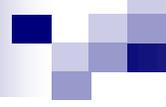


A Tutorial for Git and GitHub

Xiao Li

Department of Informatics

University of Zurich



Agenda

- Why use Version (Source) Control Systems
- What are Git and GitHub
- Basic Git Commands
- Fundamentals of GitHub
- Using GitHub in Project Implementation

Why version control?

- Scenario 1:
 - Your program is working
 - You change “just one thing”
 - Your program breaks
 - You change it back
 - Your program is still broken--*why?*
- Has this ever happened to you?

Why version control? (part 2)

- Your program worked well enough yesterday
- You made a lot of improvements last night...
 - ...but you haven't gotten them to work yet
- You need to turn in your program *now*

- Has this ever happened to you?

Version control for teams

■ Scenario:

- You change one part of a program--it works
- Your co-worker changes another part--it works
- You put them together--it doesn't work
- Some change in one part must have broken something in the other part
- What were all the changes?

Teams (part 2)

- Scenario:
 - You make a number of improvements to a class
 - Your co-worker makes a number of *different* improvements to the *same* class
- How can you merge these changes?

Version control systems

- A version control system (often called a source code control system) does these things:
 - Keeps multiple (older and newer) versions of everything (not just source code)
 - Requests comments regarding every change
 - Allows “check in” and “check out” of files so you know which files someone else is working on
 - Displays differences between versions

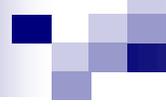
Benefits of version control

- For working by yourself:

- Gives you a “time machine” for going back to earlier versions
- Gives you great support for different versions (standalone, web app, etc.) of the same basic project

- For working with others:

- Greatly simplifies concurrent work, merging changes



What are Git and GitHub

- Git is a free and open source distributed **version control system** designed to handle everything from small to very large projects with speed and efficiency
- GitHub is a **web-based** Git repository **hosting service**, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

How to setup Git and GitHub

- Download and install the latest version of [GitHub Desktop](#). This will automatically install Git *and* keep it up-to-date for you.
- <https://help.github.com/articles/set-up-git/>



BASIC GIT COMMANDS

Introduce yourself to Git

- On your computer, open the **Git Shell** application.
- Enter these lines (with appropriate changes):
 - `git config --global user.name "John Smith"`
 - `git config --global user.email jsmith@seas.upenn.edu`
- You only need to do this once
- If you want to use a different name/email address for a particular project, you can change it for just that project
 - `cd` to the project directory
 - Use the above commands, but leave out the `--global`

The repository

- Your top-level **working directory** contains everything about your project
 - The working directory probably contains many subdirectories—source code, binaries, documentation, data files, etc.
 - One of these subdirectories, named `.git`, is your repository
- At any time, you can take a “snapshot” of everything (or selected things) in your project directory, and put it in your repository
 - This “snapshot” is called a **commit object**
 - The commit object contains (1) a set of files, (2) references to the “parents” of the commit object, and (3) a unique “SHA1” name
 - Commit objects do *not* require huge amounts of memory
- You can work as much as you like in your working directory, but the repository isn’t updated until you **commit** something

init and the `.git` repository

- When you said `git init` in your project directory, or when you cloned an existing project, you created a repository
 - The repository is a subdirectory named `.git` containing various files
 - The dot indicates a “hidden” directory
 - You do *not* work directly with the contents of that directory; various git commands do that for you

Making commits

- You do your work in your project directory, as usual
- If you create new files and/or folders, they are *not tracked* by Git unless you ask it to do so
 - `git add newFile1 newFolder1 newFolder2 newFile2`
- Committing makes a “snapshot” of everything being tracked into your repository
 - A message telling what you have done is required
 - `git commit -m “Uncrevulated the conundrum bar”`
 - `git commit`
 - This version opens an editor for you to enter the message
 - To finish, save and quit the editor
- Format of the commit message
 - One line containing the complete summary
 - If more than one line, the second line must be blank

