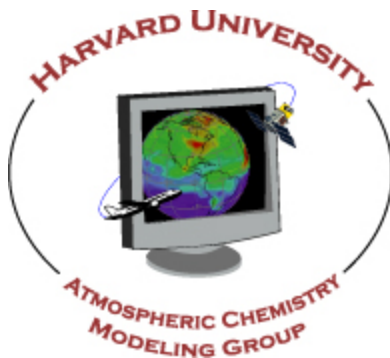


Improving Your Life With Git

Lizzie Lundgren

elundgren@seas.harvard.edu

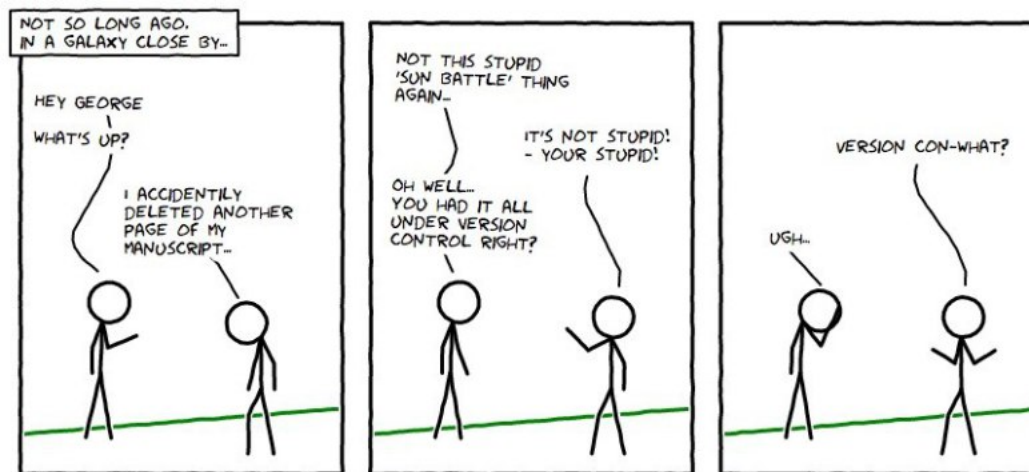


Graduate Student Forum

26 April 2018

Scenarios to Avoid

- My code was deleted from 90-day retention!
- Crap, I can't remember what I changed. I *think* it was...
- It worked before, I swear!
- Model version? No idea. I got a copy from a classmate, who got a copy from a friend.

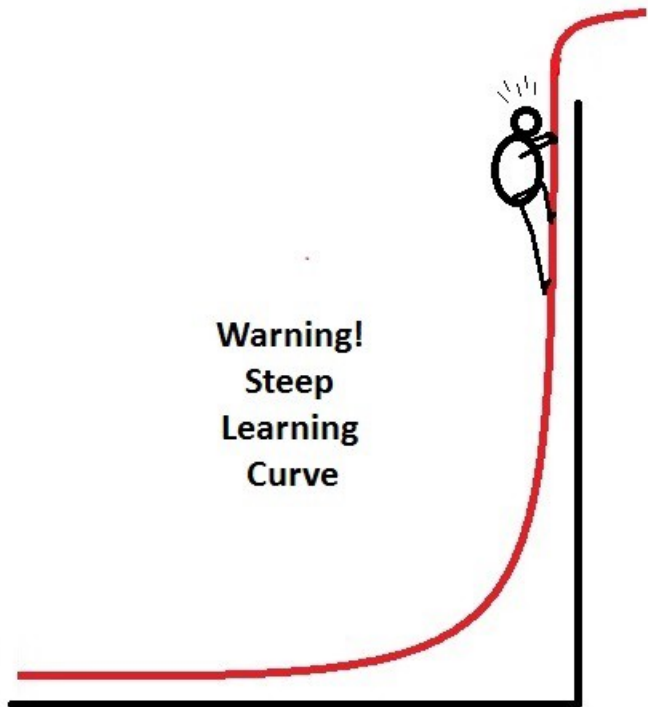


How Version Control Can Help

- Travel back in time to prior versions of a file
- But it's even more than that!
 - Track changes to remind yourself what, when, why
 - Short-term undo for recent mess-ups
 - Far back undo for catastrophic fails
 - Sandboxing areas big (branch) or small (uncommitted)
 - Branch merging
- It is especially useful for collaboration
 - Version synchronization for easy file sharing
 - Track ownership to give blame credit

Work Slow to Work Fast

- Understanding version control requires a conceptual leap
- There is a learning curve. You will mess up. It's worth it!

