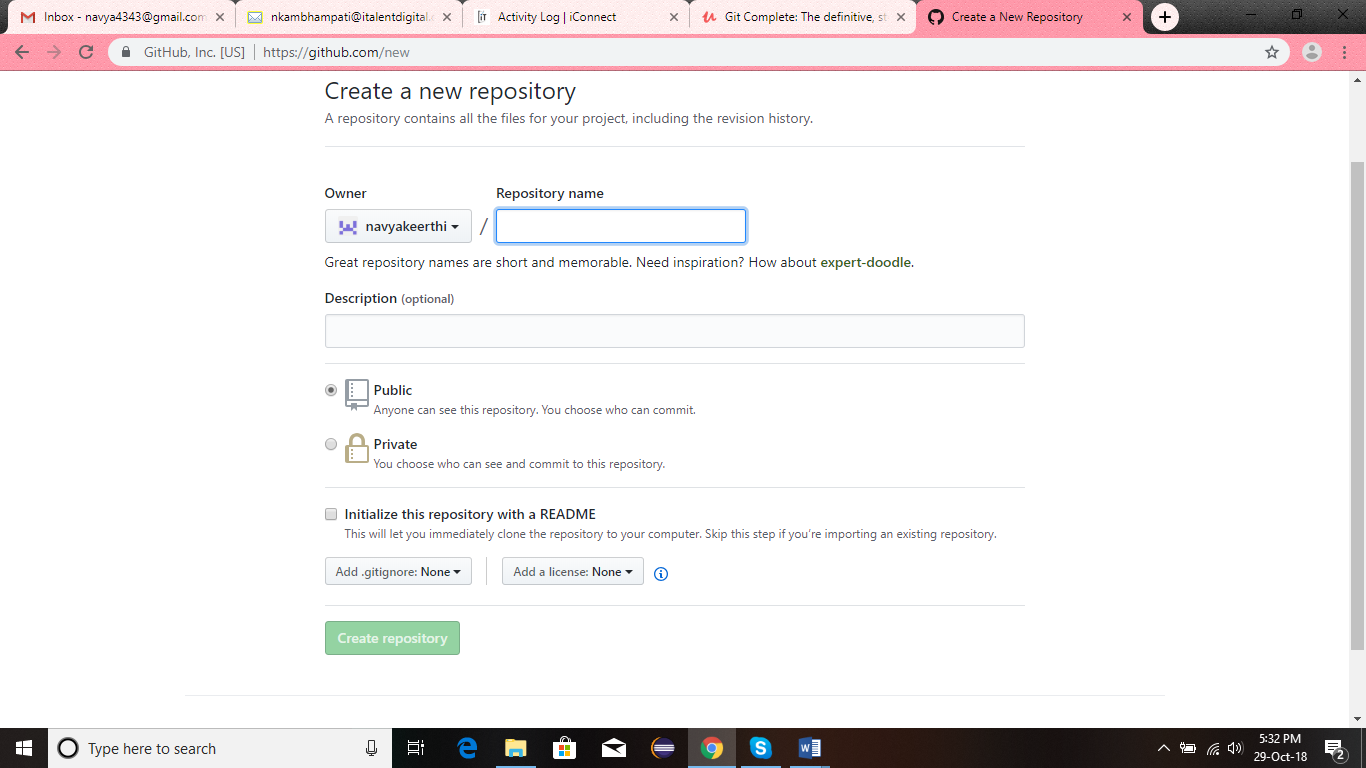
**Git installation:**

* On windows, we will walk through the steps to install and verify Git for windows, which is available from “git-scm.com”. for our interactions online, we are going to use google chrome as my web browser of choice.
* Modern versions of windows come with internet explorer, which will work just fine,
* Let’s start off by invoking our browser, in this case google chrome, then go to “git-for-windows-,github.io”.
* Once the page loads, click on the download button on the home page. This brings us to the page listing all the available downloads for the git for windows installer. You can see here that we have both 32 bit and 64 versions of the installer executables. Since we can run a 64 version of windows, I am going to choose the 64-bit version of Git for windows.
* Chose the version of Git for windows that best matches your operating system. Click on the link for Git installer for my platform, that will begin the download process for the Git installer
* Once the installer has finished the downloading go down and click on “open”.
* That will run the installer directly from the current location. If you are prompted about a security warning, just click on run. Windows also prompt you to make sure that you want to install this program. Click on yes.
* Once the git setup wizard starts, on the first page, click on “next”. The second page has license agreement. Again click on next to agree to the license agreement.
* The next page is select your destination. This is the location that Git will be installed on your system. And click on next.
* On the select components page, I would like the Git bash icon on the quick launch, or taskbar, as well as on the desktop. To get both of these icons I will check, “additional icons” which also checks “in the Quick launch “as well as “on the desktop”. then click on next.
* The next page dictates what parts of Git and Git bash are installed. We are recommended to select middle option. This will install Git bash, as well as integration with the windows command prompt and click on net button.
* Next is the stream regarding how to treat line endings. On windows text files normally end with a different style line ending than corresponding text files on Linux or the Mac operating system. Which option we choose will greatly depend on whether or not you are going to be doing any cross-platform development. We prefer using the middle option, which is to check as-is, that is don’t change anything but then commit Unix style. That means when I use Git commit, the file ending for my text files, will automatically be converted to a Unix style line ending.
* Since I bounce around between windows, Linux and Mac operating system, the middle option makes the most sense for us.
* If we do all our development on windows, and that goes for any team members sharing the same Git repository, then the last option “commit as-is” will be option you will want to select.
* Regarding the terminal emulator to use, we can choose the top option, which is minimal terminal emulator that runs the bash shell environment. Once we have made selection click on next.
* The next page “enables file system caching” is experimental and we are recommended leaving it unchecked and click on next. Now we are installing Git for windows.
* Once the installation process has completed, on the final page we can uncheck to view the README. Then click on finish button.
* Now let’s verify git installation by typing “git version” in git bash and press enter key. Git should respond with the version of Git that s installed.

