```
imran@DevOps:~$ ssh-keygen -t rsa -b 4096 -C "imranteli0706@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/imran/.ssh/id_rsa): /home/imran/.ssh/id_rsa_DO_github
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/imran/.ssh/id_rsa_DO_github.
Your public key has been saved in /home/imran/.ssh/id_rsa_DO_github.pub.
The key fingerprint is:
SHA256:AacADjOwSRR7tePN93IUS6UyLhPjrpjaFlkg/QUtMGg imranteli0706@gmail.com
The key's randomart image is:
+---[RSA 4096]----+
|B+=+oo+ .    .   |
|.E.ooo.*    o    |
|+.o.oo+o.o +     |
|  . .o= +.+ o    |
|    o. *So o     |
|   o  . + o      |
|    .  . . o     |
|  ..o .   o      |
| .o+ .           |
+----[SHA256]-----+
imran@DevOps:~$
```

Set SSH private key for github.com login

1. Go to users ssh directory
 # cd ~/.ssh
 # ls

2. Open or create config file and update it with below mentioned content
 # vi config

Host github.com
HostName github.com
IdentityFile ~/.ssh/id_rsa_DO_github
User git
IdentitiesOnly yes

```
imran@DevOps: ~/.ssh
imran@DevOps:~$ cd .ssh/
imran@DevOps:~/.ssh$ ls
config  id_rsa  id_rsa_DO_github  id_rsa_DO_github.pub  id_rsa.pub  known_hosts
imran@DevOps:~/.ssh$ cat config
Host github.com
  HostName github.com
  IdentityFile ~/.ssh/id_rsa_DO_github
  User git
  IdentitiesOnly yes
imran@DevOps:~/.ssh$
```

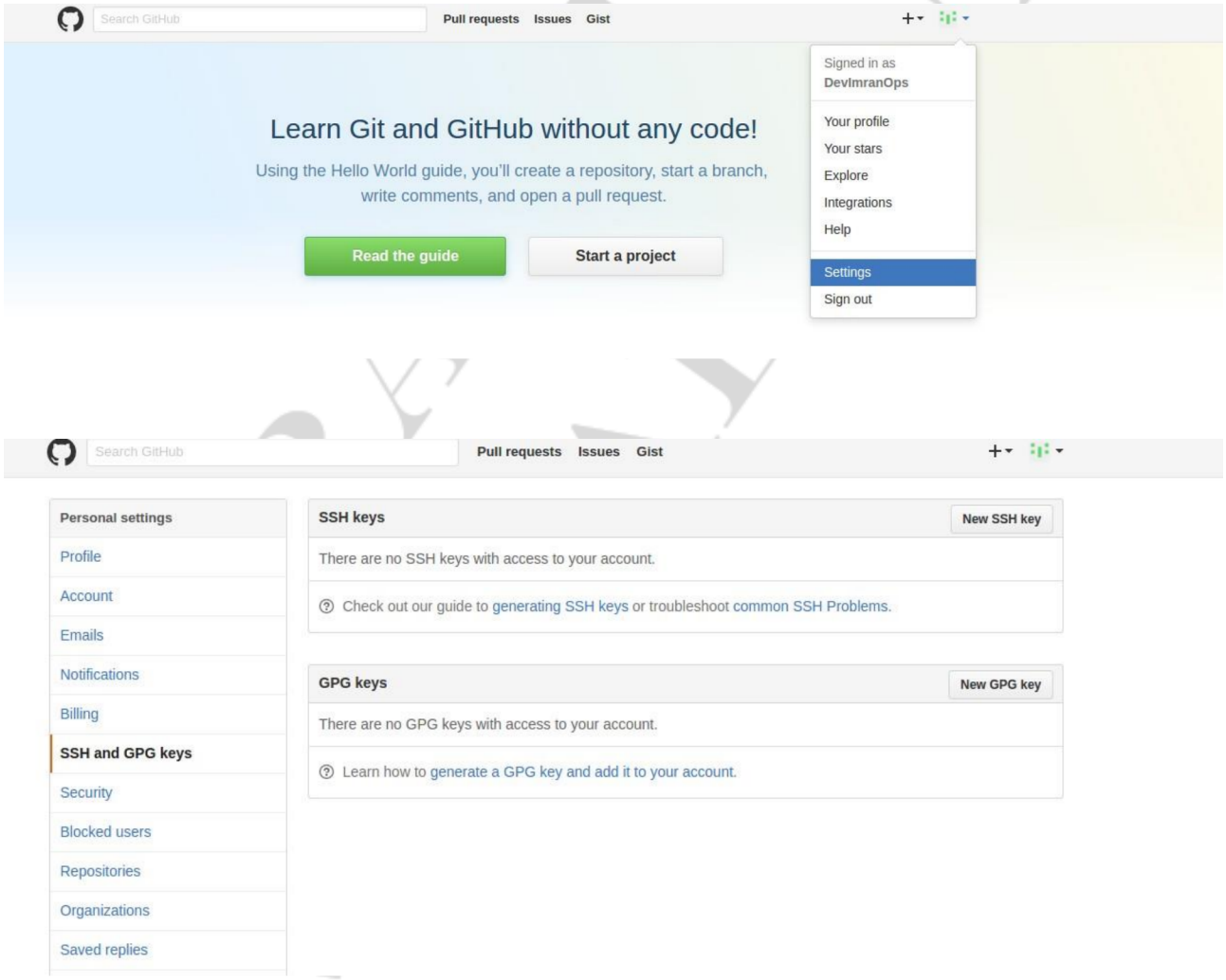
Add SSH public key in github account.