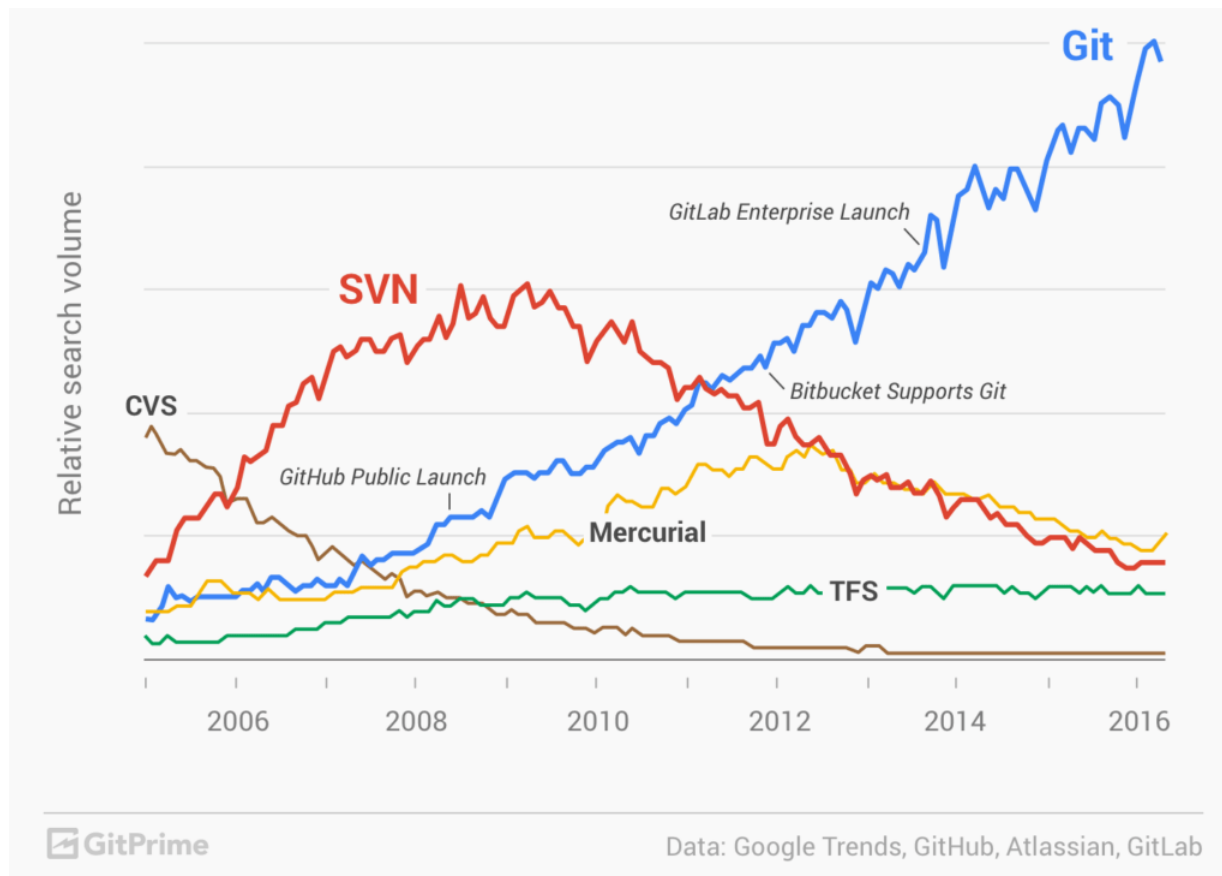


But please
try not to
do this →



Why Git?

- There are other options, but git usage has been steadily growing



Git Version Control Software

- Git version control software is a free *distributed* source code management system (as opposed to *centralized*)
- Offers many benefits over other version control software
 - Keep an *identical* copy of a shared repository on your own system
 - Every client contains the *entire* history of the project (this is huge!)
 - Easy-to-use graphical tools ship with it (git gui, gitk)
- Install git if you don't already have it
 - Check if your computer has git with 'git --version' or 'which git'
 - Use 'module load git' on Odyssey (check if it's already in your .bashrc)
 - If you don't have it, get it here: <https://git-scm.com/downloads>
- What about on Windows?
 - Check out tools for making git easier to use at <https://gitforwindows.org/>

Graphical Tools Distributed with Git

`gitk`

- Graphical repository browser for viewing and searching git history
- Use this to search history, view differences in past commits, create branches off of old versions, and revert to earlier version
- Use the `--all` option to view commits for all branches
- See <https://git-scm.com/docs/gitk>

`git gui`

- Graphical user interface for commit generation
- Use this to view uncommitted changes, write commit messages, and stage and commit files
- I frequently use the “Amend Last Commit” option to polish my commits
- See <https://git-scm.com/docs/git-gui>

These tools are optional but will make your life easier

Optional Config Files

.gitconfig

- Text file with your global settings, e.g. your name and email
- Store it in your home directory
- Feel free to copy and edit mine: `~elundgren/.gitconfig`

.gitignore

- Text file listing files git should ignore
- Store it in your repository
- See <https://git-scm.com/docs/gitignore>

.gitk

- Text file with your visual settings for the gitk repo browser
- Store it with your `.gitconfig`
- Feel free to copy and edit mine: `~elundgren/.gitk`

Remote Hosting

- To fully use the power of git you will want to make your project accessible to you and/or others from anywhere
- Privately-owned web-based hosting services allow you to do this
- There are many options, each with pros and cons:
 - GitHub
 - Bitbucket
 - SourceForge
 - GitLab
 - And more!
- I won't go into these here, other than this:
 - GitHub is very popular, and arguably the reason git is taking over the world
 - But private repos on GitHub cost \$
 - Bitbucket is less popular but a good option for free private repos
 - Harvard offers free public and private github repos with limitations
 - For more info, see <https://www.seas.harvard.edu/office-of-computing/announcements/codeharvard-harvard-enterprise-github>

