Java Basics

Lesson 1: Introduction to Git

SOFTWARE GUILD

Credits and Copyright

Copyright notices

Copyright © 2016 by The Learning House.
All rights reserved. No part of these materials may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of The Learning House. For permission requests, write to The Learning House, addressed “Attention: Permissions Coordinator,” at the address below.

The Learning House
427 S. 4th Street #300
Louisville KY 40202

Lesson 4: Git Quick Start

Java Cohort

Lesson 1 : Introduction to Git

Overview

Source control is an integral part of the software development process. The proper use of source control can be the difference between having a copy of files when they appear to be lost and really losing the files. Hard drive failures, files being accidentally overwritten and other catastrophes are hard to avoid, they just happen sometimes. Source control management systems are designed to keep history and revision information about the files and projects stored within a repository.

By storing this information we have a complete history of everything that happened to the projects within the repository. In the event of a catastrophe we can recover previously saved version. Also by saving our changes we can easily share work with other developers and collaborate on projects regardless of location. It is advised that you always use some kind of source control for all projects. Git is a great easy way to get started with source control and continue its use into the enterprise and into large scale projects. It grows with your needs as you need it to.

History of Git

Git is a distributed revision control and source control management system created by Linus Torvalds in 2005. At the time Linus, the creator of Linux, was looking for a new source control system for the Linux kernel. Linus set out in search of a system that would meet all of his requirements for the Linux kernel. One by one every system failed to meet his requirements and hence he decided to create his own revision control system.

Git, the name, comes from the British English slang for an unpleasant person. Linus explained this by stating he was an egotistical bastard and that all of his projects are named after himself, Linux and now Git. Git was started in April 2005 and by June 2005 it was managing a version of the Linux kernel. This project has continued to grow and be the source control management system of choice for numerous projects and companies and remains one of the most popular today.

Git Goals

The Git project, as stated, was designed with the Linux kernel in mind. This meant the project had specific goals in mind related to that type of project. In looking at the Linux kernel, the major goal was to support the development of a large-scale application where the developers would not be in the same location. In a distributed environment the developers would need to

be able to work together and apply changes and be able to merge each other's changes easily. Git then emphasizes the need for an easy way to merge and branch the code.

In an effort to gain acceptance the system would need to follow some standard protocols, such as HTTP and FTP. This would allow storage of repositories in a variety of locations including services and internet host providers. The system would also need to work well with existing systems in cases where there was already a system in place but Git could offer a way to extend the project to a new level. All of this and more makes up the foundations of the Git functionality today.

Why Choose Git?

There are many reasons for choosing Git. Git is cross-platform thus working on Windows, Mac and Linux alike. Git is an open source project and does not have any licensing costs for using Git. You may need licensing if you choose to use an established online provider or if you choose to purchase third party tools but the system itself does not have licensing costs. Git is also easy to learn and can be very simple only taking advantage of complex features if you need them or as you grow and require more of their use. With the flexibility and availability it makes for a very compelling choice.

The Git Workflow

One way we like to describe Git to new learners is that its common usage is like a cloud storage provider such as Dropbox or Google Drive, except that all of the syncing (upload/download) is done manually by the user.

