

# Introduction to Git and GitHub

Lan Peng

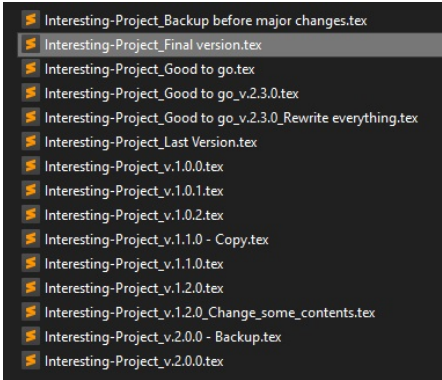
Department of Industrial and Systems Engineering, University at Buffalo, SUNY

September 28, 2021

Feel free to check out this slide on

<https://github.com/isaac0821/UB-INFORMS-Git-Workshop>

# Version Control



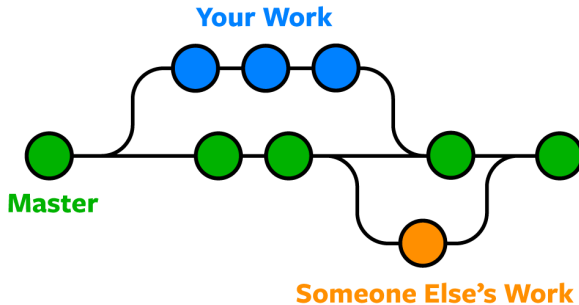
- Interesting-Project\_Backup before major changes.tex
- Interesting-Project\_Final version.tex
- Interesting-Project\_Good to go.tex
- Interesting-Project\_Good to go\_v.2.3.0.tex
- Interesting-Project\_Good to go\_v.2.3.0\_Rewrite everything.tex
- Interesting-Project\_Last Version.tex
- Interesting-Project\_v.1.0.0.tex
- Interesting-Project\_v.1.0.1.tex
- Interesting-Project\_v.1.0.2.tex
- Interesting-Project\_v.1.1.0 - Copy.tex
- Interesting-Project\_v.1.1.0.tex
- Interesting-Project\_v.1.2.0.tex
- Interesting-Project\_v.1.2.0\_Change\_some\_contents.tex
- Interesting-Project\_v.2.0.0 - Backup.tex
- Interesting-Project\_v.2.0.0.tex

# Version Control

The disadvantage of control version in this way

- Too messy, hard to find the latest (time stamps sometimes can't help)
- Cannot compare the difference between each version
- Cannot merge changes in different edition from one or more participants
- Cannot keep track of changes

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files.



# Installation

- Windows: <https://git-scm.com/downloads>
- Linux: `sudo apt-get install git`
- Mac OS:
  - ① Install Xcode from AppStore
  - ② Run Xcode
  - ③ In Xcode -> Preferences find Downloads
  - ④ Choose Command Line Tools, install it

# Installation

After installation, open the terminal and input git to see if it is successfully installed. There should be something like this:

```
C:\Users\lanad>git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
       [--exec-path<path>] [--html-path] [--man-path] [--info-path]
       [-p | --paginate] [-P | --no-pager] [--no-replace-objects] [--bare]
       [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
       <command> [<args>]

These are common Git commands used in various situations:


start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one


work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index
  sparse-checkout  Initialize and modify the sparse-checkout


examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  diff       Show changes between commits, commit and working tree, etc
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status


grow, mark and tweak your common history
  branch     List, create, or delete branches
  commit     Record changes to the repository
  merge      Join two or more development histories together
  rebase     Reapply commits on top of another base tip
  reset      Reset current HEAD to the specified state
  switch     Switch branches
  tag        Create, list, delete or verify a tag object signed with GPG


collaborate (see also: git help workflows)
  fetch      Download objects and refs from another repository
  pull       Fetch from and integrate with another repository or a local branch
  push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
```

# GitHub

- A/the global company that provides hosting for software development version control using Git.
- A website that you can goof around when you are bored at coding and don't need to worry about getting caught by your manager (she/he might be doing the same thing).