Commits and graphs

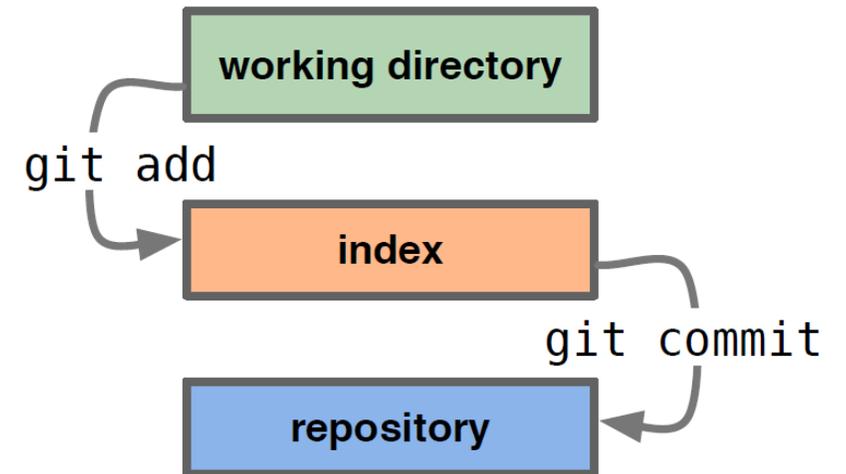
- A **commit** is when you tell git that a change (or addition) you have made is ready to be included in the project
- When you commit your change to git, it creates a **commit object**
 - A commit object represents the complete state of the project, including all the files in the project
 - The *very first* commit object has no “parents”
 - Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object
 - Hence, most commit objects have a single parent
 - You can also **merge** two commit objects to form a new one
 - The new commit object has two parents
- Hence, commit objects forms a **directed graph**
 - Git is all about using and manipulating this graph

Commit messages

- In git, “Commits are cheap.” Do them often.
- When you commit, you must provide a one-line message stating what you have done
 - Terrible message: “Fixed a bunch of things”
 - Better message: “Corrected the calculation of median scores”
- Commit messages can be very helpful, to yourself as well as to your team members
- You can’t say much in one line, so commit often

Typical workflow

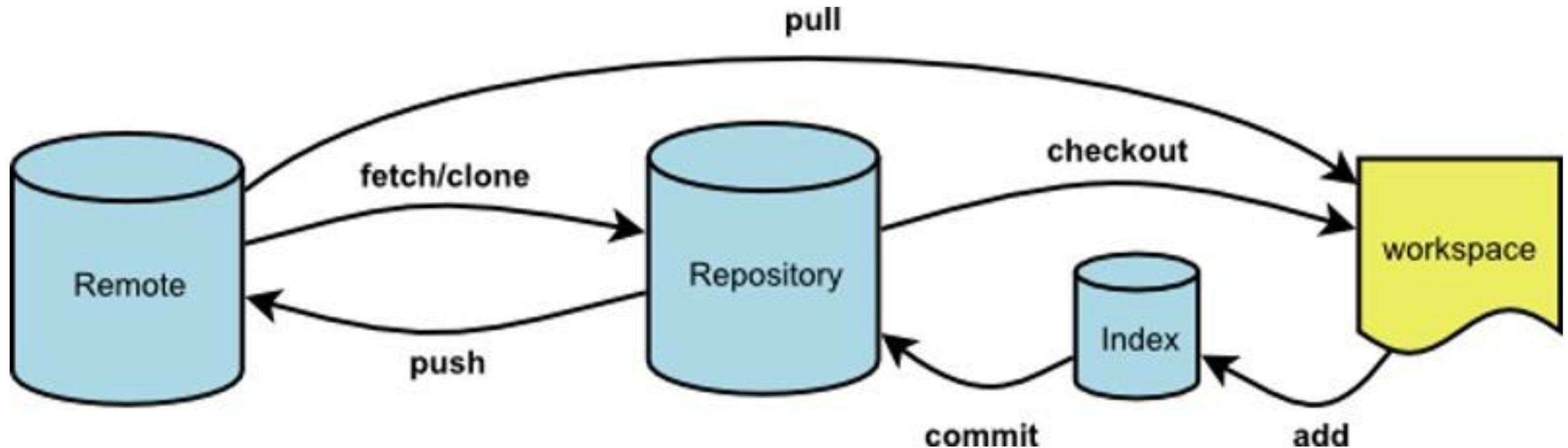
- `git status`
 - See what Git thinks is going on
 - Use this frequently!
- Work on your files
- `git add your editfiles`
- `git commit -m "What I did"`