Git Lingo and Workflow

Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
```

```
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

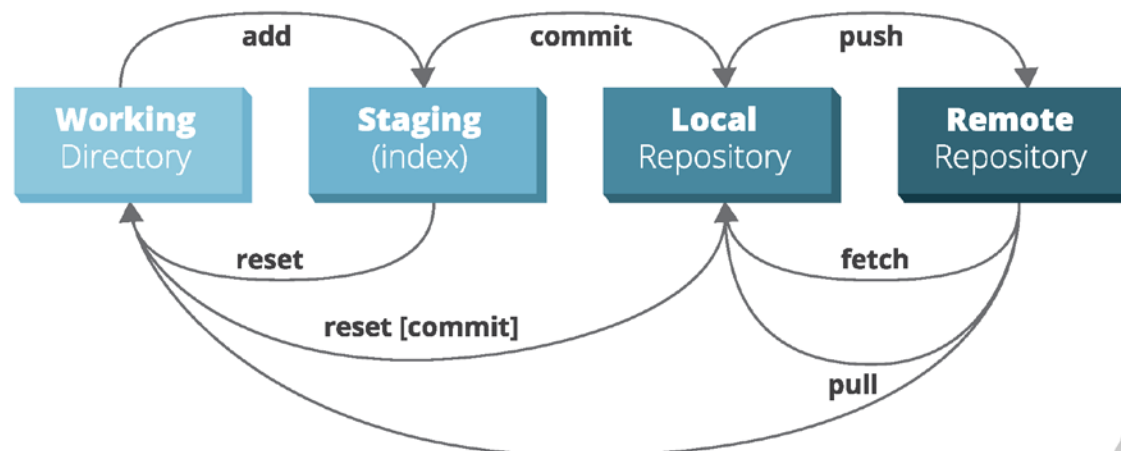
```
$ git push
```

Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit <https://training.github.com/> for official GitHub training.



Initialize a Local Repository

- Do this from the command line

```
$ cd my_working_directory  
$ git init
```

- Yes, it is that easy!
- However, your repo is empty until you stage and commit files
- What files should I add?
 - Typically people track text files, but other file types are possible
 - If you plan on using GitHub to host your project:
 - Hard file size limit: 100 MB
 - Soft repo size limit: 1 GB (will prompt a warning email)
 - No hard repo size limit but bad things might happen >1 GB
 - If you plan on using Bitbucket:
 - No file size limit
 - Soft repo size limit: 1 GB (will prompt a warning email)
 - Hard repo size limit: 2GB

Staged versus Unstaged Changes

- The *staging area*, aka *index*, is where all changes to commit are stored
- Basic steps to add your update to git history
 1. Make file changes
 2. Stage file changes
 3. Commit changes stored in the staging area
- Staging as an intermediary step allows the following:
 - Committing a subset of files
 - Committing a subset of changes in a single file
- Both of these features are very useful for keeping your git history clear and well-organized

To Stage and Commit All Files...

- Command line

```
$ git add *
```

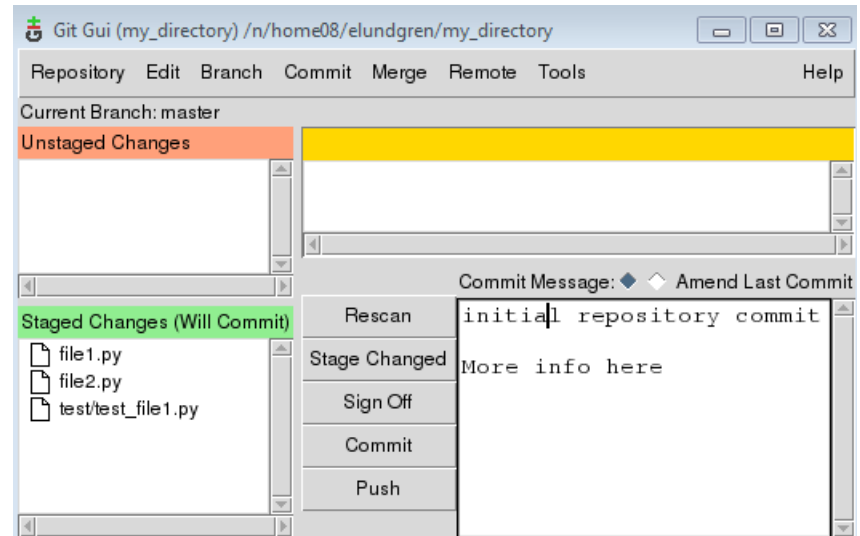
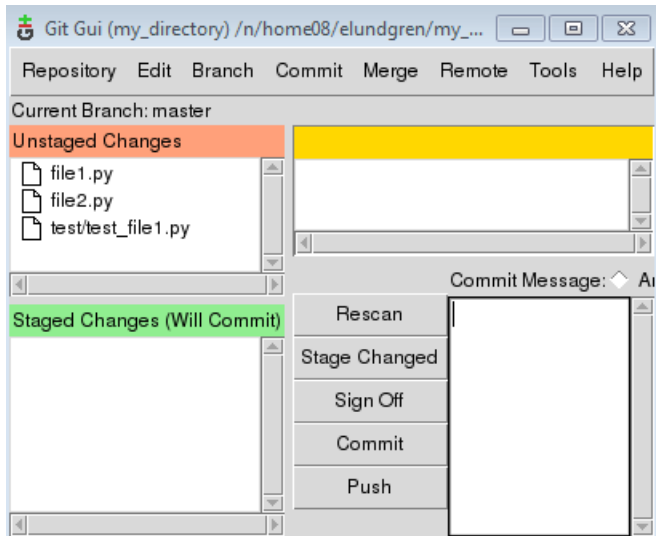
```
$ git commit -m "initial repository commit"
```

- You can also do this in one line...

```
$ git commit -am "initial repository commit"
```

- ...or using git gui

Note: Click on file in upper left box, then right-click line in upper right box to stage a subset of file changes.



