GitHub / git

Version Control

Marina Schmidt,
SuperDARN Canada, Usask

SUPERDARN CANADA

UNIVERSITY OF SASKATCHEWAN

Outline

- Beginner level:
    - What is Version Control?
    - Creating a repository
    - Four holy commands in git
    - Workflow - branching out
- Intermediate level:
    - Pull it, Push it, Merge it, Rebase it
    - Proofread the code
    - Project management for collaborations
    - Templates
- Advanced level:
    - Sub-repo it!
    - Release the code!
    - Hooking up with GitHub

What is Version Control?

" Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer. " - git docs

git SVN CVS

git it? got it? GitHub

git - Is a software tool to version control your code

GitHub - Is a web based graphical user interface (GUI) for git. It allows the git server to be hosted on GitHub website server and add many additional features for smoother collaboration. Other web based/hosting GUI's for git include:

BitBucket

GitLab

Getting Started - Read the README.md

Once you have a GitHub account you can create a repository on your own account or on an organization's account. How you want to share the code and how many people are collaborating on it will inform where you want to create it.

GitHub will ask questions to help create your repository, including adding these 3 types of files:

1. **README.md** - README describes the repository and how one might use/install it. (normally written in markdown '.md')
2. **LICENSE** - licenses determines how users will interact, use, and modify your code.
3. **.ignore** - PLEASE INCLUDE! This will prevent any unwanted binary, data files, and folders to be added to repository. In python this would be `pcy` or `__pycache__`. In GitHub if you specify the language it will autocomplete the file for you

Choose a LICENSE!

GPL - copyleft license
Permissions:
- Commercial use
- Distribution
- Modifications
- Patent use
- Private use

Conditions:
- Disclose source
- License and copyright notice
- Same license
- State changes

MIT - copyright license
Permissions:
- Permissive
- Distribution
- modifications
- Private use

Conditions:
- License and copyright notice

https://choosealicense.com/

SUPERDARN CANADA

UNIVERSITY OF SASKATCHEWAN

Clone vs. Fork

git clone

Adding to the current repo

git fork

Copying to your repo to make your own changes and not adding them to the original repo

SUPERDARN CANADA

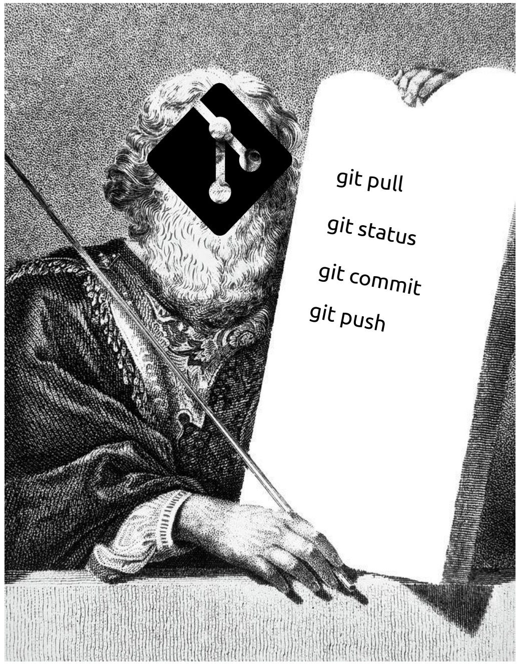
UNIVERSITY OF SASKATCHEWAN

Beam me up git!

Remote means saving changes to the hosting server like github. When you “git push” you are pushing changes to the shared repository server.

Local means saving changes on your own computer. When you “git commit -m” you are saving the changes in your local git history. No one else can see these changes.

4 Holy git commands



git pull - update any changes others have made (this includes yourself)

git status - check which files you have changed

git commit - commit changes to your local git history and leave a update your local repo to include informative message. Avoid *-m* if possible to ensure detailed messages

git push - push changes to the remote git repository

Holy git Commands in Action!

```
$ git status
On branch plot/ACF
Your branch is up to date with 'origin/plot/ACF'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   acf.py
$ git add acf.py
$ git commit -m “fixed the typo in the other name”
$ git push origin plot/ACF
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
...
```

SUPERDARN CANADA

UNIVERSITY OF SASKATCHEWAN

Branching Out

git fetch	Obtains the most recent changes to branches, including new branches from the remote server
git checkout	To checkout a branch. This command will switch branches and upload any changes made in that branch if git fetch is done beforehand. The -b option allows you to create a new branch **based on the current one you are on.**
git branch	Lets you know which branch you are on and available branches. This command is useful to use before “git checkout -b <new branch name>