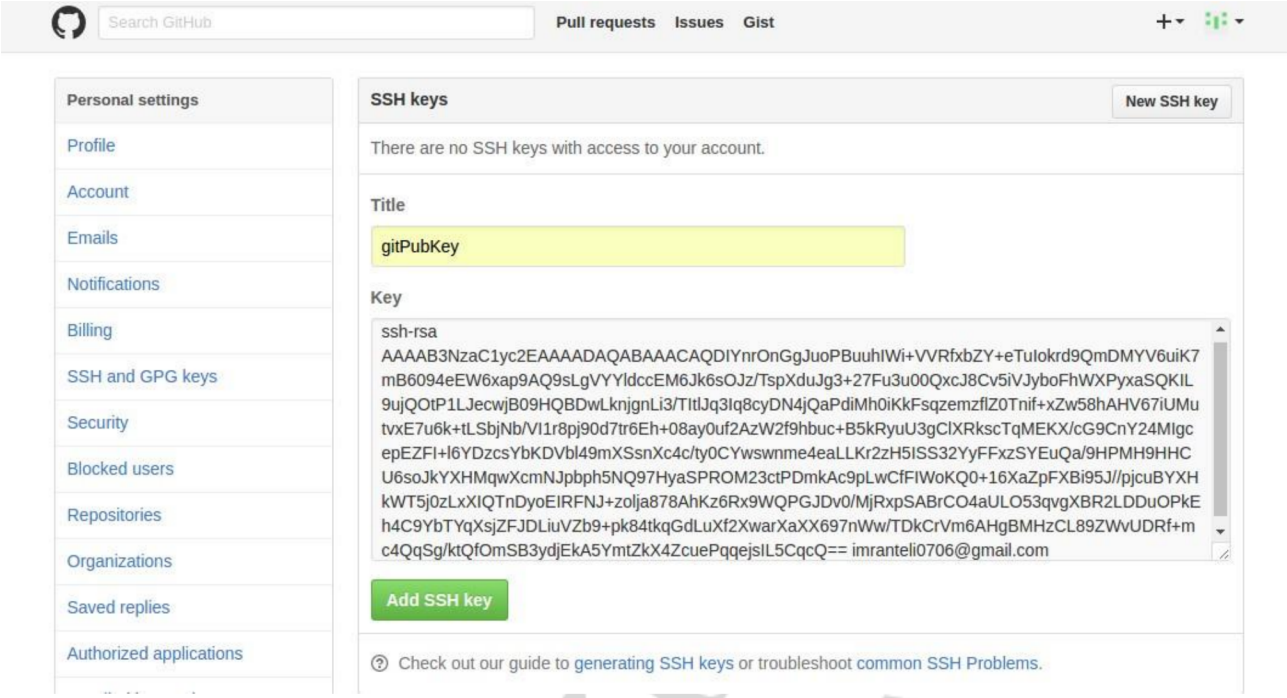
1. Copy public key

 # cat ~/.ssh/id_rsa_DO_github.pub

2. Login to github with the same email id you provided while create ssh keys

Click on settings => SSH and GPG keys => New SSH key => Give a name => Paste the public key content => Add SSH keys



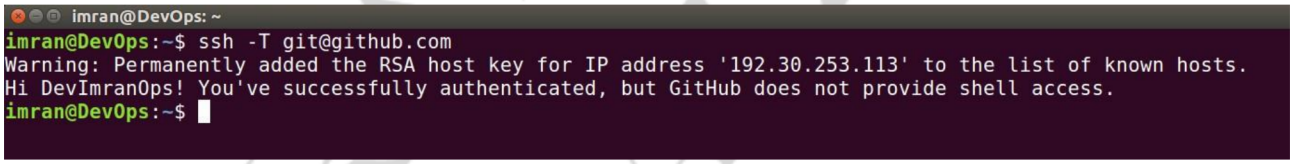


3. Test the login

 # ssh -T git@github.com

You should get a reply as below

Hi <Username>! You've successfully authenticated, but GitHub does not provide shell access.