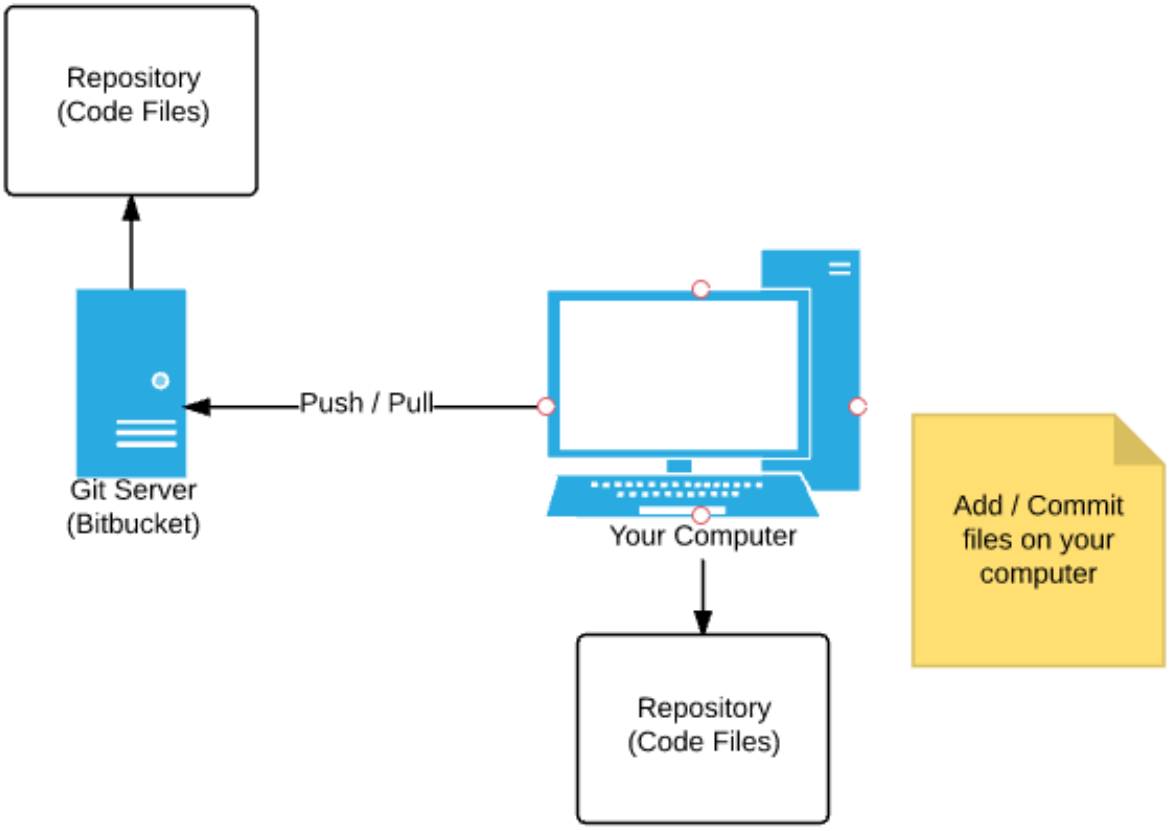
When setting up a Git repository on your computer, one of the steps will be to bind a local folder on your hard drive to a repository on the server. Once the repository is bound, you are ready to start creating and modifying your code files. To work with files and sync them up to the server, the workflow looks like this:

1. **Add:** One difference between Git and a program like Dropbox is that where Dropbox will automatically track and sync any file in its directory, Git will only track and sync files you tell it to. Oftentimes, developers will have files that should not be shared, such as files that contain passwords or other secure data, so by default Git will not sync files unless you tell it to using the **Add** command.
   The `git add` command adds a change to the staging area and lets Git know that you want to include updates to a particular file in the next commit.
2. **Commit**- Commit takes any files in the staging area that were added and creates a snapshot that is ready to be copied to the server. Committing a snapshot also requires a message that will be shown in the project history along with the commit. The message is meant to tell other developers a summary of the changes made.

   As a Git user, you can add files in bulk or one at a time. Many adds roll up to a single commit.

3. **Push**- The push command transfers commits from your computer to the remote server. The key phrase here is remote server. The add and commit commands work on your local computer and are concerned with the files there. Push is for sharing your code with the rest of your team by deploying it to the server.
4. **Pull**- The pull command takes any changes that other people have pushed to the server and brings them down to your local folder. Pull will attempt to synchronize the files on your machine with the files on the server

   Not only will the pull command copy files from the server to your computer, but in the case where you have edited a file that changed on the server, it will attempt to merge them. If it cannot merge them, it will notify you that you have a **conflict**, and you will have to manually fix the file.



The Add-Commit-Push workflow is the common workflow you will be using in this part of the course. You are not currently sharing your repository with other collaborators, but using it to track your changes and allow your instructors to view your progress on your work.

In a real project where you have other collaborators, you will pull their changes down frequently so that you always have the latest and greatest version of the code project. When you are working alone on a single machine, you will not need the pull command. If you have

multiple computers at home and want your project work on all the computers, you will need to pull on the other computers to keep everything in sync.