Managing Projects with Git and GitHub

Tim Jenness / LSST

1st ASTERICS-OBELICS International School

6-9 June 2017, Annecy, France.

H2020-Astronomy ESFRI and Research Infrastructure Cluster
(Grant Agreement number: 653477).

Brief history of revision control

- VAX/VMS file system (per file)
- Directory per release
- RCS/SCCS (per file/single directory)
- CVS (per file/directory tree)
- Subversion (client/server & file rename with whole tree tracking)
- Distributed version control (peer to peer, whole tree)

Git

- Developed for Linux kernel in 2005.
- No central repository. Every clone can pull or push to any clone. Laptop to desktop to laptop.
- Full history in each clone.
- Commits each get a SHA1 to describe the change and the commit itself (including parents).
- Tags are associated with a particular commit.
- Tracks content not file names

Git crash course

- `git clone https://github.com/org/repo`
- `git commit –a`
- `git add changed_file`
- `git push`
- `git pull`
- `git log`
- `git status`

Committing

- Uses a staging area called the “index”
- Commits are always local.
- Commits can be edited and rearranged *until they are pushed and relied upon by others.*
  - `git commit --amend`
- Can stage subset of changes via `git add –p`
- `git commit --author` if you are committing for someone else.

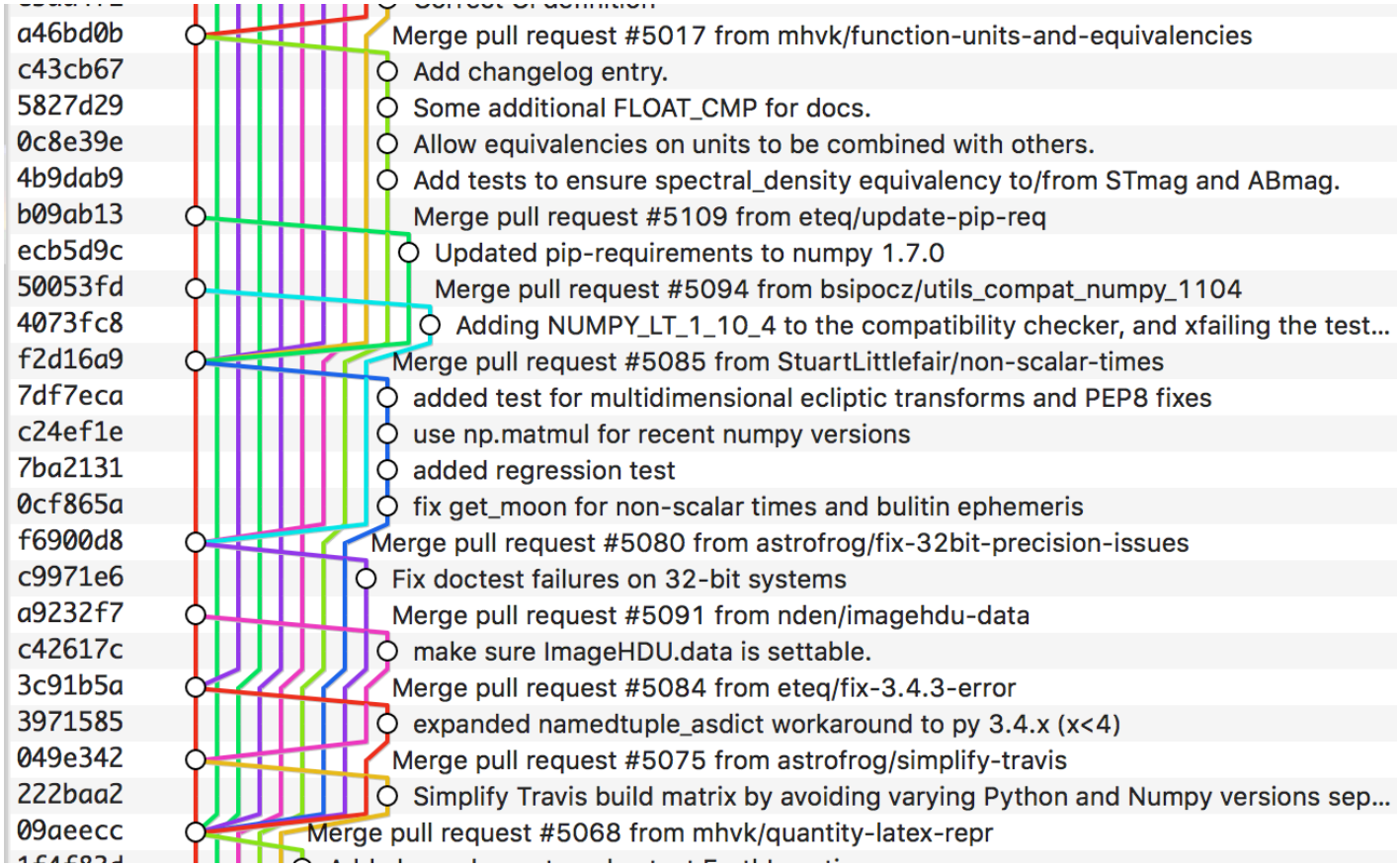
Branches

- The default branch is called master
- Small projects can get away with doing everything on master.
- But, branches can help to group related changes and are necessary when multiple features are being developed in parallel.
- Work on a feature, merge it to master when done.
- Branches are lightweight, and are nothing to be scared of.

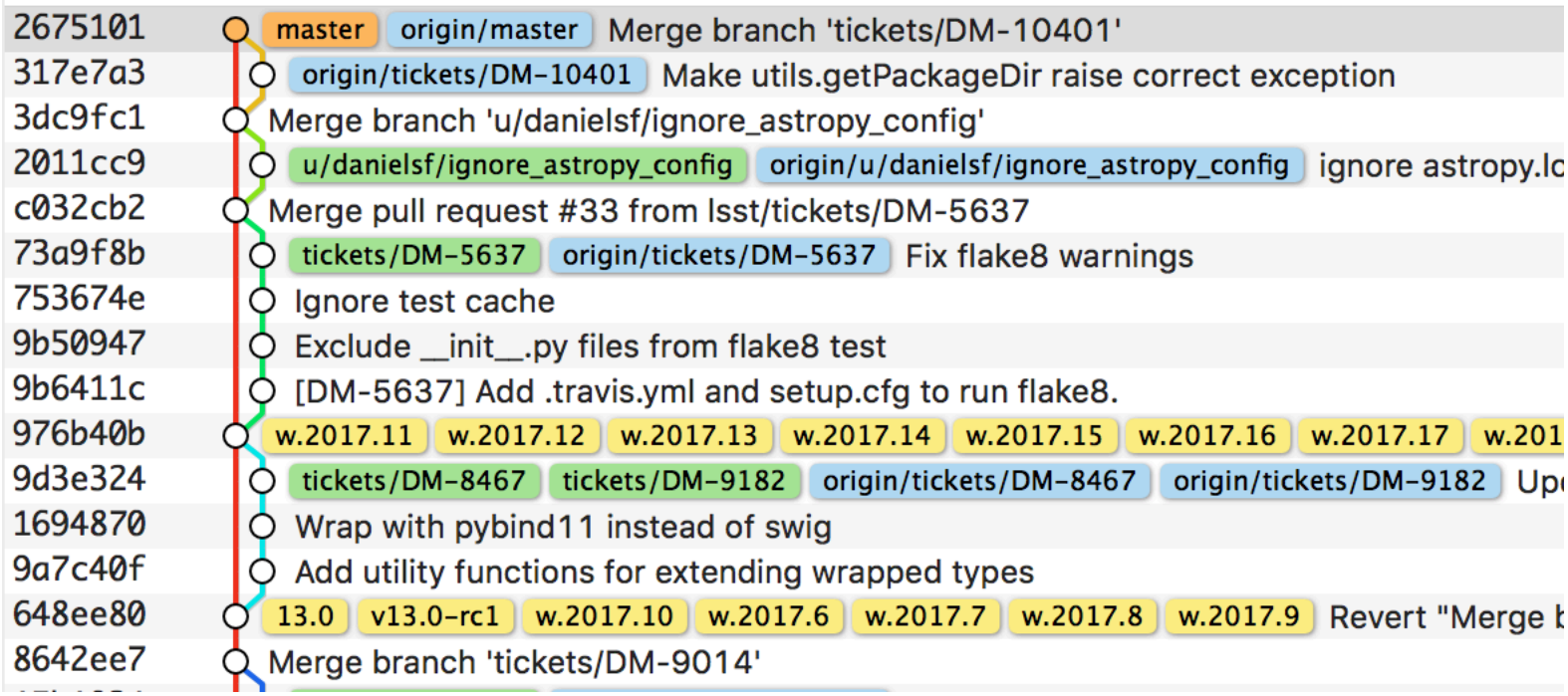
"Real" history vs "logical" history

- How can history be correct but unclear when working out why something was done that way?
- Consider how you want your project to manage branches.
- Is it more important to know when a branch was created with a merge commit "fixing things up", or is it better to have a self-contained branch with no change in the merge commit?