Click each unstaged file to stage, add message, then click commit.
You can prevent files from showing up by adding them to .gitignore

Inspect Your Commit History

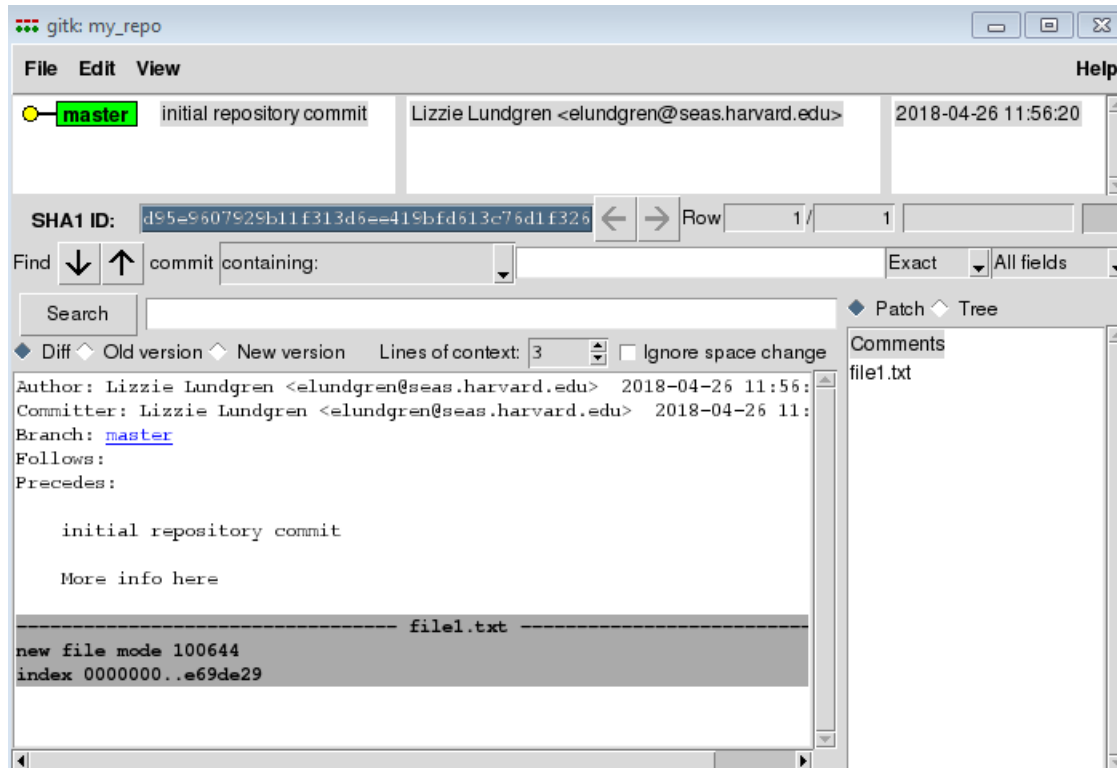
- Using the command line...

```
$ git log
```

```
$ git show
```

```
$ git show {commit hash}
```

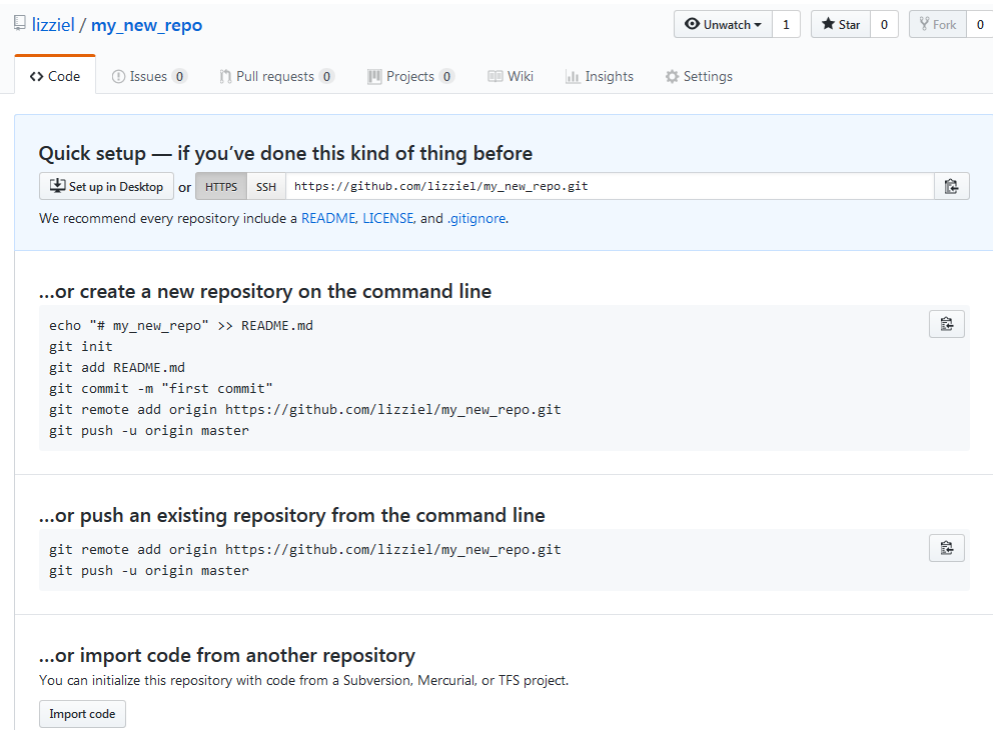
- ...or using gitk



Import to a Remote Repository

For this example I will use GitHub

- Open a free account if you don't already have one: <https://github.com/>
- Click 'Start a project'
- Enter repository name and click 'Create repository'
- Follow instructions to import code