Branching Out Example

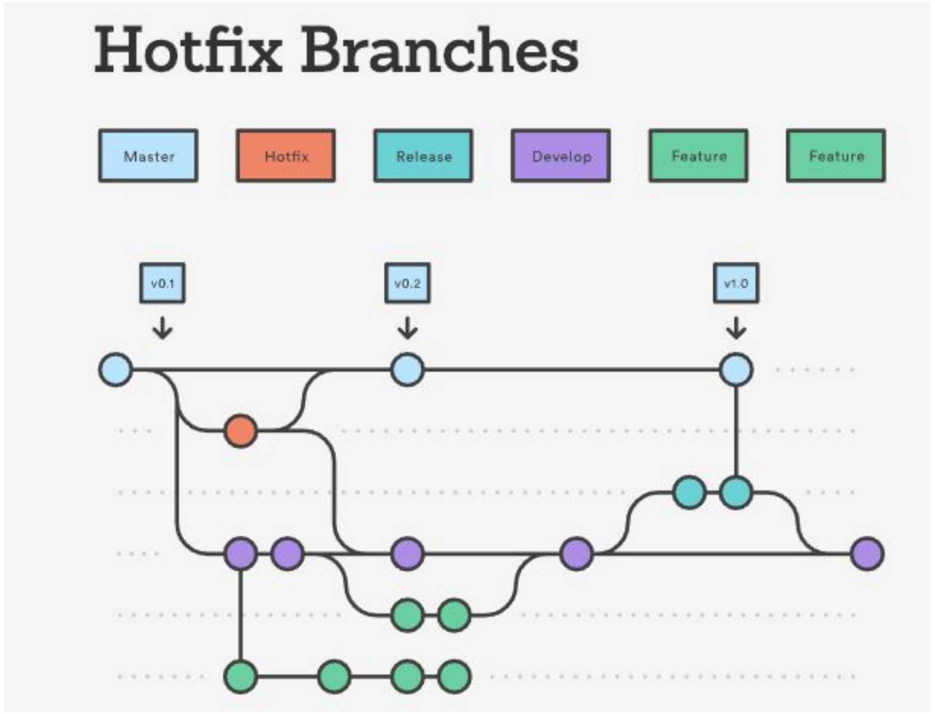
```
$ git branch
  bug/rawacf_updates
  develop
  master
* plotting/fan
$ git fetch
remote: Enumerating objects: 231, done.
remote: Counting objects: 100% (218/218), done.
remote: Compressing objects: 100% (66/66), done.
remote: Total 184 (delta 115), reused 157 (delta 94), pack-reused 0
Receiving objects: 100% (184/184), 438.21 KiB | 2.29 MiB/s, done.
Resolving deltas: 100% (115/115), completed with 23 local objects.
From https://github.com/superdarn/pydarn
   c61e06d..6b31454  develop     -> origin/develop
$ git checkout plot/ACF
Branch 'plot/ACF' set up to track remote branch 'plot/ACF' from 'origin'.
Switched to a new branch 'plot/ACF'
```

Branching Out

Master branch is the main branch where a release will be or the most stable branch you currently have for others to use.

Develop branch is the secondary branch in which new unstable features can be added and be well-tested before merging to **main/master** for a release.

Feature branch is a third branch, used for new and potentially unstable features. It branches off **develop** and makes changes to then be merged in



Why Not Stick to Master?

Benefits of branching:

- Keeping branch changes small allows very better debugging and review
- Keeps dangerous buggy code from being used by users
- Keeps things organized
- Has a nice workflow with github and Pull Requests

How to get organised with branches?

- Create a branch based on one goal or part of a goal
- Once completed, create many smaller branches off of it for other tasks to be completed
- Create a Pull Request in GitHub to merge your branch(s) into develop or other branches.
- Pull Requests allow you to see changes you made between files as well as have the nice capability of reviewing your code

Pull it, Push it, Merge it, Rebase it

git pull	Pull on your own branch only this will update any changes. Avoid pulling from another branch to a different branch as this will overwrite changes to your files.
git push	Is to save your commits to the remote server. You should pull before to ensure you have the most recent branch edits.
git merge	To bring changes from another branch or to push changes from your branch to another use merge. This will avoid overwriting any changes.
git rebase	To clean up and choose git commit history. This helps with conflict history commits or meaningless commit messages that you made but ended overwriting. Ideally good to use after a merge. GitHub support merge and rebase when you create a Pull Request.

SuperDARN CANADA

University of Saskatchewan

Proof Reading the Code - Code Review

Code Reviews - are essentially allowing someone or yourself to proof read the changes you have made. This helps with consistency, potential error, good documentation/commenting, and feedback.