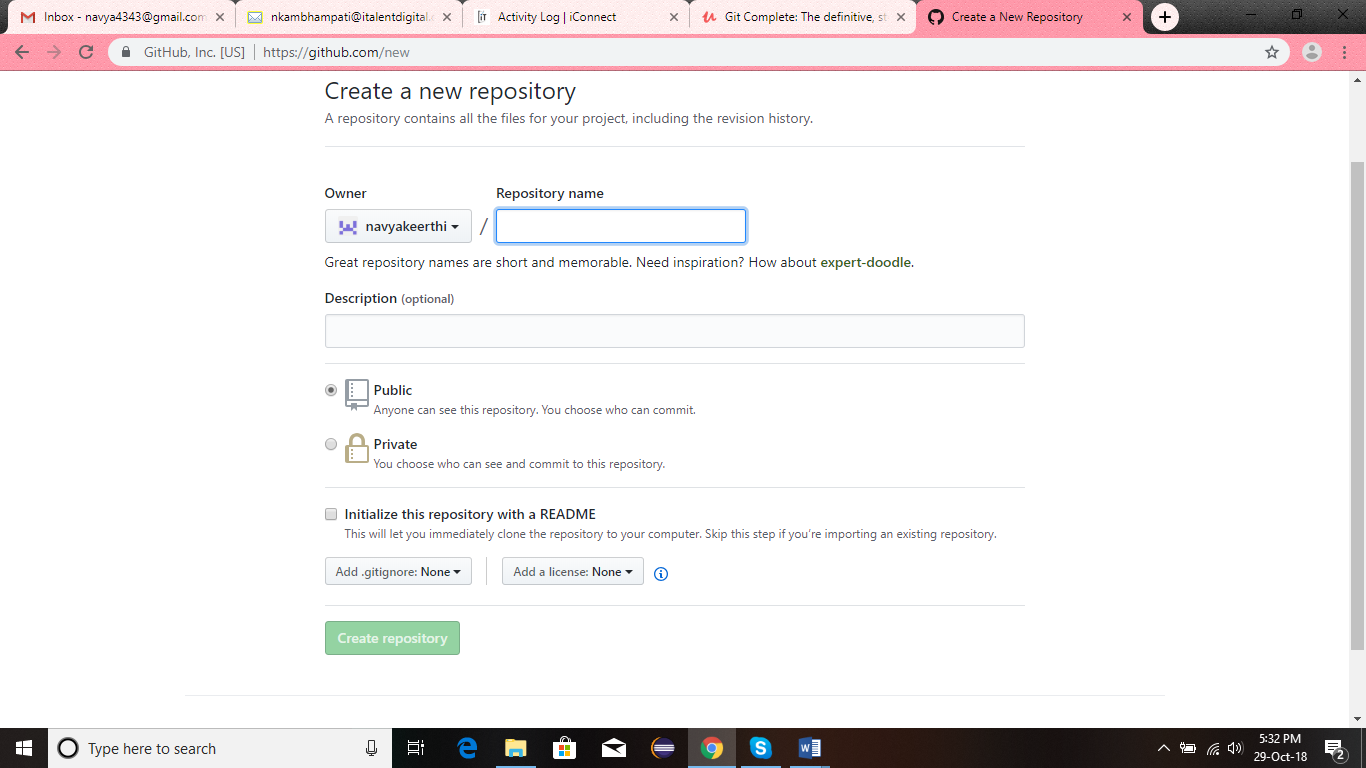
**Creation of new repository:**

* In order to create a new repository, firstly sing in into Github page by signing up to your own account.
* Next click on “new repository” button near left hand side of the page. Once the new repository page loads, you will notice a form asking for a few pieces of information.

