#the key is in ~/.ssh/id_rsa

- `git help <command>`: get help for a git command
- `git init`: creates a new git repo, with data stored in the `.git` directory
- `git status`: tells you whatâ€™s going on
- `git add <filename>`: adds files to staging area
- `git commit`: creates a new commit
  - Write good commit messages!
  - Even more reasons to write good commit messages!
- `git log`: shows a flattened log of history
- `git log --all --graph --decorate`: visualizes history as a DAG
- `git diff <filename>`: show changes you made relative to the staging area
- `git diff <revision> <filename>`: shows differences in a file between snapshots
- `git checkout <revision>`: updates HEAD and current branch

Branching and merging

{% comment %}

Branching allows you to â€œforkâ€ version history. It can be helpful for working on independent features or bug fixes in parallel. The `git branch` command can be used to create new branches; `git checkout -b <branch name>` creates and branch and checks it out.

Merging is the opposite of branching: it allows you to combine forked version histories, e.g.Â merging a feature branch back into master. The `git merge` command is used for merging.

{% endcomment %}

- `git branch`: shows branches
- `git branch <name>`: creates a branch
- `git checkout -b <name>`: creates a branch and switches to it
  - same as `git branch <name>; git checkout <name>`
- `git merge <revision>`: merges into current branch
- `git mergetool`: use a fancy tool to help resolve merge conflicts
- `git rebase`: rebase set of patches onto a new base

Remotes

- `git remote`: list remotes
- `git remote add <name> <url>`: add a remote
- `git push <remote> <local branch>:<remote branch>`: send objects to remote, and update remote reference
- `git branch --set-upstream-to=<remote>/<remote branch>`: set up correspondence between local and remote branch
- `git fetch`: retrieve objects/references from a remote
- `git pull`: same as `git fetch; git merge`
- `git clone`: download repository from remote

Undo

- `git commit --amend`: edit a commitâ€™s contents/message
- `git reset HEAD <file>`: unstage a file
- `git checkout -- <file>`: discard changes

Advanced Git

- `git config`: Git is highly customizable
- `git clone --depth=1`: shallow clone, without entire version history
- `git add -p`: interactive staging
- `git rebase -i`: interactive rebasing
- `git blame`: show who last edited which line

- `git stash`: temporarily remove modifications to working directory
- `git bisect`: binary search history (e.g.Â for regressions)
- `.gitignore`: specify intentionally untracked files to ignore

Miscellaneous

- **GUIs**: there are many GUI clients out there for Git. We personally donâ€™t use them and use the command-line interface instead.
- **Shell integration**: itâ€™s super handy to have a Git status as part of your shell prompt (zsh, bash). Often included in frameworks like Oh My Zsh.
- **Editor integration**: similarly to the above, handy integrations with many features. fugitive.vim is the standard one for Vim.
- **Workflows**: we taught you the data model, plus some basic commands; we didnâ€™t tell you what practices to follow when working on big projects (and there are many different approaches).
- **GitHub**: Git is not GitHub. GitHub has a specific way of contributing code to other projects, called pull requests.
- **Other Git providers**: GitHub is not special: there are many Git repository hosts, like GitLab and BitBucket.

Resources

- Pro Git is **highly recommended reading**. Going through Chapters 1â€“5 should teach you most of what you need to use Git proficiently, now that you understand the data model. The later chapters have some interesting, advanced material.
- Oh Shit, Git!?! is a short guide on how to recover from some common Git mistakes.
- Git for Computer Scientists is a short explanation of Gitâ€™s data model, with less pseudocode and more fancy diagrams than these lecture notes.
- Git from the Bottom Up is a detailed explanation of Gitâ€™s implementation details beyond just the data model, for the curious.
- How to explain git in simple words
- Learn Git Branching is a browser-based game that teaches you Git.

Exercises

1. If you donâ€™t have any past experience with Git, either try reading the first couple chapters of Pro Git or go through a tutorial like Learn Git Branching. As youâ€™re working through it, relate Git commands to the data model.
2. Clone the repository for the class website.
    1. Explore the version history by visualizing it as a graph.
    2. Who was the last person to modify `README.md`? (Hint: use `git log` with an argument).
    3. What was the commit message associated with the last modification to the `collections:` line of `_config.yml`? (Hint: use `git blame` and `git show`).
3. One common mistake when learning Git is to commit large files that should not be managed by Git or adding sensitive information. Try adding a file to a repository, making some commits and then deleting that file from history (you may want to look at this).
4. Clone some repository from GitHub, and modify one of its existing files. What happens when you do `git stash`? What do you see when running `git log --all --oneline`? Run `git stash pop` to undo what you did with `git stash`. In what scenario might this be useful?
5. Like many command line tools, Git provides a configuration file (or dotfile) called `~/.gitconfig`. Create an alias in `~/.gitconfig` so that when you run `git graph`, you get the output of `git log --all --graph --decorate --oneline`. You can do this by directly editing the `~/.gitconfig` file, or you can use the `git config` command to add the alias. Information about git aliases can be found here.
6. You can define global ignore patterns in `~/.gitignore_global` after running `git config --global core.excludesfile ~/.gitignore_global`. Do this, and set up your global gitignore file to ignore OS-specific or editor-specific temporary files, like `.DS_Store`.
7. Fork the repository for the class website, find a typo or some other improvement you can make, and submit a pull request on GitHub (you may want to look at this). Please only submit PRs that are useful (donâ€™t spam us, please!). If you canâ€™t find an improvement to make, you can skip this exercise.