Welcome to The Carpentries Etherpad!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try https://etherpad.wikimedia.org).

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

---

This Etherpad will be archived on GitHub after the session. Please remove any personal information if you don't want it to be archived.

Back to main CC@Home Etherpad: https://pad.carpentries.org/carpentryconathome

CC@Home: Git skills for new and prospective maintainers

July 16, 16:00 (UTC)
https://www.timeanddate.com/worldclock/fixedtime.html?msg=Git+skills+for+new+and+prospective+maintainers&iso=20200716T16&p1=1440&ah=1
~~August 5, 12:00 (UTC)~~
~~https://www.timeanddate.com/worldclock/fixedtime.html?msg=Git+skills+for+new+and+prospective+maintainers&iso=20200805T12&p1=1440&ah=1~~
The August 5 session is being postponed due to a power outage for the presenter (caused by Tropical Storm Isaias). Please sign up below if you interested in being contacted about a new time.
September 9, 15h00 UTC

Presenter: Daniel Chen
Duration: 3.5 hours (interactive workshop)
Live stream, September 9: https://www.twitch.tv/chendaniely
Recording Link: https://www.youtube.com/watch?v=uvWhSYBkZJ0

Host/Call details

- Connection link session 1: https://carpentries.zoom.us/my/carpentriesroom3
- Connection link session 2: https://www.twitch.tv/chendaniely
- Session Regional Coordinator: Session 1: Angela Li / Christina Koch, Session 2: Daniel Chen
- Link to Regional Host, Co-host/CoC Facilitators and Speakers Guide:
    - https://github.com/carpentrycon/carpentryconhome-proposals/blob/master/CCatHome_Regional_Host_Guide.md
- Code of Conduct:

    - https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html
- Feedback survey:
    - https://carpentries.typeform.com/to/eHSDFITR
- Dan's Github repository for today's workshop: https://github.com/chendaniely/2020-07-16-CCatHome-git-dan

Roll call + Check in (Session 2)

Put your name/email here if you're interested in attending the re-scheduled second session

at https://www.twitch.tv/chendaniely

- Adam Hughes / jahughes@uark.edu
- Trevor Burrows / dburrows@purdue.edu
- Rwasimitana /Fabricerwasimitanafabrice@gmail.com/he/@Rwasimitana1
- Annajiat Alim Rasel / annajiat@gmail.com
- Rachel Lombardi / rllombardi@ucdavis.edu
- Agustin Falomir Lockhart / agusfalomir@gmail.com
-
- Yuri Kotliarov / yuk4pub@gmail.com
- Julia Marchetti / julimarchetti@gmail.com

Roll call + Check in (Session 1)

- name / email / pronoun / social media handle / github username
- Daniel Chen / chend@vt.edu / /he, him / @chendaniely / chendaniely
- ~~Allison Theobold; atheobol@calpoly.edu; she/her/hers; @mtstatistics~~
- Jaki Frisina, jaki.frisina@leblibrary.com, she/her/hers
- ~~Muhammad Zohaib Anwar, zohaib_anwar@ymail.com, he/him/his, @xohaib_anwar~~
- Erin Becker / ebecker@carpentries.org / she/her/hers
- Sarah Lin / sarah.lin@rstudio.com / she/her/hers / @sarahemlin
- Stefanie Butland / stefaniebutland@gmail.com / she/her/hers / @stefaniebutland / @stefaniebutland
- Joakim Philipson / joakim.philipson@su.se / he, him, his / @JoakimPhilipson / jomtov
- James Koch / jchkoch@gmail.com / he/him/his / @jchkoch
- Elizabeth McAulay / emcaulay@library.ucla.edu / she
- Jannetta Steyn / jannetta.steyn@newcastle.ac.uk / she / @jannettas / jsteyn
- Mike Renfro / renfro@tntech.edu / he / @mikerenfro / mikerenfro
- Sandeep Mallya / sandeep.mallya@manipal.edu / he / @PSandeepMallya / biowizz
- Rwasimitana /Fabricerwasimitanafabrice@gmail.com/he/@Rwasimitana1
- Ivo Arrey / 11624476@mvula.univen.ac.za / he / @arreyves
- Drake Asberry / dasberry@email.arizona.edu / he/him/his / @AsberryDrake / @drakeasberry

- Tobin Magle/ tobin.magle@wisc.edu / she/them / @tobinmagle
- Preethy Nair
- Abhinav Mishra/ mishraabhinav36@gmail.com/ he, him / @shadytoyou (Twitter)

Notes:

This workshop is essentially the continuation of the SWC Git lesson: https://swcarpentry.github.io/git-novice/