The screenshot shows the GitHub interface for a repository named 'lizziel / my_new_repo'. At the top, there are navigation tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. Below the navigation, there are statistics for 'Unwatch', 'Star', and 'Fork'. The main content area displays 'Quick setup' instructions for creating a new repository on the command line. The instructions include a list of commands to create a new repository, add a README file, and push it to the remote repository. The repository URL is shown as 'https://github.com/lizziel/my_new_repo.git'. There is also an option to 'Import code' from another repository.

```
Quick setup — if you've done this kind of thing before
```

Set up in Desktop or **HTTPS** SSH `https://github.com/lizziel/my_new_repo.git`

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

```
...or create a new repository on the command line
```

```
echo "# my_new_repo" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/lizziel/my_new_repo.git
git push -u origin master
```

```
...or push an existing repository from the command line
```

```
git remote add origin https://github.com/lizziel/my_new_repo.git
git push -u origin master
```

```
...or import code from another repository
```

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

← I usually do this

Inspect Your Remote

- I just use the command line for this

```
$ git remote show origin
```

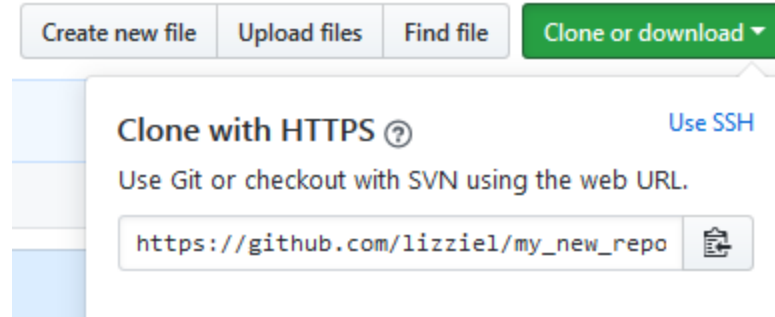
```
00 ~/my_repo $ git remote show origin
* remote origin
Fetch URL: https://github.com/lizziel/my_new_repo.git
Push URL: https://github.com/lizziel/my_new_repo.git
HEAD branch: master
Remote branch:
  master tracked
Local branch configured for 'git pull':
  master merges with remote master
Local ref configured for 'git push':
  master pushes to master (up to date)
```


Clone Remote Repository

- Say I delete my local repository. How do I get it back?

```
$ git clone https://github.com/lizziel/my\_new\_repo.git my_repo_clone
```

- What if I don't remember the link? Go to the GitHub repo:



- You can also clone anything local on your network

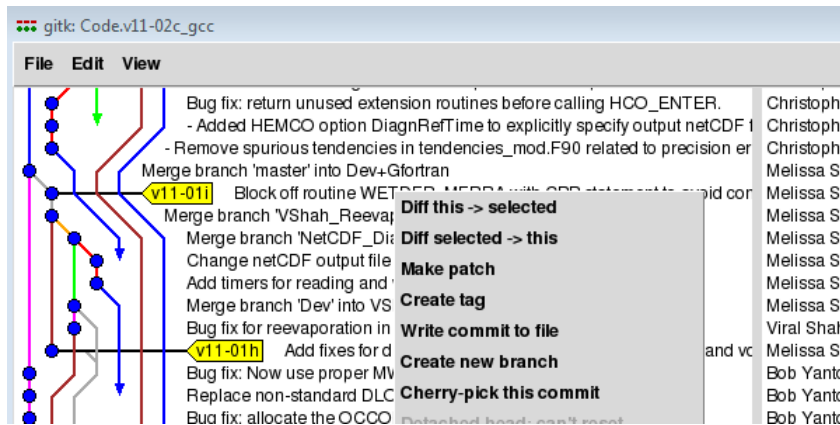
```
$ git clone ~elundgren/my_repo my_repo_clone
```

- Clone don't copy! Cloning configures the remote to be the original.

Advanced Topic Alert: If you plan on contributing to an open-source project, consider forking the project. This clones the repo to your GitHub account which you can then clone. Ultimately this enables pull requests.

Branching

- Check what branch you are on (will be asterisked)
`$ git branch`
- Create new branch off of current branch (will include unstaged)
`$ git checkout -b branch_name`
- Change to an existing branch (might require you to commit)
`$ git checkout branch_name`
- gitk is useful for branching off of old versions



GEOS-Chem example:

This popped up when I right-clicked v11-01i. Choose 'Create new branch' and then enter new branch name. It will then automatically be checked out (you will be in it).

Each one of the vertical lines in gitk is a branch. Horrifying, I know.

Push New Branch to Remote

- Make sure you are on the branch

```
$ git branch
```

- If you aren't, check it out

```
$ git checkout new_branch
```

- If you are, make sure everything is committed

```
$ git status
```

- The rest is simple, but I still look it up on the internet every time

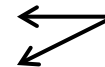
```
$ git push -u origin new_branch
```

- Check that it worked

```
$ git remote show origin
```

Woot! →

```
DD ~/my_repo $ git remote show origin
* remote origin
Fetch URL: https://github.com/lizziel/my_new_repo.git
Push URL: https://github.com/lizziel/my_new_repo.git
HEAD branch: master
Remote branches:
  master      tracked
  new_branch  tracked
Local branches configured for 'git pull':
  master      merges with remote master
  new_branch  merges with remote new_branch
Local refs configured for 'git push':
  master      pushes to master      (up to date)
  new_branch  pushes to new_branch (up to date)
```