# Initialize/Clone

- `git init`

Create a `.git` folder in current path, making current path a git repository

- `git clone`

From a remote repository clone to current path

# Branch

- List branches  
`git branch`
- Create a new branch  
`git branch [branchName]`
- Switch between branches  
`git checkout [branchName]`
- Create and switch to branch (2 steps in 1 line)  
`git checkout -b [branchName]`

## Branch (cont.)

- Delete a branch (won't delete if there are unmerged changes)  
`git branch -d [branchName]`
- Delete a branch (will delete even if branch has unmerged changes)  
`git branch -D [branchName]`
- Rename a branch, default to be current branch  
`git branch -m [oldBranchName] [newBranchName]`
- Back to main branch  
`git checkout master`

# Commit

The following are steps for commit

- Step 0: Check changes using `git status`
- Step 1.1: Add changes by `git add <filename>` or add all changes by `git add -A`
- Step 1.2: Discard changes by `git checkout <filename>`
- Step 2: Commit the changes by `git commit -m "Some description"`

After commit, the next step is push it to remote.

If you don't want to commit to remote and want to undo it

- Step 1: Find commit hash, `git log`
- Step 2: Revert commit, `git revert [commitHash]`, it actually create a new commit to revert this commit

- `git fetch`

Check if there is a new version, you might not want to pull at this moment

- `git pull origin <branch>`

From remote pull changes into local

- `git push origin <branch>`

Sync remote repository with local repository

### Important

Always remember to pull before start editing, otherwise you might have conflicts

# .gitignore

.gitignore is a list of rules that will be ignored when you are committing changes, when creating repo from GitHub, GitHub provides easy ways to choose different .gitignore file templates.

# Merge

- Step 1. Go to the branch you need to merge into (e.g. master)
- Step 2. `git merge <branch>`

# Conflict

If your version is behind the remote version, i.e., someone had changed the files and push to remote repository, you also made some changes in the same file before you `git pull` from remote. Then when you try to `git push` your version to remote, a message will alert to let you `git pull` first. Git will try to solve the conflict by itself, but it might not work. Then you need to solve the conflict.



## Solve conflict

If the conflict happens in a non-binary file (e.g., .py, .tex, basic every file can be edited by text editor), it is easy to solve:  
Use text editor to open the conflict file, search for the following structure:

```
<<<<<< HEAD
    [Text in your version]
=====
    [Text in remote version]
>>>>>> [version number]
```

Edit it and `git add`, `git commit` again

For binary files (e.g., .pdf, .exe, .docx, .dll), it is hard to merge them. What we can do is choose a version.

- Resolve using your version

```
git add [conflictFileName]
git commit -m "I used mine version"
```

- Resolve using remote version

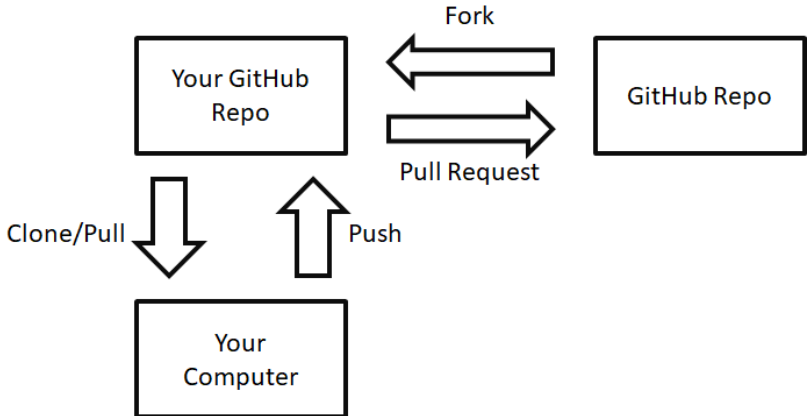
```
git checkout [conflictFileName]
git add [conflictFileName]
git commit -m "I used their version"
```

## Tip

We don't like to merge changes in the same file, here are some tips to avoid that:

- Design by module, break large files into small files. Each group member works on a separated file, have someone (e.g., team leader) design the interface between modules
- Sync with master branch more often
- Commit with small changes (it depends)

# GitHub Workflow



# Fork/Pull Request

- **Fork:** Make a copy of an existing repository under your account.
- **Pull Request:** On GitHub website, you can find a button named New pull request. It “requests” the owner/admin of the repository to do a “pull” from your branch/repository. After the owner pull your branch, your branch in remote will be deleted, but you will still have your local copy so the workflow will not be affected.

## Keep track of forked repo

- `git remote add upstream URL_OF_ORIGINAL_OWNER`  
Designate the original owner as the “upstream”
- `git fetch upstream` Fetch the latest update from original owner
- `git checkout master` Check out to master branch in your forked repo
- `git merge upstream/master` Merge the changes in the original owner into your repo
- `git push origin master` Update your master branch