Picture from class is on GitHub:
https://github.com/chendaniely/2020-07-16-CCatHome-git-dan/blob/master/whiteboard.png

Carpentries blog post on how to fix someone's PR: https://carpentries.org/blog/2020/04/maintainers-git-skills/

    - Software Carpentry notes on basic git: https://swcarpentry.github.io/git-novice/
        - The setting up git has the commands to setup your name/email/editor: https://swcarpentry.github.io/git-novice/02-setup/index.html

- Review of cloning a repository and pushing a commit
    - Best workflow: create empty repository on Github/BitBucket etc, then copy ("clone") to your computer
        - This workflow ("cloning a repo") is not covered in the Software Carpentry instructions as it mainly shows how to create a Git repo locally and link it to Github. We will be doing "Github first".
    - We start with creating a new repository, give it a name "2020-07-16-CCatHome-git-[yourname]"
        - Leave the as repo public, click "Initialize with README"
        - Click the "Code" button and copy the HTTPS URL
        - Open up the Terminal or shell in your local computer
    - Don't clone git repo into another one! Make sure you don't have nested git repositories (just like "git init" you only do it once ever)
        - Dan recommends creating a folder called "git" in your root directory, then cloning things into there
    - To open your Finder in Mac / file explorer in Windows, type "open ." or "explorer ." or "xdg -open ." in Linux
    - To clone, "git clone [URL]" = downloads the repo from the web
    - cd into that directory
    - nano README.md -- add some notes about what you're doing
    - git status
    - git add README.md
    - git commit -m"talked about git clone"
    - git remote -v (this shows the remote Github URL)
    - git push origin master
- Branching / Making a branch:
    - Why use a branch? It enables you to go back to a pre-defined point before you try some experimental things, and it allows you to collaborate confidently without!
    - Branch = A label for a series of commits
        - **git --version** (some of the prompts in the newer version are going to be a bit

different - you don't have to update Git now - or ever! - but the prompts may not be the exactly the same, FYI)
        - **git branch my_first_branch** (create a branch)
        - **git branch -a** (list all branches, the one that is green/has a star next to is the one you're on)
        - **git status** (you see that "On branch master" is the branch you're on)
            - **git log --oneline --graph --decorate --all** (this shows you the entire tree)
        - **git checkout my_first_branch** (standard way to checkout branch), new command is **git switch my_first_branch** (git switch will only work if you installed Git after August 2019)
    - Repeating the push workflow...
        - **nano README.md** (let's add a note to the README about what we learned)
        - **git status**
        - **git diff README.md**
        - **git add README.md**
        - **git commit -m "A commit on a branch!"**
        - **git log --oneline --graph --decorate --all** (this shows you the entire tree)
            - you can make this shorter by creating git aliases
            - https://git-scm.com/book/en/v2/Git-Basics-Git-Aliases
            - git config --global alias.l "log --oneline --graph --decorate --all"
        - **git push origin my_first_branch** (note that it's not "origin master" - if you're not used to working with branches, you may be tempted to type "origin master")
    - Good tip from Dan: if you can draw out the Git problem you're having visually - on a whiteboard, chances are you can figure out the command you need to fix things!
    - We then merged a PR from our branch into master, issuing a review on Github in the process, then deleted our extra branch
- Pulling our merged changes from Github:
    - **cat README.md**
    - **git pull origin master**

Exercise 1
1. create a new branch
2. go to the new branch
3. edit README.md with text
(talkl about fetch --prune,
and the log command and
deleting branches
)

4. push branch to github
5. create pr
6. merge PR
7. update and clean up your branches on
local computer

- Exercise 1 answers:
    - **git checkout -b creating_branches** (this allows you to create + switch to the branch in

one go)
        - **nano README.md** (open up text editor to add a note)
            - **git fetch --prune** (will update your local git tree with the remote. this will also delete references to branches that were deleted on the remote)
            - git stash (saves your temp changes when you switch branches)
            - git stash apply (apply your stashed changes)
        - git add / git commit / git push workflow
        - Merge PR on Github to master (you can even leave yourself a review!), then come back to local shell to pull changes and delete branch
        - git checkout master
        - git fetch --prune
        - git branch -a
        - git pull origin master
        - git branch -d creating_branches (delete local copy of remotely deleted branch)
- When someone needs to make changes to their PR, they can continue to work on branch, do not need to create a new PR
- When you merge PRs, you generally want "create a merge request" aka the default - "squash and merge" creates pretty commits, "rebase and merge" is useful when you have merge conflicts