Notice there are remote branches and local branches, and they have the same names

Get Changes from Remote

- This is relevant if you use the remote repo on different computers, or if you collaborate on a project with others
- Danger zone! Always check your branch before pulling.
- Basics:
 - Update your local remote branches to match the actual remote

```
$ git fetch
```
 - Same as above, but also merge into your checked out local branch

```
$ git pull
```
 - This method is safest

```
$ git pull origin branch_name
```
- Pull gets updates from the remote (origin) by default. But you can also pull from local repositories. Always specify branch name.

```
$ git pull ~elundgren/my_other_repo branch_name
```

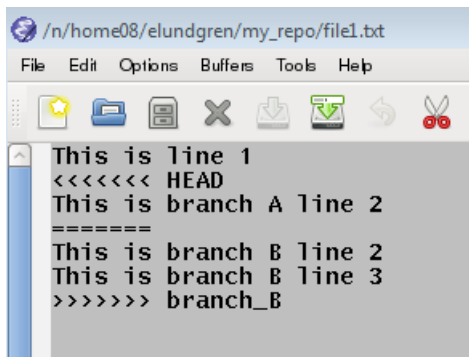
Merge Branches in a Repository

- Merge branch B into branch A within your local repo:

```
$ git checkout branch_A
```

```
$ git merge branch_B
```

- Obsessively checking log and status first is good practice
- You may run into merge conflicts...



```
/n/home08/elundgren/my_repo/file1.txt
File Edit Options Buffers Tools Help

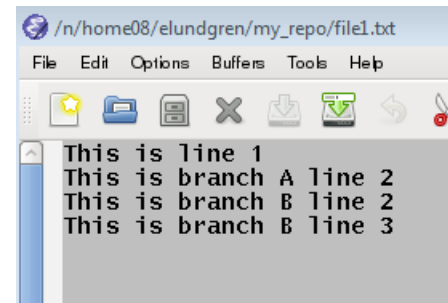
This is line 1
<<<<<< HEAD
This is branch A line 2
====
This is branch B line 2
This is branch B line 3
>>>>>> branch_B
```

← Current branch only (A) separated by <<<<<<HEAD and =====

← Merged branch only (B) separated by ===== and >>>>>> branch_name

Edit the file to make it how you want, delete the separator text, and commit the merge.

“Resolved” file to commit →



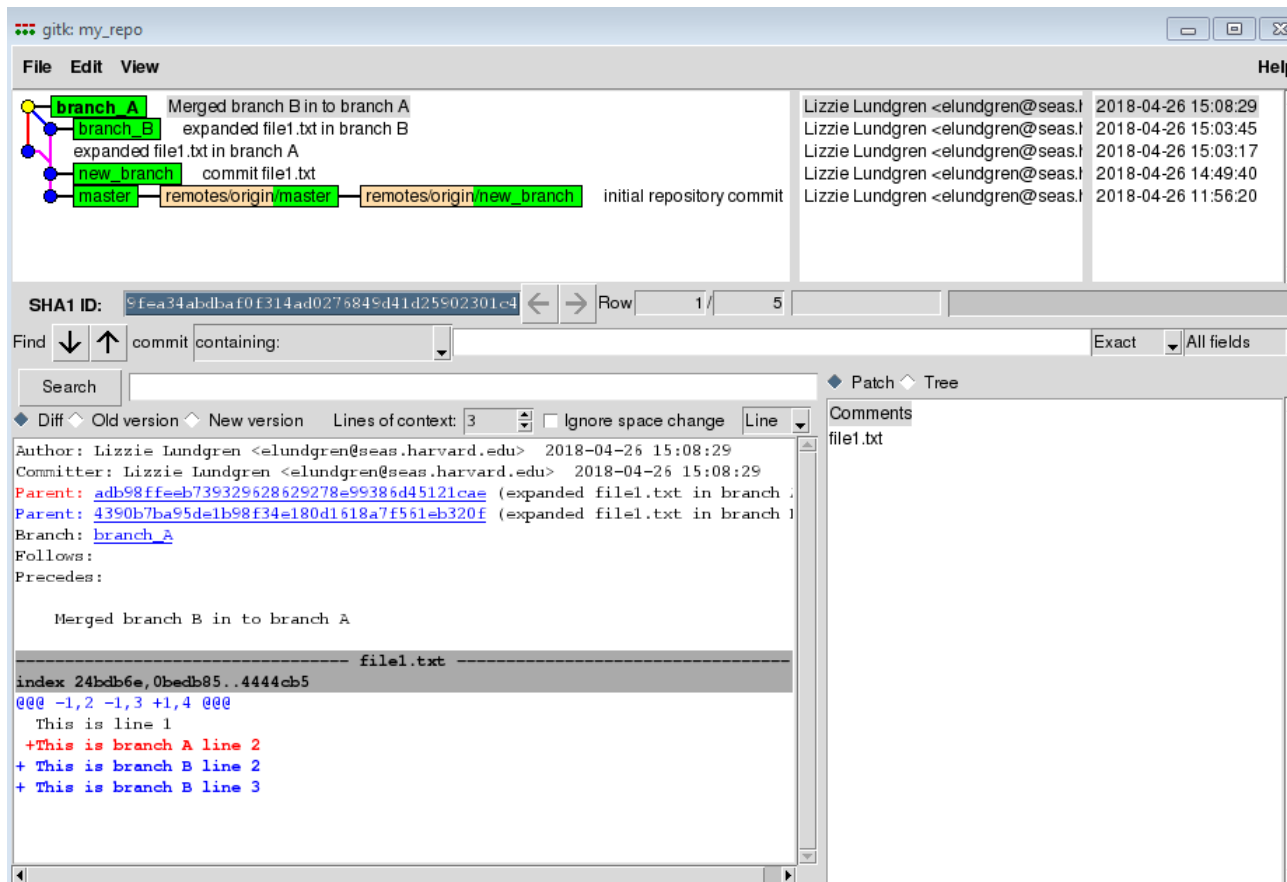
```
/n/home08/elundgren/my_repo/file1.txt
File Edit Options Buffers Tools Help

This is line 1
This is branch A line 2
This is branch B line 2
This is branch B line 3
```