Consider


a46bd0b Merge pull request #5017 from mhvk/function-units-and-equivalencies
c43cb67 Add changelog entry.
5827d29 Some additional FLOAT_CMP for docs.
0c8e39e Allow equivalencies on units to be combined with others.
4b9dab9 Add tests to ensure spectral_density equivalency to/from STmag and ABmag.
b09ab13 Merge pull request #5109 from eteq/update-pip-req
ecb5d9c Updated pip-requirements to numpy 1.7.0
50053fd Merge pull request #5094 from bsipocz/utils_compat_numpy_1104
4073fc8 Adding NUMPY_LT_1_10_4 to the compatibility checker, and xfailing the test...
f2d16a9 Merge pull request #5085 from StuartLittlefair/non-scalar-times
7df7eca added test for multidimensional ecliptic transforms and PEP8 fixes
c24ef1e use np.matmul for recent numpy versions
7ba2131 added regression test
0cf865a fix get_moon for non-scalar times and bulitin ephemeris
f6900d8 Merge pull request #5080 from astrofrog/fix-32bit-precision-issues
c9971e6 Fix doctest failures on 32-bit systems
a9232f7 Merge pull request #5091 from nden/imagehdu-data
c42617c make sure ImageHDU.data is settable.
3c91b5a Merge pull request #5084 from eteq/fix-3.4.3-error
3971585 expanded namedtuple_asdict workaround to py 3.4.x (x<4)
049e342 Merge pull request #5075 from astrofrog/simplify-travis
222baa2 Simplify Travis build matrix by avoiding varying Python and Numpy versions sep...
09aeecc Merge pull request #5068 from mhvk/quantity-latex-repr

