COURSEWARE

Professional Skills

Agile Fundamentals

Jira

Git

- Introduction to Source Control
- Basics
- Cloning
- Forking
- Branching
- Merging
- Reverting
- GitHub Pull Requests
- GitHub Reviews
- GitHub Actions

Databases Introduction

Java Beginner

Maven

Testing (Foundation)

Java Intermediate

HTML

CSS

Javascript

Spring Boot

Selenium

Sonarqube

Advanced Testing (Theory)

Cucumber

MongoDB

Express

NodeJS

React

Basics

Contents

- Overview
- Basic Workflow
- Common Commands and Concepts
  - Cloning a Repository (`git clone`)
  - Staging a Change (`git add`)
  - Local Repository Status (`git status`)
  - Commiting a Change (`git commit`)
  - Pushing Changes (`git push`)
  - Retrieving Remote Changes
  - Ignoring Files
- Tutorial
- Exercises

Overview

Git is used for tracking changes in source code during software development. It is designed for coordinating work amongst programmers, but it can be used to track changes in any set of files.

Basic Workflow

The basic workflow for using Git includes staging, committing and pushing changes.
Before a change can be committed it must be staged and to apply your changes for everyone else on the team, the changes must be pushed to the remote repository.

Common Commands and Concepts

As we know, Git is an incredibly popular tool and there are some commands that are likely to be encountered most days when using it in a basic project workflow.

Cloning a Repository (`git clone`)

To download a remote repository you can use the `git clone` command and provide the URL of the remote repository.
The clone command is very useful because it configures the local repository for you, the remote repository is automatically configured for when you need to push your new changes to it.

```
# git clone [REPOSITORY_SSH_URL]
git clone git@github.com:bob-crutchley/vim.git
```

Staging a Change (`git add`)

Staging is the step that you must take before commiting a change.
Staging is a feature in Git that enables the developer to choose what changes are actually going to get committed to the repository when the commit is made. There is a few ways you stage files:

Express-Testing
Networking
Security
Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

```
# stage all files
git add --all

# stage all files (only if you are at the root of your project)
git add .

# stage selected files: git add [FILES]
git add file_1.txt file_2.txt
git add *.txt
```

Local Repository Status (`git status`)

Knowing the current state of your local repository is very useful so that you can understand what commands to run.
For instance, how can you know what files have been staged and not staged?

```
git status
```

Commiting a Change (`git commit`)

When you make a commit to a Git repository, you are effectively "saving" the changes that you have staged to the repository.
Unlike saving files in most other programs, Git also requires a message to be saved against the commit along with some basic information about the user from the Git config shown above.
What you put in this message is important, so that you understand what it is that you changed on that particular commit.
Commits can be reverted, so it helps when there is a concise message about what was implemented or removed.

```
# git commit -m "[COMMIT_MESSAGE]"
git commit -m "initial commit"
```

Pushing Changes (`git push`)

To apply the changes that you have made in a remote repository you must push them to that remote repository.
Only changes that have been committed will be pushed to the remote repository.
We can use `git push` and provide the remote repository (default is origin) and remote branch to push our changes.

```
# git push -u [REMOTE] [REMOTE_BRANCH]
git push -u origin main
```

Retrieving Remote Changes

Because Git is a collaborative tool, other people could have made changes to code in the remote repository, these changes will need to be pulled down to avoid conflicts in the code.

```
# git pull [REMOTE] [REMOTE_BRANCH]
git pull origin main
```

Ignoring Files

Git segregates all files in your local repository into "tracked" (files which have been staged or committed), "untracked" (files which have not been staged) or "ignored" (files which will be ignored).
We often do not want some files in our remote repository.
This could be because they are large or unnecessary (e.g. compiled code, build output directories, dependency caches).
In order to make sure that Git does not track them, we need to create a .gitignore file in our repository.
This file is a list of intentionally untracked files that Git should ignore.
We can use an asterisk "*" to specify which type of files should be ignored (an asterisk matches anything except a slash).

For example, if we use "*.pyc", *Git will ignore all files that end in ".pyc".*
A leading "/*" means match in all directories.*
For example, if we use "**/venv", Git will ignore a file or directory "venv" anywhere in our repository.
Our .gitignore file may look like one of these examples:

```
**/venv
*.pyc
__pycache__
```

```
.settings/
target/
bin/
```

Tutorial

- Create a new Git repository
- Clone the repository to your [USER_HOME]/projects/
- Configure the user.name and user.email properties globally
- Add the following files to the repository folder, it doesn't matter what these files contain
  - file.java
  - file.cs
  - file1.txt
  - File2.txt
- Stage the java file and commit it
- Stage all text files and commit them
- Push the changes to your remote repository
- Add a new file called file.py
- Stage all files and commit them
- Push the changes to your remote repository

Exercises

There are no exercises for this module.