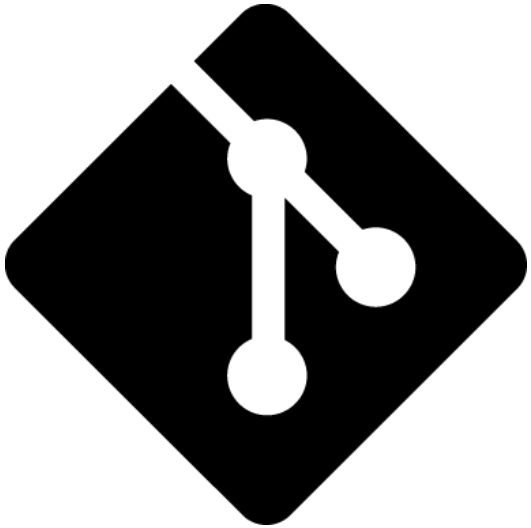


Hello Git

Version Control Using Git

Purdue Linux Users Group

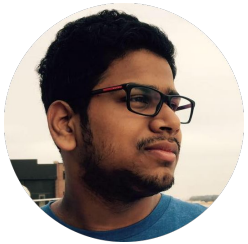
Hello World!

We are **PLUG.**

And we are here to teach you how to collaborate better!

About Us

- Promote information and awareness about open-source projects
- Technology and computer enthusiasts
- Our activities include:
  - Tech talks by professors
  - LAN Parties
  - Tech showcase by members



Amol Moses Jha
President

Dominic Yoder
Vice President

Anunai Ishan
Secretary

Parth Shelgaonkar
Treasurer

1

Version Control

What even is version control?
Why are you guys so obsessed with it?
Are you even okay? Do you need help?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

The Simple Version *(pun intended)*

How do you share files?

- Email
- Dropbox, Google Drive

How do you keep track of changes?

This is why we need *version control*.

- Keep track of changes
- Convey those changes better to other collaborators

Git: One of the *most popular* version control systems

2

Git - Getting Started

Cool! You got me onboard.
But how do I start using Git? How do I run it?
What do I do?

Getting Started with Git:
Initializing a Repository

- `git init`

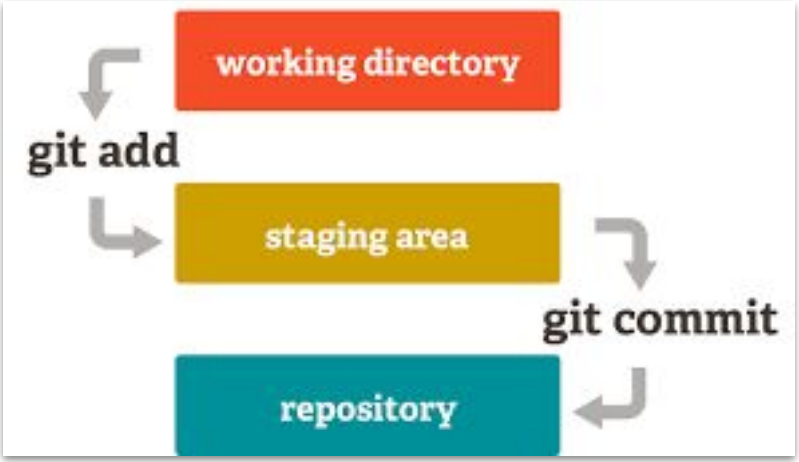
  Instruct git to initialize the current working directory to a repository

- `git clone`

  Instruct git to copy a repository into a directory

  (more about this later)

Getting Started with Git:
Saving Changes



Getting Started with Git:
Saving Changes

- `git add`

  Add all changes to the **staging area**, where Git keeps track of the changes in the files specified.

- `git commit`

  Save a snapshot of the **staging area**— essentially create a *version* of the project at a specific point in time.
  Commits are descriptive in order to help you to be able to navigate through them if needed later.

Getting Started with Git:
Inspecting Changes

- `git diff`

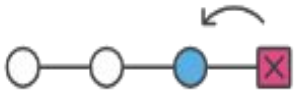
  Examine changes in the staging area from the previous version, or “commit”

- `git log`

  Show a list of all previous commits, the commit message and their “IDs”

Getting Started with Git:
Undoing Changes

- `git revert <commit-id>`



Undo all changes from the to the specified commit, creating an inverse “commit” in the process

- `git reset --hard <commit-id>`

Deletes history, making so that the changes after the commit never happened!

⚠ Never run `reset --hard` on publicly-shared repositories.

3

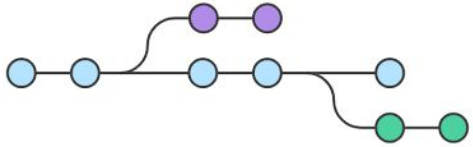
Git - Getting Good

Tell me more!
Knowledge is power!

Getting Good with Git: gitignore

- All files don't need to be tracked
  - Examples include:
    - Compiled code (`.class`, `.o`, `.pyc` files)
    - Project-specific dependencies (`node_modules`)
- Such files tend to litter our repository and Git history.
- We ignore such files by specifying them in a file called `.gitignore.`
- Git ignores the files mentioned in the `.gitignore` file while keeping track of changes made to project.

