

# git workflows for science

Dr. Brian R. Kent

Scientist, National Radio Astronomy Observatory

<https://www.cv.nrao.edu/~bkent/>

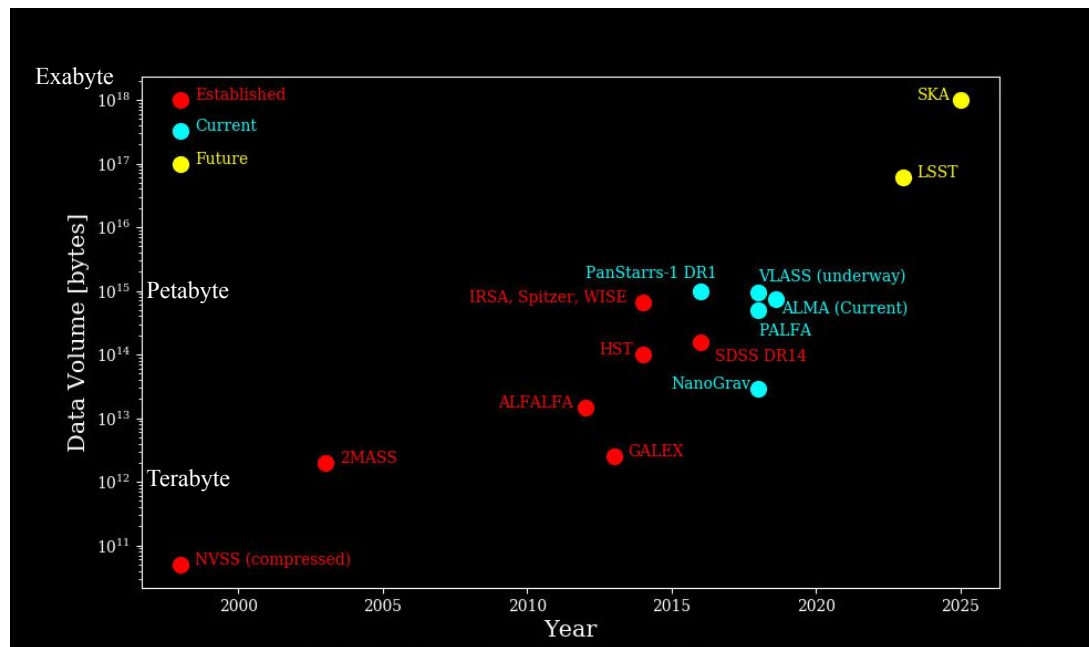
Twitter & Instagram: @VizAstro

# How do we work in science?

We follow the scientific method, ask questions, research the questions, form a hypothesis, test our hypothesis through experimentation and observations, draw conclusions, and report the findings.

Our data acquisition rates continue to climb in astronomy. Call it what you will - algorithm development, software, data science, app development - we have to track the tools that process the data for science.

And astronomers are acquiring **a lot** of data...



Kent 2018

<http://cdsads.u-strasbg.fr/pdf/2019ASPC..523....3K>

# What are we trying to accomplish?

We need to track the prototyping of ideas. These might be:

- A new algorithm...

- A new data reduction scheme...

- A calculation of some sort...

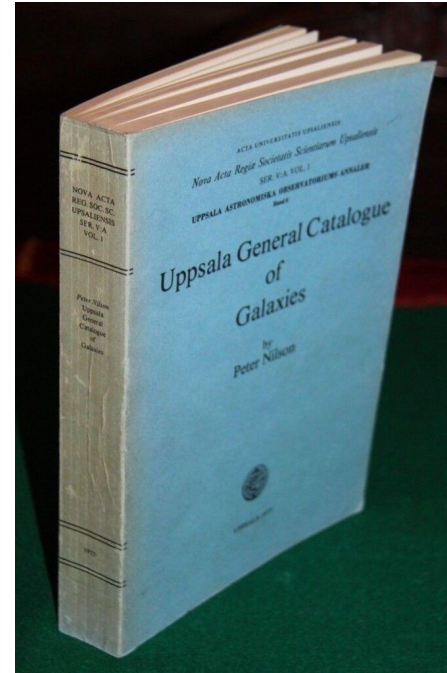
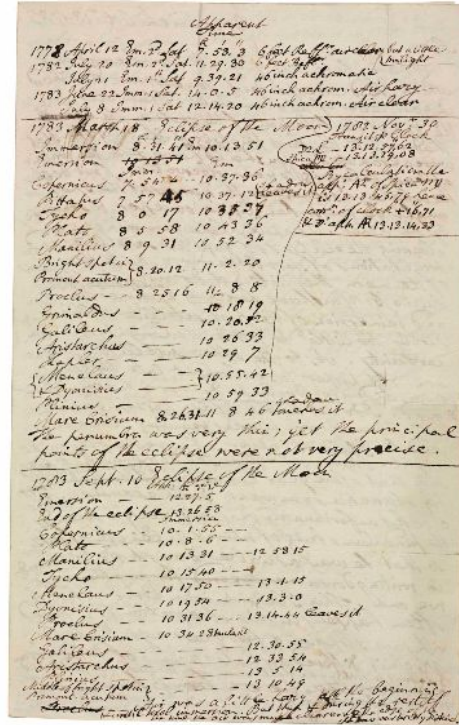
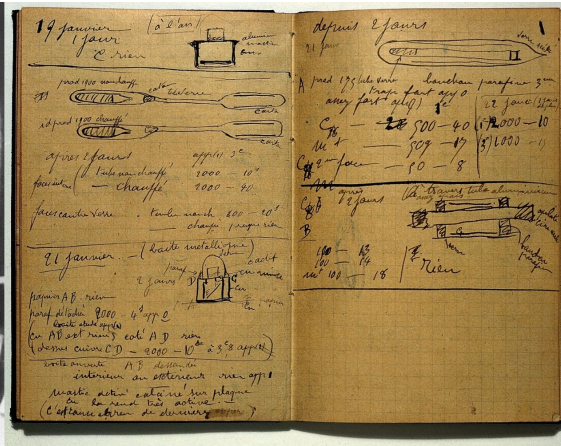
- Prototype software or package...