19. GIT Cheat Sheet from Atllasian

Git Basics

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at <repo> onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use --global flag to set config options for current user. +
`git add <directory>`	Stage all changes in <directory> for the next commit. Replace <directory> with a <file> to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use <message> as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options. +
`git diff`	Show unstaged changes between your index and working directory +

Undoing Changes

`git revert <commit>`	Create new commit that undoes all of the changes made in <commit>, then apply it to the current branch.
`git reset <file>`	Remove <file> from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes. +
`git clean -n`	Shows which files would be removed from working directory. Use the -f flag in place of the -n flag to execute the clean.

Rewriting Git History

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto <base>. <base> can be a commit ID, a branch name, a tag, or a relative reference to HEAD. +
`git reflog`	Show a log of changes to the local repository's HEAD. Add --relative-date flag to show date info or --all to show all refs.

Git Branches

`git branch`	List all of the branches in your repo. Add a <branch> argument to create a new branch with the name <branch>.
`git checkout -b <branch>`	Create and check out a new branch named <branch>. Drop the -b flag to checkout an existing branch.
`git merge <branch>`	Merge <branch> into the current branch.

Remote Repositories

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use <name> as a shortcut for <url> in other commands.
`git fetch <remote> <branch>`	Fetches a specific <branch>, from the repo. Leave off <branch> to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy. +
`git push <remote> <branch>`	Push the branch to <remote>, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist. +

git config

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias.<alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline"`
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. <editor> arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

git log

`git log -<limit>`	Limit number of commits by <limit> . E.g. "git log -5" will limit to 5 commits
`git log --oneline`	Condense each commit to a single line.
`git log --stat`	Include which files were altered and the relative number of lines that were added or deleted from each of them.
`git log -p`	Display the full diff of each commit.
`git log --author="<pattern>"`	Search for commits by a particular author.
`git log --grep="<pattern>"`	Search for commits with a commit message that matches <pattern>.
`git log <since>..<until>`	Show commits that occur between <since> and <until>. Args can be a commit ID, branch name, HEAD, or any other kind of revision reference.
`git log -- <file>`	Only display commits that have the specified file.
`git log --graph --decorate`	--graph flag draws a text based graph of commits on left side of commit msgs. --decorate adds names of branches or tags of commits shown.

git diff

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit.

git reset

`git reset`	Reset staging area to match most recent commit, but leave the working directory unchanged.
`git reset --hard`	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
`git reset <commit>`	Move the current branch tip backward to <commit>, reset the staging area to match, but leave the working directory alone.
`git reset --hard <commit>`	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** <commit>.

git rebase

`git rebase -i <base>`	Interactively rebase current branch onto <base>. Launches editor to enter commands for how each commit will be transferred to the new base.

git pull

`git pull --rebase <remote>`	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

git push

`git push <remote> --force`	Forces the `git push` even if it results in a non-fast-forward merge. Do not use the --force flag unless you're absolutely sure you know what you're doing.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	Tags aren't automatically pushed when you push a branch or use the --all flag. The --tags flag sends all of your local tags to the remote repo.

Summary:

22. Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares.
23. SDLC model are suitable for their project and it would also help the developers and testers understand basics of the development model being used for their project.
24. SDLC is an organized and time bounded way of doing Development, testing and release
25. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates. There are various SDLC models like Waterfall, Spiral, iterative Agile and few other.
26. Version control systems are a category of software tools that help a software team manage changes to source code over time.
27. Version control software keeps track of every modification to the code in a special kind of database.
28. Git is a Distributed VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source.
29. Git a retain long-term change history of every file.
30. Traceability, being able to trace each change made to the software and connect it to project management and bug tracking software such as JIRA.
31. Branching and merging. Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of changes.