Inspect the Merge

- Merging can be messy. It is good to check your work.
- Use gitk for this purpose

```
$ gitk --all &
```



The screenshot shows the gitk GUI for a repository named 'my_repo'. The top left pane displays a commit graph with the following nodes and descriptions:

- branch_A**: Merged branch B in to branch A
- branch_B**: expanded file1.txt in branch B
- new_branch**: commit file1.txt
- master**: remotes/origin/master
- remotes/origin/new_branch**: initial repository commit

The top right pane shows a list of commit messages and timestamps:

Commit Message	Timestamp
Lizzie Lundgren <elundgren@seas.l	2018-04-26 15:08:29
Lizzie Lundgren <elundgren@seas.l	2018-04-26 15:03:45
Lizzie Lundgren <elundgren@seas.l	2018-04-26 15:03:17
Lizzie Lundgren <elundgren@seas.l	2018-04-26 14:49:40
Lizzie Lundgren <elundgren@seas.l	2018-04-26 11:56:20

The middle pane shows the SHA1 ID: `5f-a34abdbaf0f314ad0276849d41d25902301c4`. Below it is a search bar with the text 'commit containing:'. The bottom pane shows the diff view for 'file1.txt' with the following content:

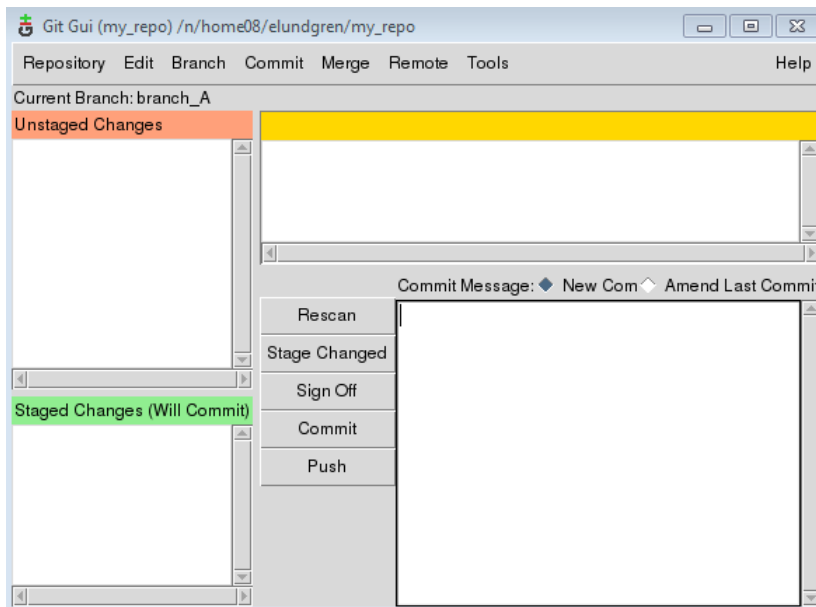
```
Author: Lizzie Lundgren <elundgren@seas.harvard.edu> 2018-04-26 15:08:29
Committer: Lizzie Lundgren <elundgren@seas.harvard.edu> 2018-04-26 15:08:29
Parent: adb98ffeab739329628629278e99386d45121cae (expanded file1.txt in branch
Parent: 4390b7ba95de1b98f34e180d1618a7f561eb320f (expanded file1.txt in branch
Branch: branch_A
Follows:
Precedes:

Merged branch B in to branch A

----- file1.txt -----
index 24bdb6e,0bedb85..4444cb5
@@@ -1,2 -1,3 +1,4 @@@
This is line 1
+This is branch A line 2
+ This is branch B line 2
+ This is branch B line 3
```

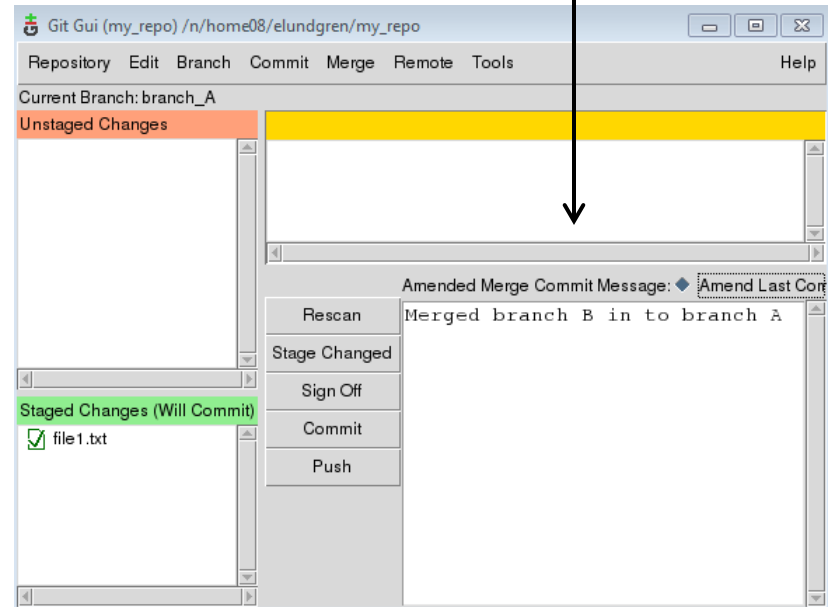
Amend a Commit

- My last commit had 'in to' but I want it to be 'into'
- You can update the last commit, but be careful



Click here ("Amend Last Commit")

Now I can edit my previous commit message and to overwrite it



Do NOT do this if you have pushed to the remote!!!