- Expanding on the work of others...

We need to tabulate our data, metadata, calculations, and results, for *reproducibility*.

We need to share these ideas and data with others, often merging inputs from multiple parties. Conflicts need to be resolved.

# Scientific record keeping



# Version control systems

Allows us to track changes to code. Systems like cvs, svn, and git.

From the book Pro git: <https://git-scm.com/book/en/v2>



*“The major difference between Git and any other VCS (Subversion and friends included) is the way Git thinks about its data. Conceptually, most other systems store information as a list of file-based changes. These other systems (CVS, Subversion, Perforce, Bazaar, and so on) think of the information they store as a set of files and the changes made to each file over time (this is commonly described as delta-based version control).”*

git written by Linus Torvalds:

<https://www.linuxfoundation.org/blog/10-years-of-git-an-interview-with-git-creator-linus-torvalds/>

# git

Try <https://github.com/> or <https://bitbucket.org>

Experiment, create repos, look at the workflows of others

Have a look at git guides if you are new to using it:

<https://guides.github.com/>

# Cloning an existing git repository

**git clone** <https://github.com/rougier/matplotlib-tutorial.git>

```
> git clone https://github.com/rougier/matplotlib-tutorial.git
```

```
Cloning into 'matplotlib-tutorial'...
```

```
remote: Enumerating objects: 479, done.
```

```
remote: Counting objects: 100% (33/33), done.
```

```
remote: Compressing objects: 100% (30/30), done.
```

```
remote: Total 479 (delta 13), reused 5 (delta 3), pack-reused 446
```

```
Receiving objects: 100% (479/479), 3.30 MiB | 0 bytes/s, done.
```

```
Resolving deltas: 100% (148/148), done.
```

**Forking a repository:** <https://docs.github.com/en/get-started/quickstart/fork-a-repo>

# git commands (<https://education.github.com/git-cheat-sheet-education.pdf>)

**git** *(no switches - gives a summary of commands)*

**git checkout <branch name>**

**git branch -a** *list branches and a star will appear by the one currently checked out*

**git log** *what has been done?*

**git status** *any changes?*

**git diff <hash1> <hash2>** *compare the two*



# Integrated Development Environments

Makes coding easier!

<https://www.jetbrains.com/pycharm/>

<https://code.visualstudio.com/>

<https://netbeans.apache.org/>

<https://www.eclipse.org/ide/>



# Cloud resources

<https://research.google.com › colaboratory>

(syncs with github!)