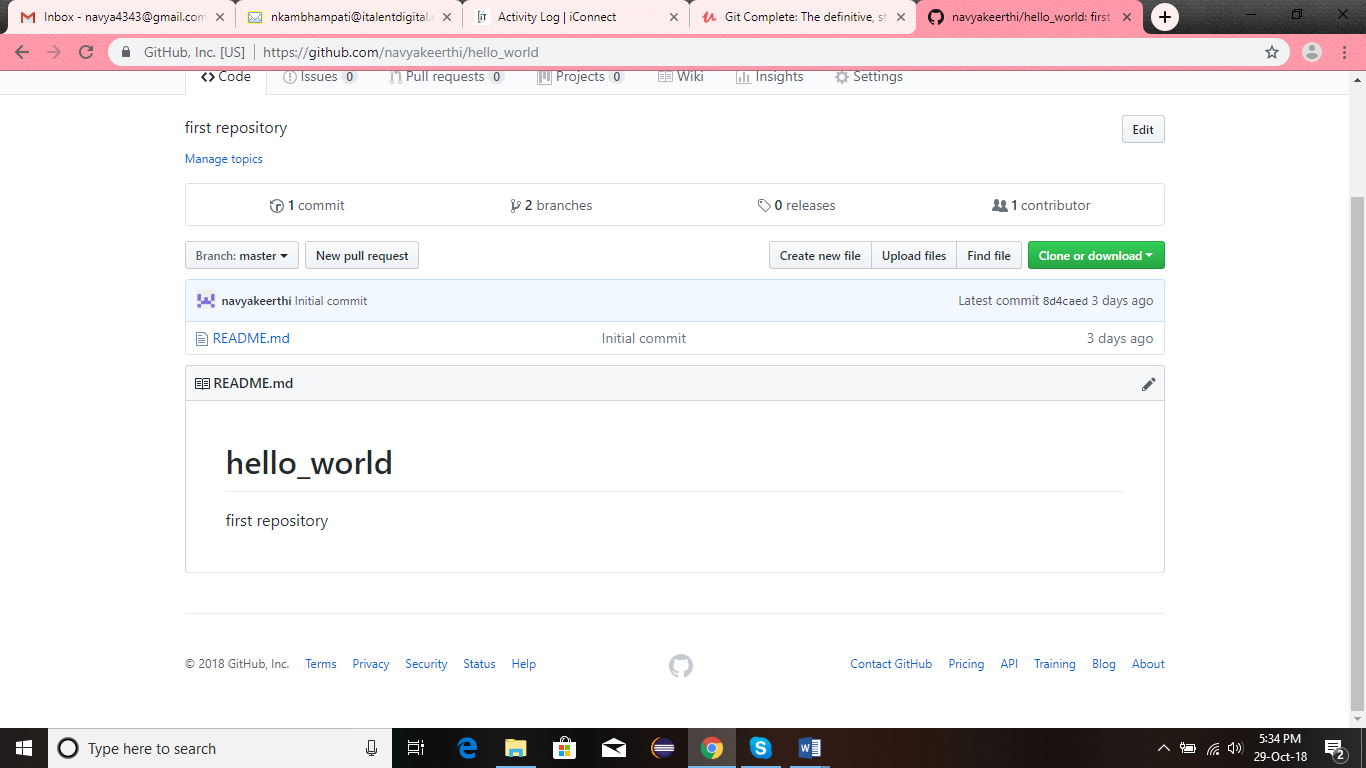




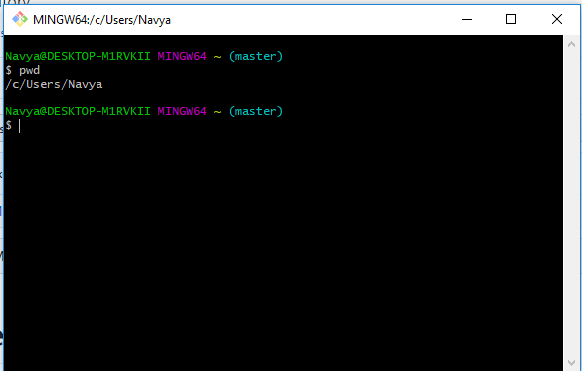
* For new accounts there should only be one owner available in the dropdown, and it should be defaulted to your username.



* “repository name” is a name you wish to give your git repository. I recommend a short, but descriptive name in all lowercase, no spaces and words separated by dashes.
* The description is optional, but certainly a good idea. Leave the repository as public and click “initialize this repository with a README “that will start off our repository and click the button “create repository”.
* Once the repository is created, you will land on the main repository page. Thus we created a new repository. Now we need to prepare our local system before we can proceed.



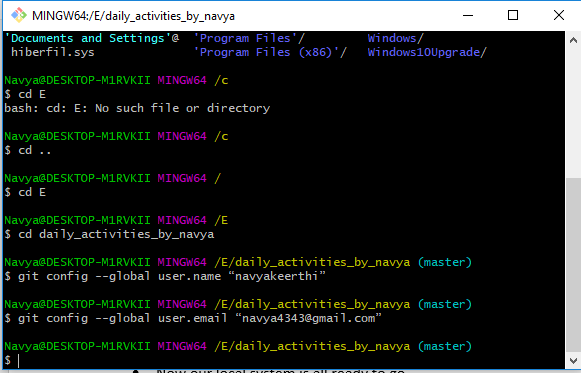
* We are going to prepare our local system through commands. On windows we are going to use Git bash.
* When we open our terminal, by default our terminal program will put us in our home directory, which we can confirm by typing “pwd”.



* Git requires two bits of information before we can do very much. i.e. the name and email address. If we don’t provide it, Git will automatically figure it out.