X. Maven

Apache Maven is a java based software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Maven provides d a complete build lifecycle framework for project management. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

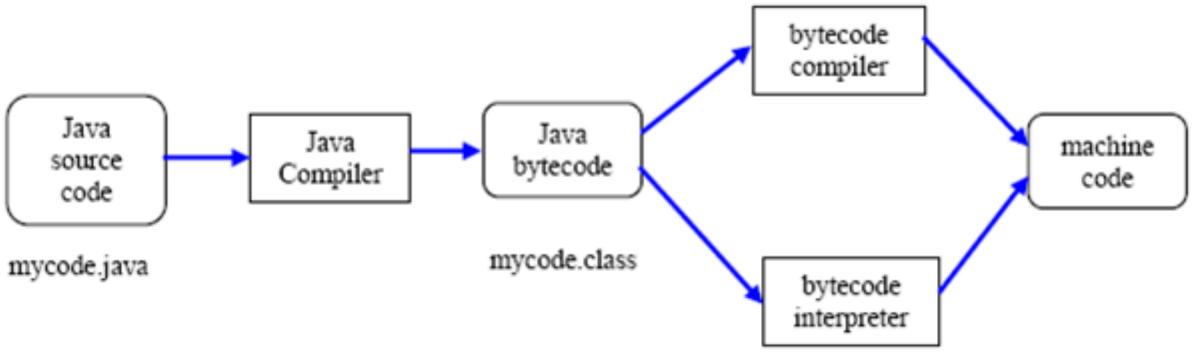
In abstract, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of build related tasks.

1. Build process

The "Build" is a process that covers all the steps required to create a deliverable product of your software into preproduction and production. In the Java world, this typically includes:
Generating source.
Compiling sources.
Executing tests (unit tests, integration tests, etc).
Packaging (into jar, war, ejb-jar, ear).
Running health checks (static analyzers like Checkstyle, Findbugs, PMD, test coverage, etc).
Generating reports.
A defined build process is an essential part of any development cycle because it helps close the gap between the development, integration, test, and production environments. A build process alone will speed the migration of software from one environment to another. It also removes many issues related to compilation, classpath, or properties that cost many projects time and money.

Compilation and execution

Compilation and execution of a Java program is two step processes. During compilation phase Java compiler compiles the source code and generates bytecode. This intermediate bytecode is saved in form of a .class file. In second phase, Java virtual machine (JVM) also called Java interpreter takes the .class as input and generates output by executing the bytecode.



Various build tools available:

12. For java - Ant,Maven,Gradle.
13. For .NET framework - NAnt
14. c# - MsBuild.

2. Ant vs Maven vs Gradle:

Ant

Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications. Ant can also be used effectively to build non Java applications, for instance C or C++ applications. More generally, Ant can be used to pilot any type of process which can be described in terms of targets and tasks.

It has very low learning curve thus allowing anyone to start using it without any special preparation. It is based on procedural programming idea. After its initial release, it was improved with the ability to accept plug-ins.

Major drawback was XML as the format to write build scripts. XML, being hierarchical in nature, is not a good fit for procedural programming approach Ant uses. Another problem with Ant is that its XML tends to become unmanageably big when used with all but very small projects.

Maven

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Its goal was to improve upon some of the problems developers were faced while using Ant. Maven continues using XML as the format to write build specification. However, structure is diametrically different.

While Ant requires programmer to write all the commands that lead to the successful execution of some task whereas Maven relies on conventions and provides the available targets (goals) that can be invoked. As the additional, and probably most important addition, Maven introduced the ability to **download dependencies over the network** (later on adopted by Ant through Ivy). Main benefit from Maven is its life-cycle. As long as the project is based on certain standards, with Maven one can pass through the whole life cycle with relative ease. This comes at a cost of flexibility.

Now the interest for DSLs (Domain Specific Languages) continued increasing. The idea is to have languages designed to solve problems belonging to a specific domain. In case of builds, one of the results of applying DSL is Gradle and for e.g. gradle is used in Android for build and packaging.

Gradle

Gradle aims to help organizations ship better software, faster. Faster builds is one of the most direct ways of achieving this;Gradle combines good parts of both tools and builds on top of them with DSL and other improvements. It has Ant’s power and flexibility with Maven’s life-cycle and ease of use. For example, Google adopted Gradle as the default build tool for the Android OS.

Gradle does not use XML. Instead, it had its own DSL based on Groovy (one of JVM languages). As a result, Gradle build scripts tend to be much shorter and clearer than those written for Ant or Maven. The amount of boilerplate code is much smaller with Gradle since its DSL is designed to solve a specific problem: move software through its life cycle, from compilation through static analysis and testing until packaging and deployment.