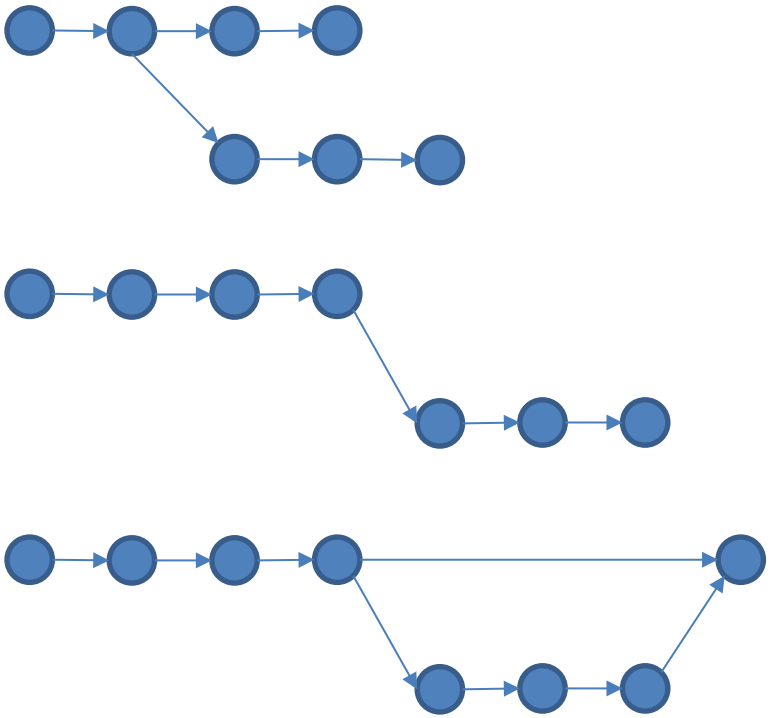
Versus


2675101 master origin/master Merge branch 'tickets/DM-10401'
317e7a3 origin/tickets/DM-10401 Make utils.getPackageDir raise correct exception
3dc9fc1 Merge branch 'u/danielsf/ignore_astropy_config'
2011cc9 u/danielsf/ignore_astropy_config origin/u/danielsf/ignore_astropy_config ignore astropy.lo
c032cb2 Merge pull request #33 from lsst/tickets/DM-5637
73a9f8b tickets/DM-5637 origin/tickets/DM-5637 Fix flake8 warnings
753674e Ignore test cache
9b50947 Exclude __init__.py files from flake8 test
9b6411c [DM-5637] Add .travis.yml and setup.cfg to run flake8.
976b40b w.2017.11 w.2017.12 w.2017.13 w.2017.14 w.2017.15 w.2017.16 w.2017.17 w.201
9d3e324 tickets/DM-8467 tickets/DM-9182 origin/tickets/DM-8467 origin/tickets/DM-9182 Up
1694870 Wrap with pybind11 instead of swig
9a7c40f Add utility functions for extending wrapped types
648ee80 13.0 v13.0-rc1 w.2017.10 w.2017.6 w.2017.7 w.2017.8 w.2017.9 Revert "Merge b
8642ee7 Merge branch 'tickets/DM-9014'

What is rebasing?

- “rebasing” refers to moving the base commit of a branch and then reapplying the commits relative to a different place.
- Usually you rebase to the head of the branch you started from.
- This changes the SHA1 of the commit (the parent commit changes).

Rebasing



We changed history?

- Git treats the shared history of a project with great care.
- Distributed version control can only work if we all agree on that history.
- You can modify local (unpushed) history whenever you want.
- However, never modify a commit that other people rely on.
- Feature branches can be transient, if your project admins are using rebase.
- If you are sharing a feature branch, you may need to agree who is doing the rebasing.
- You will have to “force push” your branch to the server.

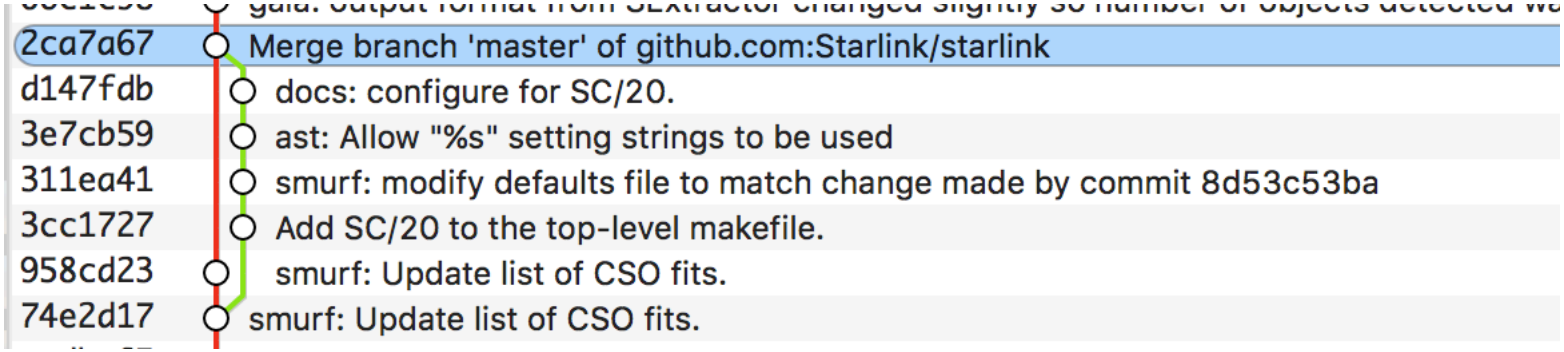
Rebasing vs Merging

- Both approaches are valid.
- Depends on what your priorities are.
- Merging without rebasing:
  - Simple to describe (keep on committing, merge when done)
  - History is exactly what happened
  - But history can be complicated as branches are merged while older branches are developed.
  - The code you test on the branch probably won't be the code you end up with after the merge.

Rebasing

- It's not usually important to know *when* someone started work on a branch or what state the repository was in.
- "Clean" history simplifies bisecting to find when a bug appeared.
- Rebasing allows you to test exactly what the merged codebase will look like without doing the merge.
- If a branch is around a long time before merging, rebasing "as you go" can make eventual merging significantly less complicated and risky.
- Using `--no-ff` when merging is a great way to indicate in the history what the relevant feature commits were.

Where did this merge come from?



- These come from two people sharing a branch and both people committing.
- You commit on your local branch, someone else pushes some changes to GitHub.
- When you pull how can git resolve this when it can't rewrite history?
- Create a merge commit with local commits on one side and the remote commits on the other.
- In almost every case we do not want these merge commits.

How do we fix it?

- Use `git pull --rebase`
- This will stash your local commits, pull in the remote commits, then add your commits to the tip of the current branch.
- If you notice after you've pulled. Do Not Panic.
- Use `git reset --hard HEAD^`
- And then run the rebase pull.