Command for providing name is ‘git config --global user.name “username” ‘

Command for providing email is ‘git config --global user.email “email’ ‘



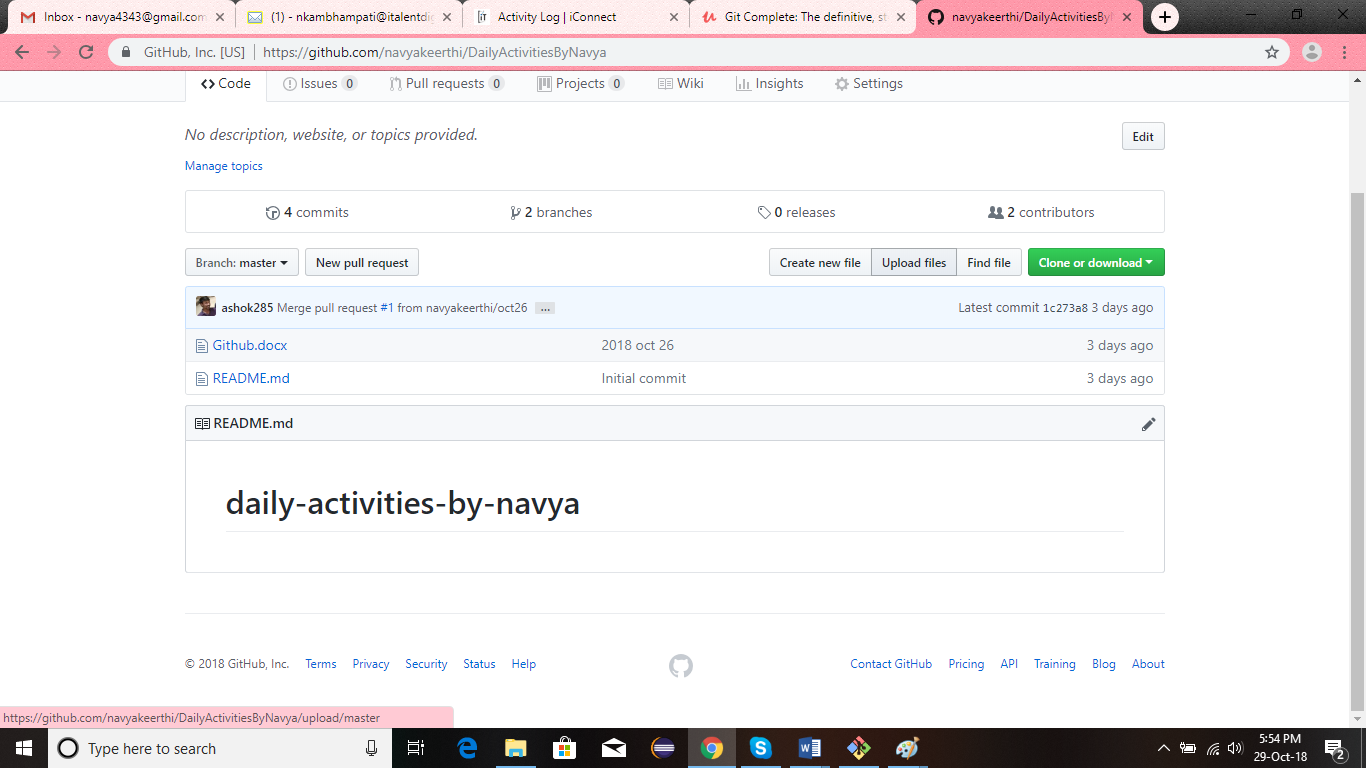
* Now the following command lists back your name and email address we have entered above.

Command “git config –global –list

User.name=username

User.email=email

* Now our local system is all ready to go.
* Now have get a copy of our browser repository onto the local system. This process is known as cloning a repository.
* Locate the clone options at the right top of the repository block. Make sure ‘HTTPS’ is selected and click the copy button to copy the HTTPS URL for the repository to your systems clipboard.



* Return to the bash and type the following command which makes a full copy of our repository on Github to our local system

Command “git clone ‘your copied url’ “and press enter.

* Doing this git will automatically create a directory named after our repository. We could confirm this by typing the following command. You will get your repository created in local system.

