COMPSCI 1XC3 - Computer Science Practice and Experience: Development Basics

Topic 2 - Version Control

Zheng Zheng

McMaster University

Winter 2023

In real life, code changes a lot over the lifespan of a project

Imagine a project with hundreds of developers working on it.

Coding as a Team

One approach might be to have different developers work on separate areas of the code.

- AKA: Divide and Conquer!

In practice, this is pretty common.

- One developer works on one component (e.g., a library file in C), the next developer works on something else, etc.
- This is known as code “ownership”
- Pros:
  - Developers don’t make changes to overlapping areas of code.
  - Developers build expertise with an area of the code, and can make changes and updates faster.

Coding as a Team (cont.)

The problem of course is that:

1. Coding projects don't always break down that easily
2. We're trusting people to stay in their lane
3. At a certain point the various components need to be integrated.

Thus, the problem of multiple developers working on the same code is unavoidable for large projects. In addition, there is the **bus factor** to consider.

- "The bus factor is a measurement of the risk resulting from information and capabilities not being shared among team members, derived from the phrase 'in case they get hit by a bus'." - Wikipedia

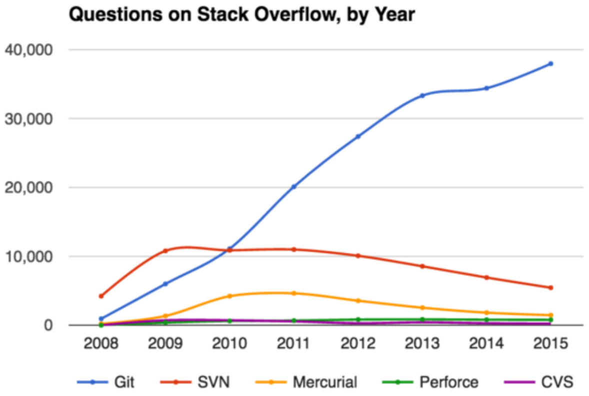
Modern problems require modern solutions

Enter Version Control

Version Control Systems manage changes to source code and related file in **repositories** or colloquially **repos**.

- Repositories make it very straightforward to keep your files up-to-date with the latest changes.
- Whenever changes are committed to the repo, a new version of the files is created.
- Developers can access older or alternative versions of the files, so detrimental changes can be reversed!
- Repos are important in the open source community, as they can be used to manage crowd-sourced coding projects.

Version Control System Usage



Git

Git was created in 2005 by Linus Torvalds.

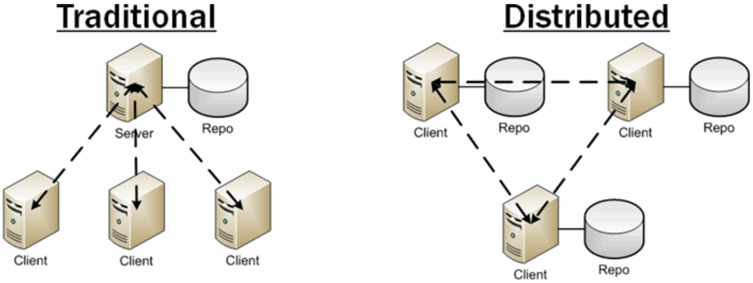
- That's right folks! The Linux Guy
- He named it "git" because it's British slang for an "unpleasant person"
- Apparently, people found him to be a git during coding projects.

Today, the most popular way to obtain a git repository is through the internet service **github**.

- Previously, you had to pay a subscription for private repositories, but since being bought out by Microsoft, private repos are free (with some probable limitations)!

Git (cont.)

Previous version control systems followed the traditional **client-server model**. Git uses a **distributed model**.



Though in practical terms, a server is still be necessary to coordinate the distributed clients.

GITting Started!

Even if github is hosting your repository, you will need the software installed if you don't want to do everything in a web browser.

- `https://git-scm.com/downloads`
- In general, GUIs are fine to use, but command line is usually faster, and can be integrated into scripts etc.
- A professional software developer is expected to understand how to use the command line version.

To create a new git repo...

```
$ git init
Initialized empty Git repository in /[...]/code/.git/
```

Files and folders with the `.` prefix are *hidden files*. You can view hidden files with `ls -a`

Git Workflow

Here are some general procedures for working in a repository (these are applicable to other version control systems!)

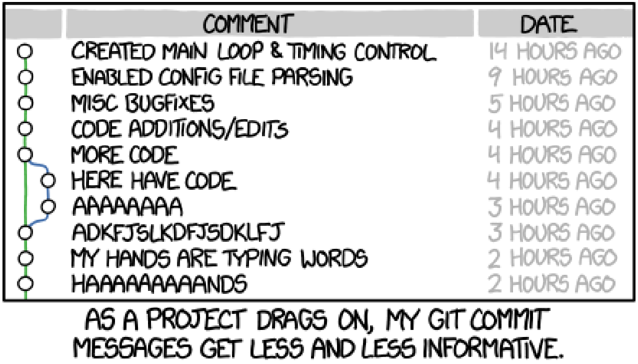
- In a repo, files and directories are either *tracked* or *untracked*.
  - Only tracked files and folders are a part of the repository.
  - Files and folders must be **add**ed to the repository manually.
- As you work on files, periodically **commit** your changes to the repository.
  - Think of this as *SUPER* saving your work.
- If you're working on a networked git repo (which is probable), you also have to **push** your commits to the network.
  - Doing this every time you commit is a good idea.