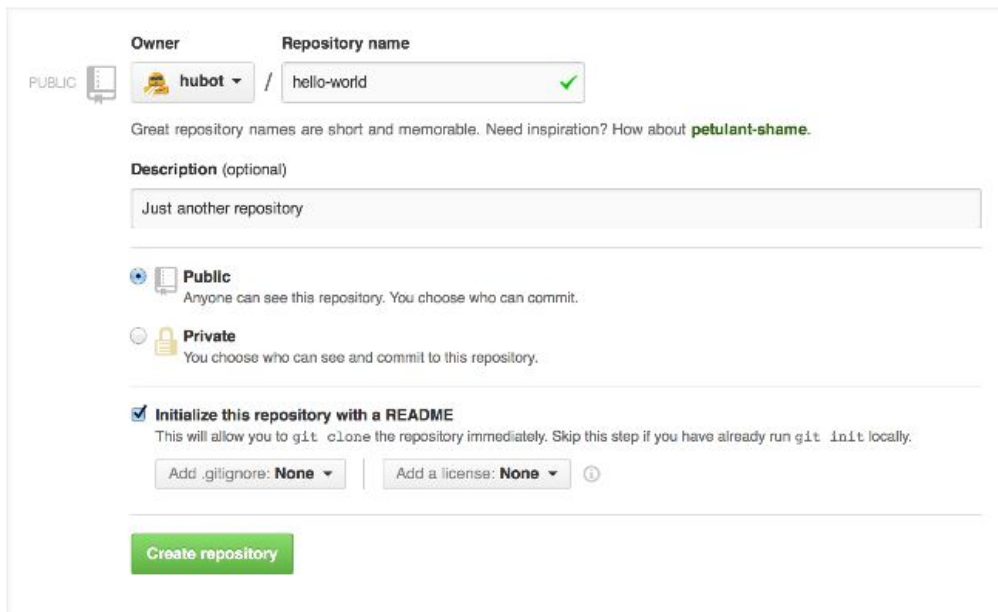
<https://mybinder.org/>

# Starting a repository

Let's hop onto github and make an example...

Start by creating a new repo and adding a **README.md** file

Learn how to edit these files with:  
<https://www.markdownguide.org/>



The screenshot shows the GitHub 'Create repository' form. At the top, there's a 'PUBLIC' label and a computer icon. The 'Owner' is set to 'hubot' with a dropdown arrow. The 'Repository name' is 'hello-world' with a green checkmark. Below this, a message says: 'Great repository names are short and memorable. Need inspiration? How about **petulant-shame**.' The 'Description (optional)' field contains 'Just another repository'. There are two visibility options: 'Public' (selected) with a blue circle icon, and 'Private' with a yellow lock icon. The 'Public' option says 'Anyone can see this repository. You choose who can commit.' The 'Private' option says 'You choose who can see and commit to this repository.' There is a checked checkbox for 'Initialize this repository with a README'. Below this, it says 'This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.' There are two dropdown menus: 'Add .gitignore: None' and 'Add a license: None'. At the bottom is a green 'Create repository' button.

# Creating a branch

In general, we do not want to commit changes to the main branch...

...so let's create a new branch (sometimes referred to as a *feature branch*). Give your branches unique, descriptive names.

...changes will be made to this branch.

...We will test and verify the changes.

...We will commit the changes and comment

# Committing changes

Treat your commits like a scientific logbook. Give details. Others should understand why and where you made changes. Include notes and references as to why these changes were made.



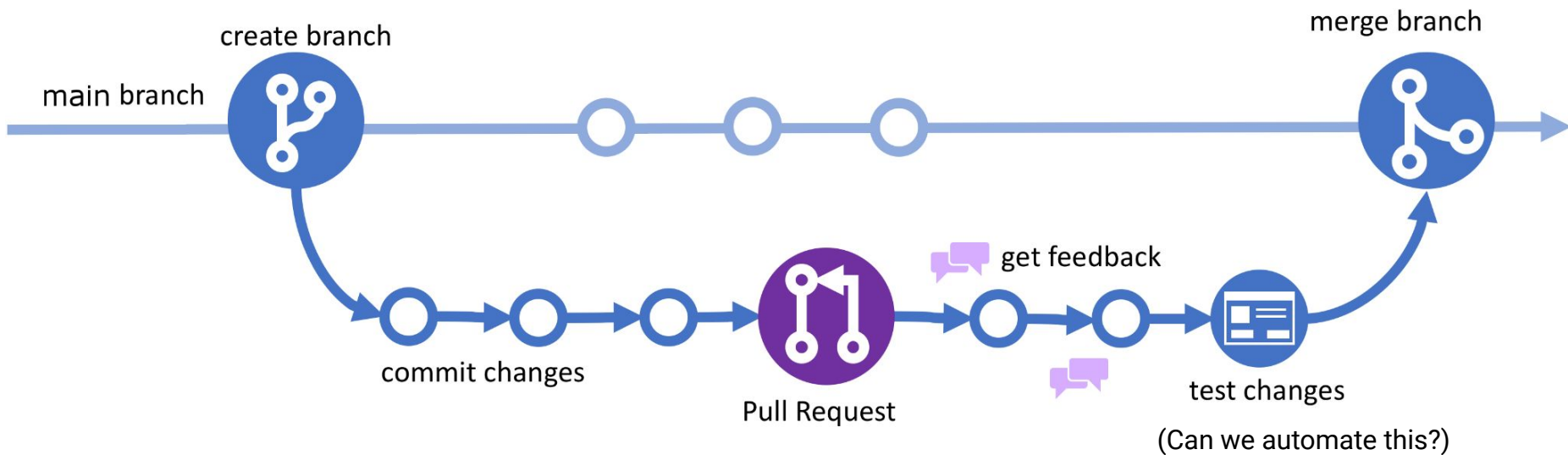
	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT  
MESSAGES GET LESS AND LESS INFORMATIVE.

xkcd...