Command “ls” and click enter

* Now change to the folder by “cd’ command. When you type “ls” you will see “README” file.
* When we initialized our repository on GitHub, and if we ask Git about state of the repository with “git status”, it tells us that “I am on the master branch”.
* The master branch is the default branch, by convention, for a git repository.
* The git status command also tells us master is “up-to-date” with ‘origin/master’ which refers to the master branch on the Github version of the repository.
* The git clone command automatically set up a relationship back to the repository on Github and named that reference “origin”
* Finally, git tells us the working directory is clean. The working directory is where we do all the work and Git monitors for the changes.
* We use “git status” command to see if there are any changes between the working directory, the staging area and our local repository and our remote repository.
* Now w have our repository on local system.

**Adding a new file:**

* Let’s add a new file using bash commands.
* Create a simple text file. For this type the following command.

“echo “Git Quick Start” >> start.txt “and click enter

* Type ls and you can find the file in the working directory.
* We can use “cat” command to display the contents of the file as such below

“cat start.txt” and press enter

* Now type the command as “git status”. Git says that we have an untracked file. It is just a file in our working directory that has not been added to Git yet. i.e. we haven’t specifically told git about it.
* Now we add the file using the following command.

“git add start.txt”

* Now if you type git status, it tells us there is a new file in staging area, which git describes as ‘changes to be committed”.
* The staging area is designed to allow you to build up several related changes, so they can be committed.
* Now we commit the file using the following command.

“git commit –m “adding start.txt to git” ” and press enter.

* Thus our file is added to the git local repository
* We now should add it to Github by command “push”. The command for pushing the file to browser is as follows.

“git push origin master”

Here origin refers to the Github copy of our repository.

Master refers to our default and only branch in the repository

* Since you are making changes on your repository on Github, the “git push” command will prompt you for your Github username and password.
* Once the push command returns, we can check the result by refreshing the browser.

**Text editor setup:**

For windows we could go with a decent text editor, notepad. We can do much better by using notepad++, which is available for free.

Also for windows, bash alias and git configuration are used.

**Basic commands overview:**

* We will see how to start a project with git in three ways.

1. Starting fresh without source code
2. Starting with existing source code on your local system
3. Joining a project on GitHub by forking and cloning.

* Now we will walk through git basic workflow from adding, editing files, adding them to gits staging area, committing to the git repository and finally pushing and pulling changes back to Github.
* Then we will continue working with our repository with some file management related tasks like renaming, moving and deleting files.
* Once we have made several commits with our repositories we will use git log command to display the repository history.
* Also ignoring the unwanted files.

**Starting a fresh project without source code:**

Step 1: got to the folder of git from the bash

Step 2: initialize a fresh git project as “git init fresh-project”. By doing so git created a folder on the name fresh-project.

Step 3: check it by typing the command “ls”

Step 4: enter into the folder by typing “cd fresh-project”

Step 5: type “ls –al” to list all files and folders including dot files and folders and “–l” tells to list in the listing format. We can see that we have “. git” folder. This is where the git repository lives.

Step 6: type ‘git status” to find the branch in which you are present.

Step 7: create any file as “mate example.txt. paste any paragraph and save using cntrl+s and close it and return to bash.

Step 8: as we are in git repository and we should add the file to git local system. Otherwise git returns untracked file when git status command is given. We could add as “git add example.txt”

Step 9: now we should commit the changes into git as “git commit –m “adding new file””

**Adding git to an existing project:**

To do this firstly we have to down an existing project. Download it from initializr.com.

Step 1: return to the command prompt or bash and go to the git folder.

Step 2: unzip the downloaded folder using the command “unzip ~ /downloads/filename” and press enter.

Step 3: verify the project by typing “ls” command. You will get the folder of the downloaded project

Step 4: now go to the folder by typing “cd foldername”

Step 5: now follow the same steps as mentioned in the fresh project creation.

We are supposed to do as same as above but little change is, the above procedure contains only one file, while the present project is list of files. So we add files to git local repository as “git add .” as “. (period)” indicates the bunch of files.

Other than this everything is same. For removing any file or folder type the command “rm –rf filename or folder name”.

**Joining a project on Github by forking and cloning:**