Resources

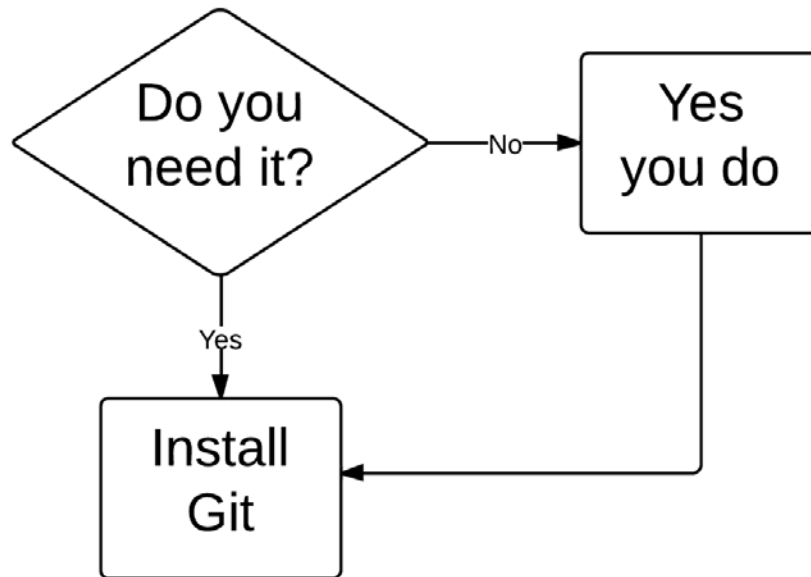
- Tons of info here: <https://git-scm.com/>
- Pro Git by Scott Chacon is a great intro textbook
 - Download for free: <https://git-scm.com/book/en/v2>
 - Borrow it from me if you prefer the feel of paper
- Google anything related to git and you will get an answer
- For command line access to the git user manual:

```
$ git command --help
```

There are lots of advanced topics not covered in this presentation. Use the web, use each other, and come find me if you need help.

Questions?

Version Control Flowchart



GitHub Price Plans

GitHub is free to use for public and open source projects.
Work together across unlimited private repositories with a paid plan.

Developer	Team	Business	
<p data-bbox="241 632 320 696">\$7</p> <p data-bbox="220 721 341 743">per month</p> <p data-bbox="220 799 341 822">Includes:</p> <ul data-bbox="115 843 450 986" style="list-style-type: none">Personal accountUnlimited public repositoriesUnlimited private repositoriesUnlimited collaborators <p data-bbox="106 1048 459 1110">Free for students as part of the Student Developer Pack.</p>	<p data-bbox="716 632 795 696">\$9</p> <p data-bbox="656 721 855 743">per user / month</p> <p data-bbox="693 799 815 822">Includes:</p> <ul data-bbox="589 843 923 986" style="list-style-type: none">Organization accountUnlimited public repositoriesUnlimited private repositoriesTeam and user permissions <p data-bbox="579 1048 933 1110">Starting at \$25 / month which includes your first 5 users.</p>	<p data-bbox="1170 632 1280 696">\$21</p> <p data-bbox="1130 721 1329 743">per user / month</p> <p data-bbox="1151 792 1307 855">Hosted on GitHub.com</p> <ul data-bbox="1079 875 1379 1089" style="list-style-type: none">Organization accountSAML single sign-onAccess provisioning24/5 support with 8-hour response time99.95% Uptime SLA	<p data-bbox="1599 632 1709 696">\$21*</p> <p data-bbox="1566 721 1765 743">per user / month</p> <p data-bbox="1595 792 1736 855">GitHub Enterprise</p> <ul data-bbox="1489 875 1843 1129" style="list-style-type: none">Multiple organizationsSAML, LDAP, and CASAccess provisioning24/7 support for urgent issuesAdvanced auditingHost on your servers, AWS, Azure, or GCP
<p data-bbox="229 1210 334 1233">Sign up</p>	<p data-bbox="629 1210 880 1233">Sign up your team</p>	<p data-bbox="1155 1210 1302 1233">Get started</p>	<p data-bbox="1559 1210 1769 1233">Start a free trial</p>

Oops, I Committed a Huge File!

- Pushing a huge file to the remote means its history (and storage footprint) will remain for eternity
- But there are things you can do, all of which are messy
- If it's a solo project:
 - You can delete the file and rebase to eliminate the commit from history
 - It's rather messy. Seek solutions others have developed and posted online
- If it's a collaboration but no one has fetched from the remote yet:
 - Same approach as for a solo project
 - Make sure you warn everyone not to clone, fork, fetch, pull, or push
- If it's a collaboration and your collaborators have your change:
 - Eek! Rebasing will mess with everyone's local repository
 - You might mess up your collaborator's changes
 - Inform everyone of the problem
 - See advice from your git expert friends