Interactive rebase

- With `–i` option git will show you all the commits on your branch and let you edit, squash, and reorder.
- Extremely powerful way to “clean up” your history.
- Allows you to remove all those “fix typo” commits!
- Consider crafting your commits to allow the future you to see a coherent narrative for how the feature was implemented.
- Future you does not care about the blind alleys you went down and the mistakes you made.

Interactive rebase

- `git rebase –i master`

```
pick 1df7567 Add derive requirements
pick f283332 Add labels to each ID and link to them internally
pick 9e515ca Fix unit in calProcTime
pick 3cb906f Fix most parameters
pick c4094b3 Add back transSNR

# Rebase 9970550..c4094b3 onto 9970550 (13 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
```

Commits

- Commits are going to be read by future you
- Commits might be reverted if you change your mind later on. That's impossible if you combined a bug fix in the same commit with your new feature.
- Do not combine different concepts in a single commit.
- Whitespace changes should always be in a different commit to substantive changes.
  - Unless you want to annoy your reviewer with 50 lines of removing trailing whitespace in the commit and one line of fix you want them to accept.

Commit messages

- Follow the git standard.
- One short line with a description of what the commit is doing.
- Then a blank line.
- Then, if warranted, some detail about the change. Preferably with some background that would not be appropriate for inline comments.
- Tools rely on that format when summarizing.

Cherry Picking

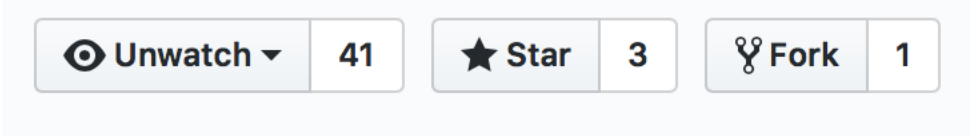
- Sometimes you want one commit from a branch rather than all of them.
- Use `git cherry-pick`
- Be careful not to end up with lots of duplicate commits because of merging branches where lots of commits have been cherry picked.

GitHub

- Place for collaboration started in 2008.
- Git is distributed but GitHub is centralized.
- Has concept of a Fork then Pull Request to allow people to submit changes without having push access.

Fork & Pull

- Fork a repository then clone to your development system.