Keeping it simple

- If you:
 - Make sure you are current with the central repository
 - Make some improvements to your code
 - Update the central repository before anyone else does
- Then you don't have to worry about resolving conflicts or working with multiple branches
 - All the complexity in git comes from dealing with these
- Therefore:
 - Make sure you are up-to-date before starting to work
 - Commit and update the central repository frequently
- If you need help: <https://help.github.com/>

More Commands: Don't Get Scared.



GitHub Desktop can Help You



FUNDAMENTALS OF GITHUB

Introduce yourself to GitHub

- Register on GitHub

- <https://github.com/>

- Authenticating to GitHub Desktop

- <https://help.github.com/desktop/guides/getting-started/authenticating-to-github/>

- Configuring Git for GitHub Desktop

- <https://help.github.com/desktop/guides/getting-started/configuring-git-for-github-desktop/>

Create or add a repository to GitHub

- Create a new repository on GitHub

- <https://help.github.com/articles/create-a-repo/>

- From GitHub Desktop, then Publish to GitHub

- <https://help.github.com/desktop/guides/contributing/adding-a-repository-from-your-local-computer-to-github-desktop/>

- Remember to Publish, otherwise your repository would not appear on the GitHub website.

Commit your changes on GitHub

- From GitHub Website

- <https://help.github.com/articles/create-a-repo/>

- From GitHub Desktop

- <https://help.github.com/desktop/guides/contributing/committing-and-reviewing-changes-to-your-project/>

Creating a branch for your work

- A branch is a parallel version of the main line of development in the repository, or the default branch (usually master). Use branches to
 - Develop features
 - Fix bugs
 - Safely experiment with new ideas
- From the GitHub Website
 - <https://help.github.com/articles/creating-and-deleting-branches-within-your-repository/>
- From the GitHub Desktop
 - <https://help.github.com/desktop/guides/contributing/creating-a-branch-for-your-work/>

Synchronizing your branch

- As commits are pushed to your project on GitHub, you can keep your local copy of the project in sync with the remote repository.
 - <https://help.github.com/desktop/guides/contributing/syncing-your-branch/>

Viewing the history of your commits

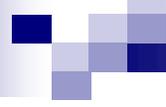
- When you click a commit on the commit timeline, you can see more details about the commit, including a diff of the changes the commit introduced.
- Each commit shows:
 - The commit message
 - The time the commit was created
 - The committer's username and profile photo (if available)
 - The commit's SHA-1 hash (the unique ID)

Revert your commit

- If you change your mind about a commit after you create it, you can revert the commit.
- When you revert to a previous commit, the revert is also a commit. In addition, the original commit remains in the repository's history.
- <https://help.github.com/desktop/guides/contributing/reverting-a-commit/>

Fork & Pull: A Collaborative model

- A fork is a copy of a repository that you manage. Forks let you make changes to a project without affecting the original repository. You can fetch updates from or submit changes to the original repository with pull requests.
- A great example of using forks to propose changes is for bug fixes. Rather than logging an issue for a bug you've found, you can:
 - Fork the repository.
 - Make the fix.
 - Submit a *pull request* to the project owner.



Using GitHub in Project Implementation

In the section of project implementation in your project report, you may describe:

- How you use GitHub in your project
- How version control helps your quality management
- How you collaborate with your teammate in GitHub

References

Some content of the slides are adapted from:

- <https://help.github.com/desktop/guides/getting-started/>
- <https://help.github.com/desktop/guides/contributing/>
- <https://help.github.com/categories/collaborating/>
- <http://www.cis.upenn.edu/~matuszek/cit591-2012/Lectures/git.ppt>