Learning to Love Version Control

Manodeep Sinha

Swinburne University

What will you learn?

- **Why** you should use version control

What will you learn?

- Why you should use version control
- When you should use version control

What will you learn?

- Why you should use version control
- When you should use version control
- How you should use version control

What is version control?

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

Why should you version control?

- Makes tracking your own work easier

Why should you version control?

- Makes tracking your own work easier
- Lets you recover from “oopsies”

Why should you version control?

- Makes tracking your own work easier
- Lets you recover from “oopsies”
- Collaborating is significantly easier (esp. with modern tools)

Why should you version control?

- Makes tracking your own work easier
- Lets you recover from “oopsies”
- Collaborating is significantly easier (esp. with modern tools)
- Note: Version Control is not *backup*. Backups also protect you from hardware failures

How do you know if you need version control?

- If you find yourself making files called XXX_v1, XXX_v2, XXX_final etc

How do you know if you need version control?

- If you find yourself making files called XXX_v1, XXX_v2, XXX_final etc
- If you are wondering how was the code working the hour/day/week before and what broke…

How do you know if you need version control?

- If you find yourself making files called XXX_v1, XXX_v2, XXX_final etc
- If you are wondering how was the code working the hour/day/week before and what broke…
- Everyone could use version control

When should you version control?

- Working on a document (code/paper) that you can not easily recreate

When should you version control?

- Working on a document (code/paper) that you can not easily recreate
- For me, I use git + github as version control + backup. For all substantial work (>= 1 day effort), I create repos first on GitHub and then start working

When should you version control?

- Working on a document (code/paper) that you can not easily recreate
- For me, I use git + github as version control + backup. For all substantial work (>= 1 day effort), I create repos first on GitHub and then start working
- The standard VCS is designed for many developers working simultaneously on parts of a big project. Astronomers are typically 1 person/project - almost zero overhead

When should you NOT version control?

- Large binary files (> few MB)

When should you NOT version control?

- Large binary files (> few MB)
- Where the file does not change (use backup)

What are the version control software?

- git (https://github.com/git/git)

What are the version control software?

- git (https://github.com/git/git)
- mercurial (hg) https://www.mercurial-scm.org/

What are the version control software?

- git (https://github.com/git/git)
- mercurial (hg) https://www.mercurial-scm.org/
- svn

What are the version control software?

- git (https://github.com/git/git)
- mercurial (hg) https://www.mercurial-scm.org/
- svn
- cvs

What are the version control software?

- git (https://github.com/git/git)
- mercurial (hg) https://www.mercurial-scm.org/
- svn
- cvs
- others…probably

git/hg ?

- Choice of VCS may be made for you

git/hg ?

- Choice of VCS may be made for you
- I will recommend git + GitHub ecosystem. hg uses python and I have run into python installation issues

git/hg ?

- Choice of VCS may be made for you
- I will recommend git + GitHub ecosystem. hg uses python and I have run into python installation issues
- Either choice is fine. We will cover only git here

Let’s create a repo

- git init — this tells git that there will be version controlled repository in that directory (and sub-directories)

Add the files

- git add <list of files> — tells git that these named files are the ones that you want to “track”

Add the files

- git add <list of files> — tells git that these named files are the ones that you want to "track"
- This previous command only adds to a "staging" area — and has not yet been "officially" recorded

Commit the files

- git commit -m “message” — tells git that you are happy with all the “staged” changes

Commit the files

- git commit -m “message” — tells git that you are happy with all the “staged” changes
- This command stores the changed files as a differences between the last commit and the current state.

Check the commit was made

- git log — shows the history of all previous commits up to this point (i.e., the messages that you added when committing with “git commit -m <msg>”

Check the commit was made

- git log — shows the history of all previous commits up to this point (i.e., the messages that you added when committing with “git commit -m <msg>”
- Every commit has a weird looking hex number associated (called a hash) that uniquely identifies that commit

Make and commit another change

- git add -u . — tells git to “stage” all changed files that are currently being tracked

Make and commit another change

- git add -u . — tells git to “stage” all changed files that are currently being tracked
- The “-u” is important; otherwise, git will add ALL files that are lying around in that directory and sub-directories

Make and commit another change

- git add -u . — adds to staging area all changes to tracked files

Make and commit another change

- git add -u . — adds to staging area all changes to tracked files
- git commit -m <msg> — commits the changes and records them (you can see that via git log)

How to decide what to add in a commit?

- Commit changes frequently —the beauty is that you can always go back to a previous version if your changes were wrong

How to decide what to add in a commit?

- Commit changes frequently —the beauty is that you can always go back to a previous version if your changes were wrong

- Try to keep commits logically grouped — i.e., separate out commits that solve different aspects of functionality

What if you already have toooooo many uncommitted changes?

- If you do not use version control, this is likely the case

What if you already have soooooo many uncommitted changes?

- If you do not use version control, this is likely the case
- You have two options

What if you already have toooooo many uncommitted changes?

- If you do not use version control, this is likely the case
- You have two options
- git add <list of all relevant files> && git commit -m “Initial commit”

What if you already have toooooo many uncommitted changes?

- If you do not use version control, this is likely the case
- You have two options
- git add <list of all relevant files> && git commit -m “Initial commit”
- or, git add -p — this lets you pick and chose which changes are bundled together into one commit.

github/bitbucket

- At minimum, stores your codes/papers

github/bitbucket

- At minimum, stores your codes/papers
- Also, a backup

github/bitbucket

- At minimum, stores your codes/papers
- Also, a backup
- You do not need github/bitbucket to use version control

github/bitbucket

- git clone <github url>

github/bitbucket

- git clone <github url>
- git add <list of files>

github/bitbucket

- git clone <github url>
- git add <list of files>
- git commit -m “initial commit”

github/bitbucket

- git clone <github url>
- git add <list of files>
- git commit -m “initial commit”
- git push

github/bitbucket

- git remote add origin <github url>

github/bitbucket

- git remote add origin <github url>
- git pull origin master —allow-unrelated-histories

github/bitbucket

- git remote add origin <github url>
- git pull origin master —allow-unrelated-histories
- git push