# Workflow

## GitHub Flow



# Creating automated tests

<https://docs.pytest.org/en/6.2.x/>

GitHub Actions: <https://github.com/features/actions>

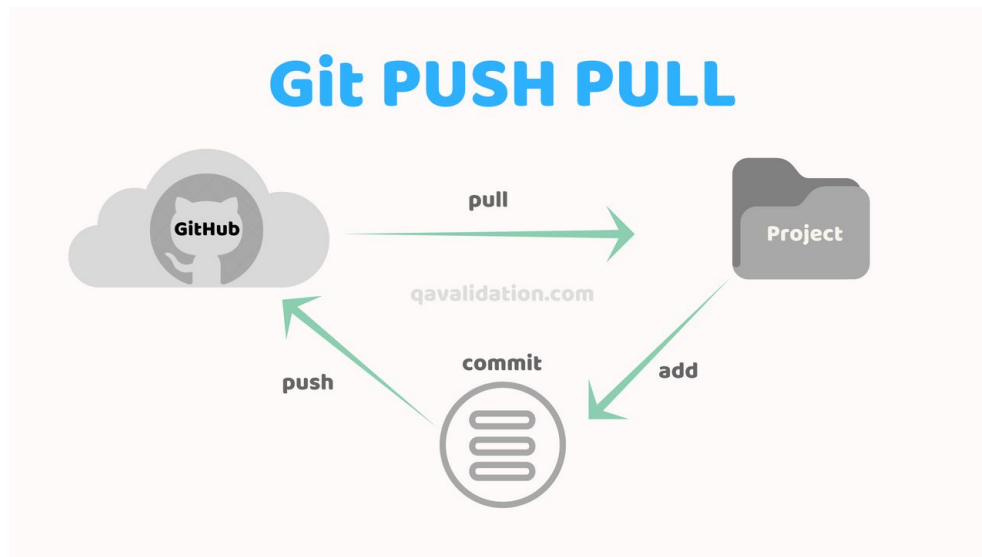
Anatomy of a test:

```
# content of test_example.py
def inc(x):
    return x + 1

def test_answer():
    assert inc(3) == 5
```

# PUSH and PULL!

Because of the nature of git's design, you have a local copy of the entire repository. We need to push the change so that others will see it on the branch.

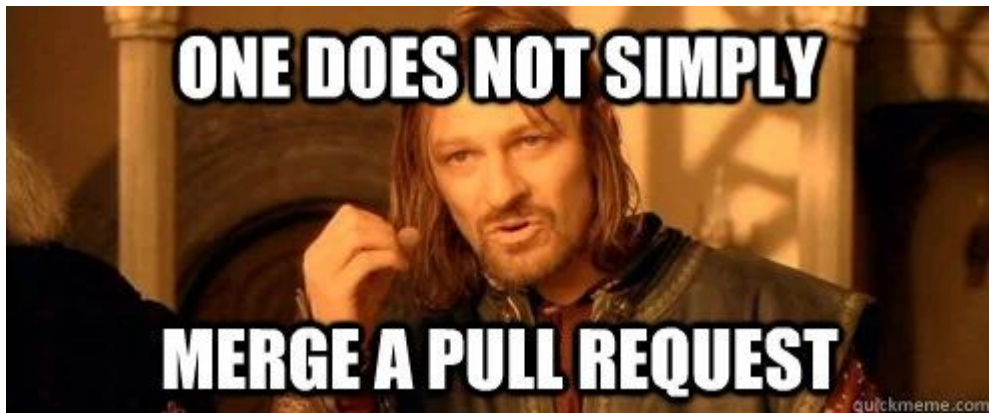




# Pull requests, discussion, update, and merge

This is where we are ready to share with others. We can state what we intend to do, have a documented discussion, point out further refinements, and have automated tests run.

When all is satisfactory, we can merge...



# Sample workflow demonstration

- Define our project
  - Define our requirements
  - Gather our data
  - Create a repository
  - Branch off of main
  - Display a VLASS archive image in Colab
  - Created a pull request (PR), discuss and merge to main
- 
- Create a new branch
  - Create separate updates
  - PR-merge one branch
- 
- Create a simple automated test
  - Define packages and requirements
  - Create a workflow with GitHub Actions
  - Create a new PR, let the workflow run its tests. If it passes, we can merge.

# Resources

Think about how automation and version control can help you in your project workflow. These are *valuable jobs skills to have for the future!*

<https://githubuniverse.com/>

<https://guides.github.com/>

[https://www.youtube.com/results?search\\_query=intro+to+github+](https://www.youtube.com/results?search_query=intro+to+github+)