Basic Git Commands

We're not talking about server uploads/downloads just yet, just commands operational on local repositories.

- `git add <filename>`
  - Tells git to track the specified files and/or folders
  - You can add multiple files as well:
    - `git add f1 f2 f3`
    - `git add *.c`
- `git commit -m "log message"`
  - Commits everything in the working directory and all subdirectories.
  - Commits expect a log message, which can be specified using the `-m` flag.
  - If you fail to provide one, git will open up a text editor (like nano) because you probably just forgot, right?
  - Descriptive log messages are important! Important like good commenting habits!

Don't be this guy



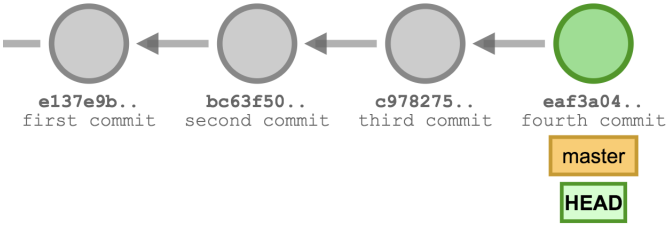
	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Basic Git Commands (cont.)

- `git status` displays...
  - Which files are and are not being tracked
  - Which files have changed since the last commit
  - The current working branch
- `git log`
  - Displays the date, time and author of commits on the current branch.
  - Also displays commit ID numbers and lovely log messages!
- `git checkout <ID# or branch name>`
  - Changes your tracked files to the specified commit or branch.
  - You can use this to undo changes and navigate the repository.

Head Master



- The **master** is the default branch created with the repository.
- The **head** of a branch is just the most recent commit.
  - Checking out a branch automatically takes you to the head of that branch.
- `Git Cheat Sheet:`
  https://education.github.com/git-cheat-sheet-education.pdf

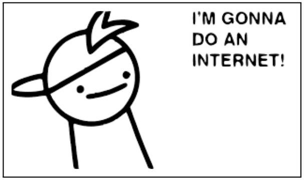
So What's the Point of This Again?

All the foregoing repository management stuff is great and all, but likely overkill for many tasks.

- It's more applicable than you think, any project of significant size would benefit from this approach.

Where things come alive is the ability to *network* your repositories!

- `git clone <repo>`
  - Creates a local instance of a remote repository
- `git push`
  - Transmits commits to the other networked repos
- `git pull`
  - Receives commits from the other networked repos and checks out the current head.

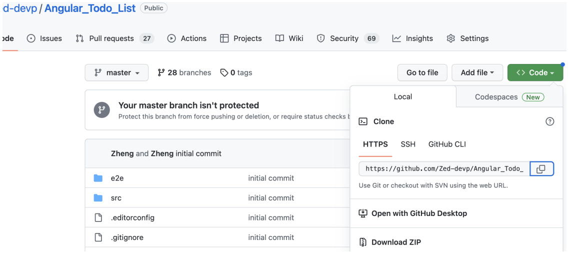


A note on URLs...

To clone a repo, you first have to know where it is, and what protocol you're using.

- `ssh://[user@]host.xz[:port]/path/to/repo.git/`
- `git://host.xz[:port]/path/to/repo.git/`
- `http[s]://host.xz[:port]/path/to/repo.git/`

If your repo is being hosted by GitHub, they have a handy button you can use to get the correct URL:


d-devp / Angular_Todo_List Public

ode Issues Pull requests 27 Actions Projects Wiki Security 69 Insights Settings

master 28 branches 0 tags Go to file Add file Code

Your master branch isn't protected
Protect this branch from force pushing or deletion, or require status checks b

Zheng and Zheng initial commit

e2e	initial commit
src	initial commit
.editorconfig	initial commit
.gitignore	initial commit

Local Codespaces New

Clone

HTTPS SSH GitHub CLI

`https://github.com/Zed-devp/Angular_Todo_`

Use Git or checkout with SVN using the web URL.

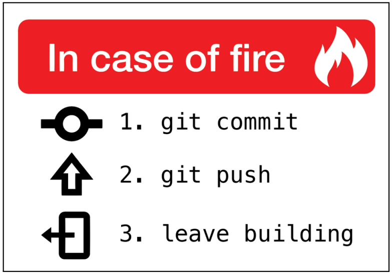
Open with GitHub Desktop

Download ZIP

Collaborative Workflow

When working on a project with humans, it is curteous to:

- Always `pull` the latest changes *before* you start working.
- Always `push` your changes *promptly* when you're done.
- Document! Your! Changes! Period! Exclamation Mark!


