git
What is git?

- Distributed version control system (DVCS).
  - Version control = history of development
  - Distributed = everyone has a copy of history
- Written by Linus Torvalds for managing the Linux Kernel source.
- A way of talking about revisions in tree form.
git for scientists

- in addition to the article than the one I sent out.

online git tutorial

- https://try.github.io
git for computer scientists

- git stores “blobs” of data
- trees are directories of blobs, can point to subtrees (subdirectories)
- commits point to trees and other commits.
Separate commits with shared trunk = branches
Four locations

• Working directory: the files you work on.

• The Index: changes that are going to be added to local repository.

• Local Repository: revision history on your machine.

• Remote repository: revision history on another machine.
Git Data Transport Commands
http://osteele.com

commit -a
add (-u) → commit
push

workspace
index
local repository
remote repository

pull or rebase
fetch

checkout HEAD
revert
compare
diff HEAD
checkout
diff
Each of the four places might have a different version! (And repos might have multiple versions!)
Local vs. Remote

• Two graphs of changes.

• Shared history? Shared trunk in the graph.

• “Repo” = repository.
git init

• Get a directory ready for being a git repository.

git clone <URL>

• Clone a remote repository.

• Including branches (more on branches later)
Working alone

• “git init” a local repository

• While True:
  HACK HACK HACK HACK
  git add files
  git commit -m “hack iteration N”

• I can go back in time, in case I make a mistake, to remember what I was doing, ...
git add <file>

• Prepare <file> to be committed.

• Copies <file> to the Index (Index is local-only).

git commit -m "<Message>"

• Everything in the Index -> Local Repo

• <Message> describes what was done.
Most basic git usage
master

- The default name of the initial branch.
branches

• branches are just names for certain points in the revision history.

• The revision history on branches can diverge, which is where they get their name
branches

Three branches.
git branch <branch_name>

- Create a new branch called <branch_name>, from the current HEAD.
- HEAD = special name for the current active branch.
- Does not switch HEAD to new branch.

```bash
git checkout <branch_name>
```

- Switch HEAD to new branch, and update working directory.
HEAD points to master.

If we run `git checkout very_nice_feature`...
HEAD points to very_nice_feature, and the Working Directory will have whatever files match that branch.
git checkout -b <branch_name>

- Create and move HEAD to new branch.
Branches can be merged.
Using topic branches

- Find a bug
- `git checkout -b fix_a_bug`
- Loop:
  - hack hack hack
  - `git add <modified files>`
  - `git commit -m "fix step #N"`
  - test
- `git checkout master`
- `git merge fix_a_bug`
git merge <branch>

- Merges history of current HEAD and <branch>

- “Fast-forward” = no changes on HEAD since <branch> diverged.

- If not “Fast forward” -> might run into merge conflicts (beyond the scope of this tutorial).
Sharing your work

• Ready to put your work online?

• Create an online repo (e.g., on github)

• git remote add origin <URL> (once)

• git push origin master (each time you update)
git remote

• List your remotes (often only origin)

```bash
git remote add <name> <URL>
```

• Add a new remote (with a name, at URL)
**git push <remote> <branch>**

- Take any changes I’ve committed, and push them to a remote on that branch.

- **Must be a fast forward.**

**git fetch <remote>**

- Copy any changes from <remote> to my local repository, **but do not change any of my work.**
git merge <remote>/<branch>

- Merge changes from a remote branch (previously fetched) into current HEAD.

git pull <remote>/<branch>

- combined fetch & merge.

- I prefer separate fetch & merge.
git diff

- Diffs between working directory and Index.


git diff <commit>

- Diffs between working directory and <commit>.


git diff <c1> <c2>

- Diffs between two commits.
Safe fetch & merge

- git fetch origin
- git diff origin/master
- ok? git merge origin/master
- not ok? … something else… (often commit, then merge anyway, and deal with problems)
Dealing with merge conflicts

- Basically, edit files with conflicts (list them with `git status`), fix the conflicts, then `git add` those files. `git commit` when done (no message).
- Don’t panic.
- Search Google.
- Ask a TF.
gitk

- Useful tool, especially for seeing all the branches.
- `gitk --all`
Questions?