Do you need to be a good programmer to code review?

No, good code should be well descriptive and commented to allow the reviewer knowing what is going on

Do I need to know how to program in that language?

No, sometimes scientist are better because they can ensure the science is done right. Plus by reading others code it will help you learn to code better to.

GitHub and Code Reviews

GitHub allows a nice graphical interface and nice features to code review so that anyone can do it:

1. Open a PR on GitHub
2. Select “files changed”, top left corner
3. Look at code changes highlighted in green (added) and red (removed)
4. Once you have a comment for a line slide your mouse curse to the line number and blue “+” will appear and click on it
5. Leave a comment in a box, you can also suggest a change via:
   ```` ```suggestion ````
   `<change>`
   ```` ``` ````
6. Click start review to avoid spamming them with emails
7. Once you are done go to the top right and select “Finish your review”
8. Add any extra comments about the review and select what is required from these changes:
   a. Comment
   b. Approve
   c. Request Changes - this will prevent the developer/users to merge the code until you approve the changes

Project Management

Labels - In GitHub you can label *issues* and *pull requests* this is to help organize *issues* and *pull requests* in categories for a user/developer to pick up. You can also create your own labels.

Projects - GitHub on the main tab list next to *Actions* is *projects* which allows a user to generate a workflow for given tasks/goals. Once you create one you will be able to select a template - recommend Automated KanBan.

Kanban - is developer style of organizing the process/workflow of tasks via these categories:

- TODO - List tasks and notes of things to be done for this project, this can include *issues*
- In Progress - Once a task or *issue* is picked up and worked on it goes to in progress. Typically *pull requests* will be here.
- In Review - once a *pull request* is created and reviewed/tested it will end up here
- Done - the *pull request* is merged and the *issue* is closed.

Please note projects do not include deadlines and they can be endless lists/tasks as they focus on an idea.

Milestones - Are like projects but created with deadlines and have no templates associated to them. To create a milestone go to the *pull request* tab then click **Milestones** next to *New pull request.* Milestones are good for code release or features that have a deadline. Once you created a Milestone you can add *pull requests* and *issues* to it. The Milestone will show the status of how many tasks are open and closed, and a status bar of completion.

Templates

Templates - are forms you can create for *issues* and *pull requests* to ensure users place the correct information in for given categories like:

- New Features
- Bug
- Documentation

To create a template:

1. create a .github folder in your main repository directory
2. create sub-folders in .github named:
   a. "ISSUE_TEMPLATE" and
   b. "PULL_REQUEST_TEMPLATE"
3. In either of the sub-folders start creating markdown files (.md) for a template

Git Inception

git sub-repo is a repository that is used in another repository. That advantage of using sub-repos is:

- Modularize
- Keep commits separate between packages
- Decoupling

HOW to do it:

1. Have 2 seperate repositories
2. In your working repository you would like to add the other repo:

```
git submodule add <other repo url link>
```

3. To update the sub-repo:

```
git submodule update --init --recursive
```

```
git submodule update --remote --recursive
```

4. Bonus! add these commands to setup.py, your sub-repo downloads on install

Release the Code!

1. Go to the main page of the repository
2. Click on **releases** to the top right of the page
3. Click **Draft a new release**
4. Type the next version number of your code: *major.minor.pathch:*
   a. *Major:* large changes to the code base like structural changes
   b. *Minor:* additive feature or deprecations to the code
   c. *Patch:* bug fixes, minor small changes and updating some code
5. Select the target branch, typically **main** also known as **master**
6. Then give details on your release:
   a. Title - typically release version number
   b. Details - typically what changes, fixes, documentation, and deprecations have been made
7. You can add pre-compiled code to your release but python usually does use this feature
8. You can select pre-release if you still need to test the code to ensure stability
9. Click **Publish Release!**

Hooking up to the web - Webhooks/Integrations

Readthedocs - hosting platform for building, versioning, and documentation

DOI's and hosting platform for data and software for citing in publications

Multi user chatting app

Thank You!

Helpful Links

Create Repo on GitHub:
https://help.github.com/en/enterprise/2.17/user/github/creating-cloning-and-archiving-repositories/creating-a-new-repository
Choose A License:
https://choosealicense.com/
Code Review in GitHub:
https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-request-reviews
Merge vs. Rebase:
https://www.atlassian.com/git/tutorials/merging-vs-rebasing
Releasing code:
https://help.github.com/en/enterprise/2.13/user/articles/creating-releases
SSH-key gen for GitHub for faster pushes:
https://help.github.com/en/enterprise/2.17/user/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account

SUPERDARN CANADA

UNIVERSITY OF SASKATCHEWAN