In case of fire

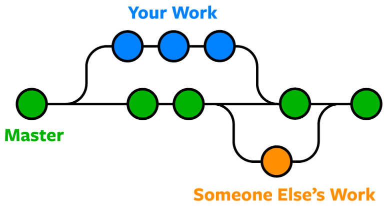
1. `git commit`
2. `git push`
3. `leave building`

- Any major changes (such as new features) should be made in a separate **branch**, and then **merged** back into `master` when the new feature is (mostly) complete.

Let me go out on a limb here...

It is your job to keep `master` stable at all times.

- Adding a feature, or even general development will often break things for a while, and pushing broken code into `master` is the height of bad manners!
- Once your branch is finished, you can merge it back into `master`, often with minimal effort.
- This way, different people's changes remain isolated throughout development.



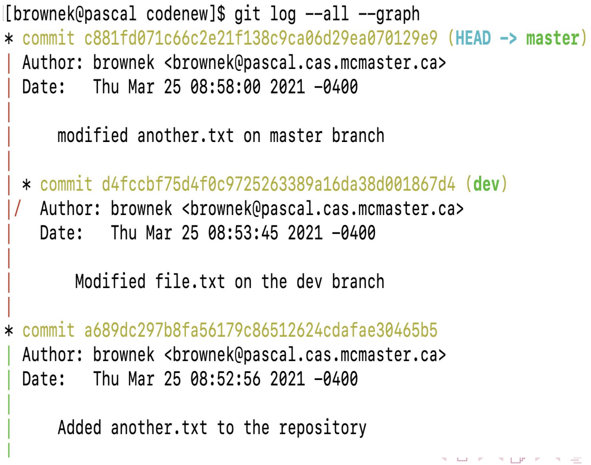
Assistant Branch Manager

Branching is easy!

- `git branch <branch name>`
  - Creates a new branch with the given name.
  - You have to switch to the new branch manually if you want to use it.
- `git switch <branch name>`
  - Switches to the specified branch.
  - Similar effect to changing your working directory in the file system.
- `git log --all`
  - Displays commits from all branches, not just the active branch.

Assistant TO the Branch Manager

`git log --all --graph` even gives you a visualization:



```
[brownek@pascal codenew]$ git log --all --graph
* commit c881fd071c66c2e21f138c9ca06d29ea070129e9 (HEAD -> master)
| Author: brownek <brownek@pascal.cas.mcmaster.ca>
| Date:   Thu Mar 25 08:58:00 2021 -0400
|
|     modified another.txt on master branch
|
| * commit d4fccbf75d4f0c9725263389a16da38d001867d4 (dev)
|/  Author: brownek <brownek@pascal.cas.mcmaster.ca>
|   Date:   Thu Mar 25 08:53:45 2021 -0400
|
|       Modified file.txt on the dev branch
|
* commit a689dc297b8fa56179c86512624cdafae30465b5
| Author: brownek <brownek@pascal.cas.mcmaster.ca>
| Date:   Thu Mar 25 08:52:56 2021 -0400
|
|     Added another.txt to the repository
|
```

Why the Fork Not?

You may notice on GitHub, one of your popularity indicators is the number of **forks** your repository has.

- A **fork** is a branch operation over an entire repository.
- Forking is often used to start a new project from an existing one.
    - LibreOffice was forked from OpenOffice in 2010, because some OpenOffice community members didn't like how Oracle did its licensing for previous open source projects.
- There's no command for forking with git, but many git GUI tools have a button for it (including GitHub!)

Merge Dragons!

`git merge <branch name>`

- Merges the specified branch into the currently active branch.
- You can see which branch you're in by using `git branch`.
- Merging requires a commit message, and git will prompt you with one for a text editor if it's missing.

git is pretty clever at merging branches, but there is always a possibility of **conflicts**.

- If the changes are in different files in the different branches, merger is trivial!
- If the changes are in different parts of the same file, merger is trickier, but will probably work...
- If the changes are in the same part of the same file, git probably won't be able to figure it out!

Conflicts in the Workplace

A **conflict** happens when a branch merger can't be resolved automatically.

- **Fixing conflicts is your job as the programmer!**
- It's tedious and no one likes it.

General workflow:

- When a conflict occurs, git puts something like this in the file with the conflict:

```
<<<<<<< HEAD
This text conflicts with the other text.
=======
I am the very model of a modern major general.
>>>>>>> branch_being_merged
```

Conflicts of Interest

- When this occurs, the easiest way to fix this is to open up the file in your favourite text editor, and manually make the changes.
- This requires you to read the conflicting lines and figure out what the solution should be!
  - If the computer could figure it out, it would have!
- Once you have resolved the conflicts (or as you are resolving the conflicts), commit and push your changes so that they're fixed for everyone, and not just you!

Fixing conflicts is a *major* pain, and many team workflows have conflict avoidance as a top priority. That said, they are inevitable, so you're going to need to figure out how to deal with them! Rolling back to a previous version, even if it's just a fact-finding mission, can be of critical importance!

Acknowledge

The contents of these slides were liberally borrowed (with permission) from slides from the Summer 2021 offering of 1XC3 (by Dr. Nicholas Moore).