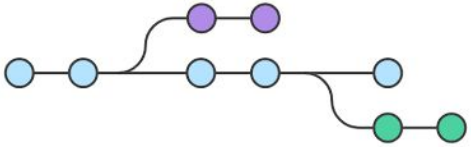
Getting Good with Git:
Branches



Creating branches allows us to:

- Work on different features
- Experiment with different implementations
- Fix bugs without affecting the current, stable state of the project

Getting Good with Git:
Branches



- `git checkout -b <new-branch-name>`

  Creates a new branch to work with

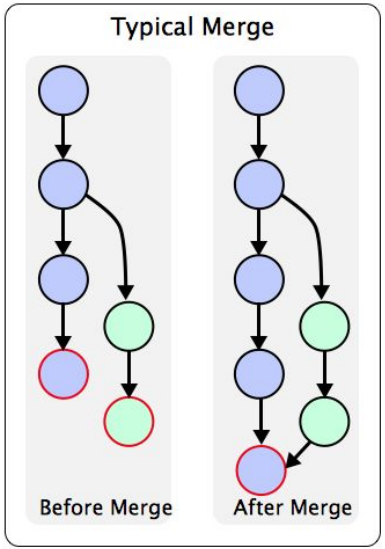
- `git checkout <branch-name-to-checkout>`

  Switches to the branch whose name is specified

Getting Good with Git: Merging

How does merging help?

- Brings changes relayed in other branches back to a single branch



Getting Good with Git:
Merging

- `git merge <branch-name>`

  This command is run in the branch in which you want to merge `<branch-name>` specified in the command.

Getting Good with Git:
Merging

Merge Conflicts

- Occur when two collaborators change the same file in different settings and then try to perform a merge
- The best way to avoid a merge conflict is to try to never have one.
- Always try to work on distinct components and claim ownership of your part in a project.

Don't be a git.

Getting Good with Git:
Remotes

- Basically a “remote” location where a copy of the repository is saved

- Allows multiple users to create a copy of the repository and work collectively

Getting Good with Git:
Remotes

- `git remote add <name-of-remote> <url-to-add>`

  Adds a remote destination for the repository

- `git clone <url-of-remote>`

  "Clones" the repository — makes a complete copy with the origin already set

Getting Good with Git:
Syncing with Remote

- `git fetch <name-of-remote>`

  Obtain changes made in remote into the local repository

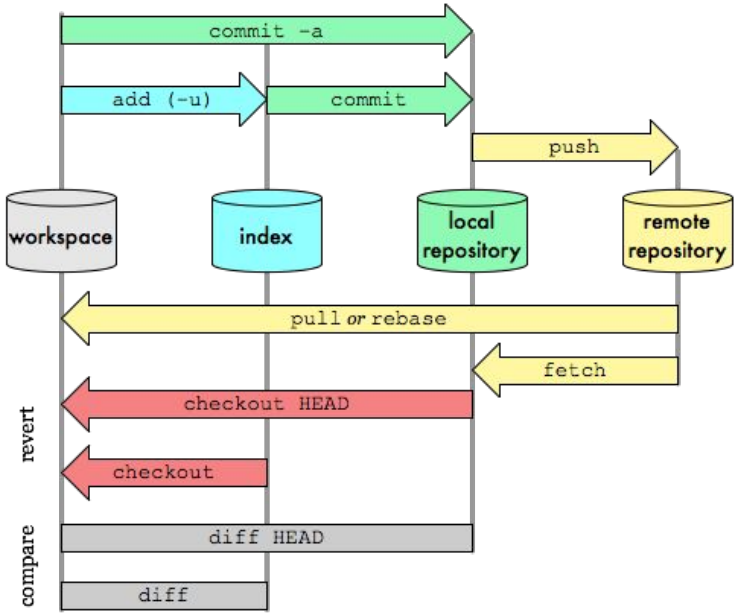
- `git pull <name-of-remote>`

  Obtain changes made in remote, and merge them in a single step

- `git push <name-of-remote>`

  Publish changes made in local repository to the remote repository

Putting It All Together:
Typical Workflow with Git



Git ≠ GitHub

- Common misconception
- **GitHub** is a **web-based hosting service for version control using Git**.
- Some other ones:
  - GitLab
  - Bitbucket
  - Beanstalk
  - RocketGit
  - Codebase

4

Git - GUI Clients

Terminal bad!

GUI Clients

- A comprehensive list can be found here: https://git-scm.com/downloads/guis
- We will discuss GitKraken today.

Fun Fact

RCS (Revision Control System), an early version control system was released at Purdue by Walter Trichy in 1982.

Wikipedia - RCS

Resources

- Git Book - https://git-scm.com/book/en/v2
- Codecademy - https://www.codecademy.com/learn/learn-git
- Presentation - https://github.com/ianunai/hello-git

Thank You!

Questions?

Connect with PLUG

facebook.com/purduelug

purduelug.org

We hope to see you at our callout.