CMPS 3390 Homework 3

Fall 2024

Version Control

1. **Git Resources**
   - **GIT**: The industry standard for tracking changes in code over time and managing project history. Git was developed by Linus Torvalds and is widely used for its flexibility, branching capabilities, and ability to handle large projects efficiently.
   - **Github**: GitHub is a cloud-based platform where you can store, share, and work together with others to write code. Storing your code in a "repository" on GitHub allows you to:
     - Showcase or share your work.
     - Track and manage changes to your code over time.
     - Let others review your code, and make suggestions to improve it.
     - Collaborate on a shared project, without worrying that your changes will impact the work of your collaborators before you're ready to integrate them.
   - **Github Gists:** Think of a gist like a tiny little single file repo. A gist is a simple way to share code snippets, text, or other information via GitHub. Gists also support versioning, so you can see how your gist changes over time.
   - **Bitbucket**: Bitbucket is another cloud-based GIT platform owned by Atlasian. While Bitbucket is free for personal use, Bitbucket is primarily marketed to larger software development companies and teams. This is due to the fact that Atlassian also owns Jira, Trello, and Confluence which allows them to integrate version control directly into these project management tools.
2. **Version Control Key Concepts**
   - **Initialize:** In order to begin tracking/committing the changes to your project, you must first initialize a new git repo.
   - **Stage:** Once you have initialized your repo, you can start adding/staging files that you would like to track. Any time changes are made to your files you must stage the changes before they will be committed.
   - **Commit:** Once you have staged your changes, you can commit them to the repo. This is what actually SAVES the changes to your repo and creates a historical change record.
   - **Branch:** Any time you plan on making significant changes to your project, or if you have multiple environments (like DEV, STAGING, PRODUCTION) branching allows you to create a copy of the project that you can commit to without affecting the main branch.
   - **Merge:** Once you have successfully committed to a branch, you may eventually want to apply those changes to the source branch with a merge. If no changes have been made to the source branch, you can fast-forward, otherwise you may need to resolve any merge conflicts.

3. **Working With Remote Repositories (source)**
    - **Clone:** If you would like to make a local copy of a remote repository (and you have proper permissions) you can use this command.
    - **Remote Add/Remove:** If you have an existing local repo that you would like to associate with a specific remote repo, you would use the remote command.
    - **Fetch:** If you would like to download/update the history of your repo from the remote source WITHOUT doing a pull/merge.
    - **Pull:** Does the same thing as fetch, but will ALSO attempt to merge the most recent commit from the remote repo. This is good to do BEFORE you start making changes.
    - **Push:** Uploads and merges your local commit history to the remote/source repository
4. **Remote-Specific Concepts:**
    - **Forking:** If you do not have permission to clone a repo to your local machine, you can fork a copy of it to your remote account instead. Then you can clone YOUR copy and make changes.
    - **Pull Requests:** When you would like the original project (that you forked from) to incorporate your changes, you can request that they PULL your changes.

Virtualization

1. **Hypervistors**
    - Hypervisors manage and run virtual machines on physical hardware. They allow multiple OS instances to run simultaneously on a host machine. There are two types:
        - **Bare-metal:** Directly installed on hardware, without a host OS (e.g., Microsoft Hyper-V).
        - **Hosted:** Runs on top of a host OS (e.g., Oracle VirtualBox, VMware Workstation).
2. **Runtime Environments**
    - These provide the environment necessary for programs to run, handling execution, memory management, and other system interactions. Examples include:
        - **Java Virtual Machine (JVM)**: Executes Java bytecode on any system with a compatible JVM.
        - **Node.js:** A runtime for executing JavaScript code outside of a browser.
3. **Containers**
    - Containers package applications with all their dependencies but share the host OS kernel, making them more lightweight than traditional VMs. They are highly portable and often used in cloud and microservices architectures.
      Popular container platforms include **Docker** and **Kubernetes**.
4. **System Emulators**
    - Emulators mimic hardware to allow software designed for one system to run on another. Examples include QEMU, which can emulate different processor architectures and platforms, and game console emulators.