Sri Lanka Institute of Information Technology

APPLICATION FRAMEWORKS

VERSION CONTROLLING

LECTURE 02

Faculty of Computing
Department of Software Engineering
Module Code: SE3040

VERSION CONTROLLING

- What and why?
- Terminology
- Best practices
- GIT

WHAT?

- Managing changes to a source.
- Changes are identified using a revision number.
- Each revision has its timestamp as well as the person who done the change.
- Revisions can be restored, compared and merged.
- “Management of multiple revisions of the same unit of information”

WHY?

- Easier backups and centralized source code repository.
- Easy collaborative development.
- Overview of changes performed to a file.
- Access control.
- Conflict resolvement.

TERMINOLOGY

- **Repository**
  - Central location where all the files are being kept. Usually a directory with set of files.
- **Trunk**
  - Also referred to as master branch. This is where the most stable code is being placed which is referred as the production code.
- **Stage**
  - Mark files for tracking changes.
- **Commit**
  - Create a snapshot of the changes being made to the files.

Stage

- In GitHub, a stage refers to the point in the development process where changes to a file or set of files have been prepared for committing to the repository.

- When you make changes to files in your local repository, you need to stage them before you can commit them to the repository. Staging allows you to review your changes and selectively choose which changes to include in the commit.

- You can stage changes in GitHub using the command line interface (CLI) or through the GitHub Desktop application. Once you have staged your changes, you can commit them to the repository with a commit message that describes the changes you have made.

- By staging changes before committing them, you can keep your commits focused on specific changes, which makes it easier to track and review changes over time.

TERMINOLOGY... (CNT)

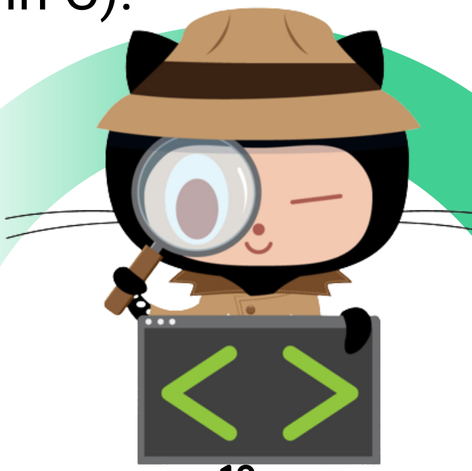
- **Branch**
  - Copy of the master branch taken at a given point. All the feature developments and bug fixes will be done in a branch. Usually it is allowed to have multiple branches at the same time.
- **Checkout**
  - Mark/unlock file for changing.
- **Merge**
  - Combining branches together to update the master branch.
- **Merge conflict**
  - Merge conflicts occur when merging a file which has been changed in two separate branches or places. Changes that interfere other changes.

BEST PRACTICES

- Use a source control system.
- Always make sure to have the latest version of the file.
- In distributed source control system advice is to get the latest source code at least start of the day.
- Checkout only what you need.
- Merge code with the development branch at least once per day.
- Always make sure code is working as expected and it is not causing any other code to break.
- Follow a formal review process when merging.

GIT

- Most popular version control system.
- Distributed version control system.
  - Client get a complete clone of the source code. In a disaster situation full source along with all history can be restored from a client.
- Free and open source.
- Multiple branches and tags.
  - Feature branches, role branches (production).
- Faster comparing to other systems (works on a linux kernel and written in C).
- Support multiple protocols
  - HTTP, SSH
- Staging area, local commits and stashing.
  - Staging area - Mark files to be committed.
  - Local commit - Commit code locally without pushing into the remote branch.
  - Stashing - Keep file changes in Stash and apply them in a later.



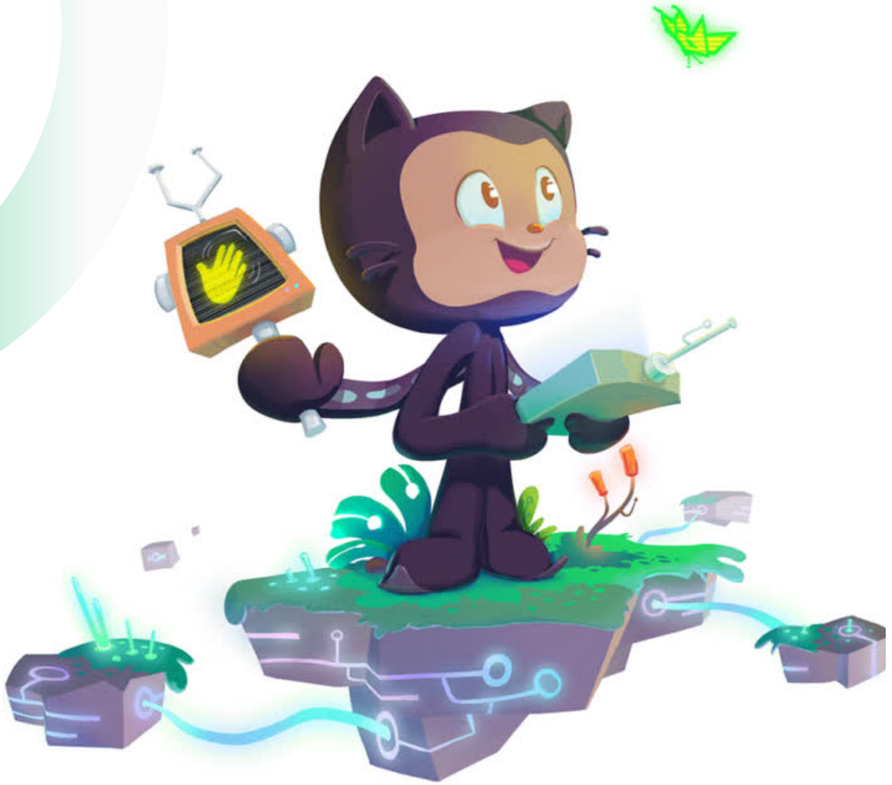
STAGING & STASHING

In Git, staging and stashing are two different concepts that can be used to manage changes to your code before committing them to the repository.

Staging: When you make changes to files in your local Git repository, you can stage them before committing them. Staging refers to the process of selecting which changes you want to include in your next commit. You can stage changes using the **git add** command. For example, if you have made changes to two files but only want to commit the changes to one of them, you can stage only the changes to that file using the **git add** command followed by the filename. Once you have staged your changes, you can commit them using the **git commit** command.

Stashing: Stashing is a Git feature that allows you to temporarily save changes that are not ready to be committed. Stashing is useful when you want to switch to a different branch, but you don't want to commit your changes to the current branch.
To stash changes, you can use the **git stash** command. This command saves your changes to a temporary location, allowing you to switch to a different branch or work on a different task. When you're ready to resume work on the original branch, you can use the **git stash apply** command to restore the changes you stashed.
Stashing is also useful when you need to pull changes from the remote repository, but you have uncommitted changes in your local repository. Instead of committing your changes, you can stash them, pull the changes from the remote repository, and then apply the stashed changes to your local repository.

GIT COMMANDS



- Git init
- Git clone
- Git add
- Git stage
- Git commit
- Git push

https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

GIT: INTERACTIVE LEARNING

- Following are two good interactive demos for learning git.
- The fundamentals are found in [1] and advanced branching demo is in [2].

[1] https://try.github.io

[2] http://pcottle.github.io/learnGitBranching/

THAT'S ALL FOLKS !

ANY QUESTIONS ?