Rebasing

- This is useful when you have merge conflicts
- git rebase
- git rebase --abort (will bring things back to normal, safety hatch for rebasing)
- git rebase --continue
    - this takes care of what is on your local computer
    - to update on your remote (on github) - can't use git push origin change_title_2
        - this will give you an error because the history is different, but this is actually what you want to do, so use -f (force) flag
        - git push -f origin change_title_2
- then do your cleanup steps
    - git pull origin master
    - git fetch --prune
    - git branch -d change_title_2

Different workflows

- Centralized workflow - everyone who is working on the project has merge access!
    - Problem with this is that everyone is working on the master branch.
    - No enforcement of code review.
    - Can enforce code review on certain branches in GitHub settings
- Feature branch workflow - no one can push directly to master
    - Requires code review
- Git flow - standardized way of naming branches
    - master branch stores official release history - tagged with version number
    - develop branch serves as integration branch for features
    - most work happens on develop branch
- Forking workflow - Requires most collaborators to fork rather than branch
    - This is the workflow used on Carpentries lessons

- People who don't have write access can't make branches, need to fork
- Forking creates a copy of the repo in their own GitHub account - you can do whatever you want with this copy
- Common model for open source projects
- Naming conventions:
    - origin = the remote repository you have push access to
    - upstream = the remote repository that you forked from
- To keep your fork in synch with the official lesson repo - you pull from upstream and you push to origin (and then submit a pull request to upstream)

Using forking workflow:

- Create a fork of the repository you want to work with (top right corner "fork" button)
- Clone to your computer - click "Code", copy the ssh url, in terminal "git clone <url>"
- Set the upstream by using
    - git remote add upstream <url of official repo>
- Now work as you would normally

<u>Questions from participants:</u>

- Could you please explain branching v forking? Not the how, but the why? We have about half who like to fork and half who like to branch and I don't get why some folks do one over the other.
    - Forks are copies of the repo in your own account. Mandatory if you're not part of the organization that owns the original repository. I had to fork to submit my pull requests for checkout.
- i am getting an error message on switch / "git switch" doesn't work for older version
    - Nope, you can use git checkout instead :) git switch is a new function and only works if you installed Git after Aug 2019
- should we update git? can't believe i never even considered it...
    - If you want to use git switch, maybe!
- A currently important question for me right now: can I rename my git repo locally and then still push it to my remote origin that still has the same name?
    - Yes, if you cloned from Git, I think Git has the remote URL stored…(see git remote -v) I know the other way around works, too! You'll just get a message warning you and telling you to change the name
    - A good workflow for renaming:
        - **git remote -v**
        - **git remote rm origin**
        - **git remote add <URL> (the one you copy from the green button on the repo)**
    - The official way to do it: git remote set-url alias url,
    - Or: git remote set-url <>` will update origin
- There is a large controversy over git pull versus a git fetch then git merge within my collaboration circles. Is there a benefit too doing one over the other?
    - git pull does both, generally good enough for all purposes
    - git fetch / merge is if you need to delete references or get references to something (a little more manual)
- Question - I know that git has some automatic assumptions about what origin and master

are if you just do “git pull” or “git push” without specifying. Could you talk about what that is and whether we should use that? I admit I always just do a bare “git pull” or “git push” without specifying origin and master. Should I?
    - There is no difference when you're working on master. If you're just working on master and not branches, just use git push/pull without "origin master" bit
    - However, when you're on a branch, get into the habit of not relying on the default. This forces you to slow down and think, what am I actually doing
- Question - is your fork your remote?
    - Yes - your fork becomes your remote! Will talk about naming conventions for remotes soon.
- Question - I’ve never actually set the upstream when working with the fork workflow on a Carpentries lesson, and everything seems to work . . . (I usually make changes locally, then push to my fork, then make a PR on GitHub). Is there a reason to set the upstream?

Notes / resources from participants:

- Working tree, staging, repo: https://medium.com/@lucasmaurer/git-gud-the-working-tree-staging-area-and-local-repo-a1f0f4822018
- For Carpentries lessons - all Maintainers for that lesson have merge permissions :-) And we’ve also implemented automatic branch deletions after resolve, so you don’t need to worry about deleting branches

- Git workflow notes and commands (to paste on your wall)
    - https://chendaniely.github.io/training_ds_r/help-faq.html
- Atlassian page on git workflows: https://www.atlassian.com/git/tutorials/comparing-workflows

Some nice things to look up to get a nice pretty terminal prompt:

- Getting your (bash) terminal to show current path and other things:
    - Use this to create your PS1 variable: http://bashrcgenerator.com/
- Show git branch in terminal:
    - In addition you can write a function and put that in your PS1 to also show git branch
    - https://gist.github.com/joseluisq/1e96c54fa4e1e5647940

git log --oneline --graph --decorate --all

github teams
ErinBecker
stefaniebutland
drakeasberry