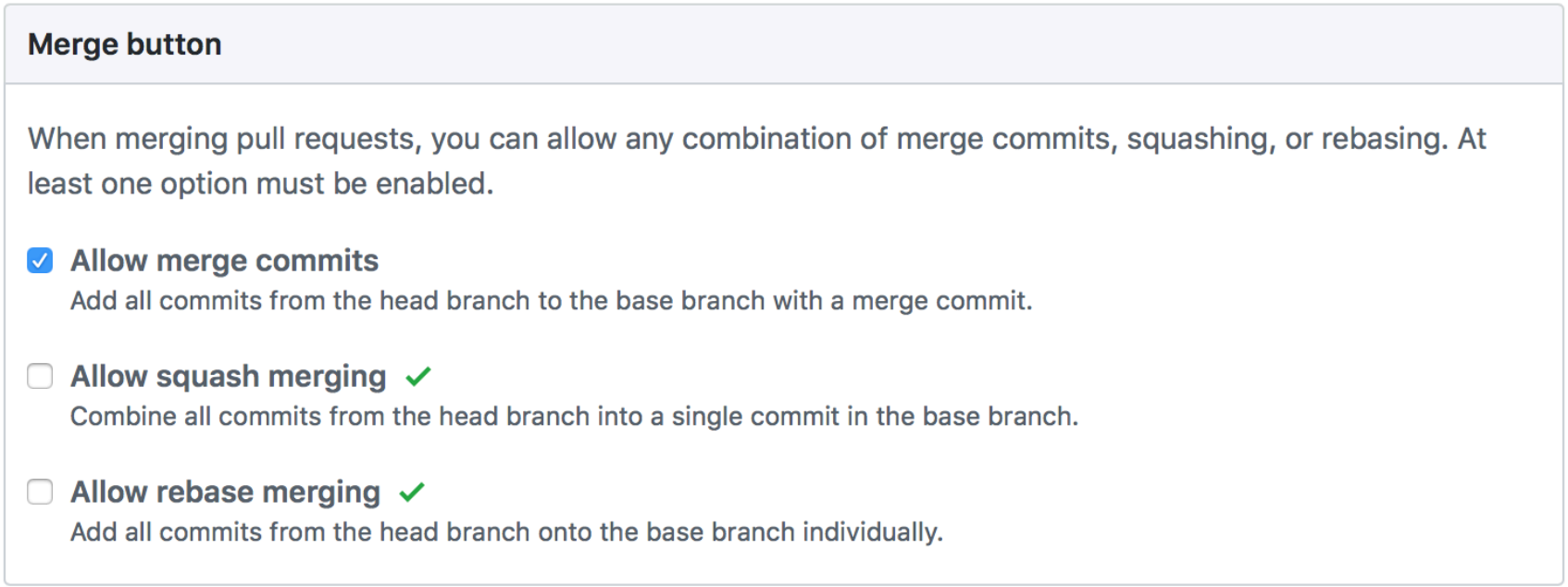


- If you plan to track upstream development always work on a branch, never master.
- Keep master up to date using `git remote`.
- Push branch, make pull request.
- Delete your branch after merging.

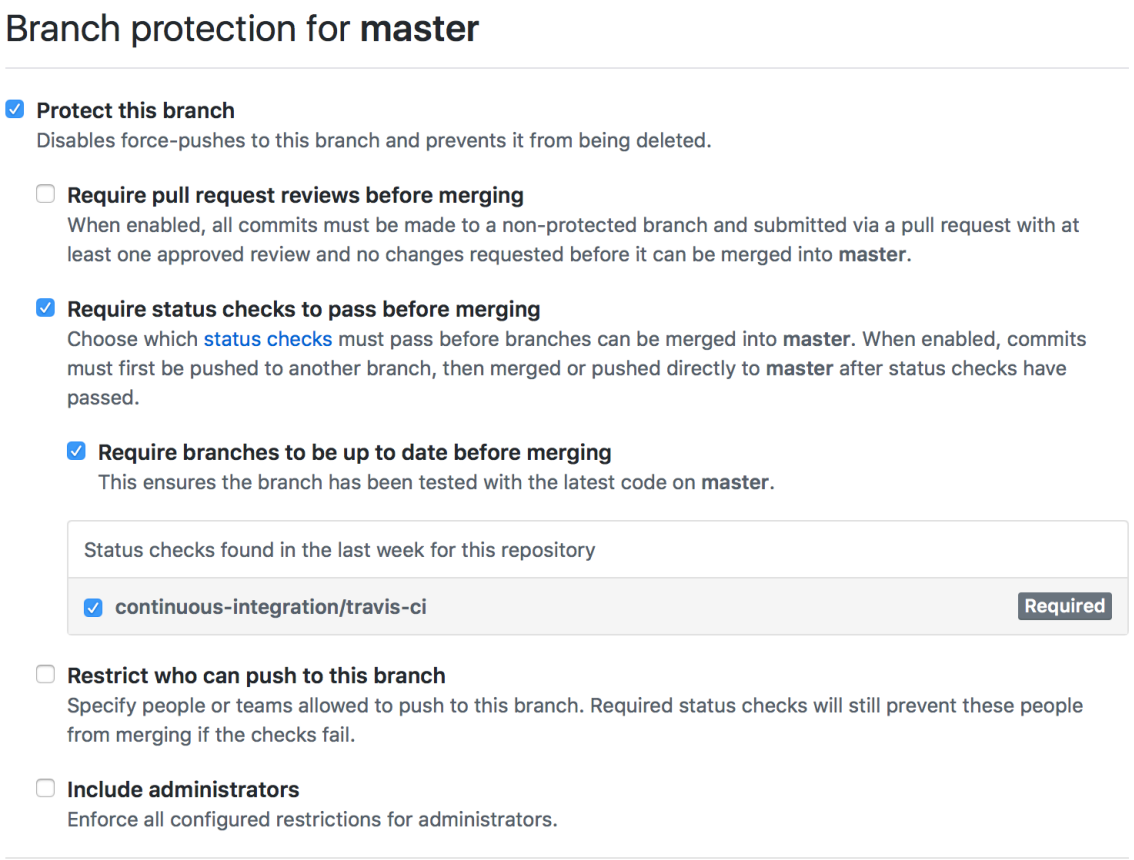
Code Review

- Important for code maintainability when multiple people are involved.
- Do not review code style, that's what linters are for.
  - For Python, connect flake8 to Travis so that the code is checked for every pull request.
- Look for logic errors, bad assumptions, missed checks.
- Were tests added for the new code?
- Were tests added for bug fixes that failed with the old version?

Repository Settings: Merge button

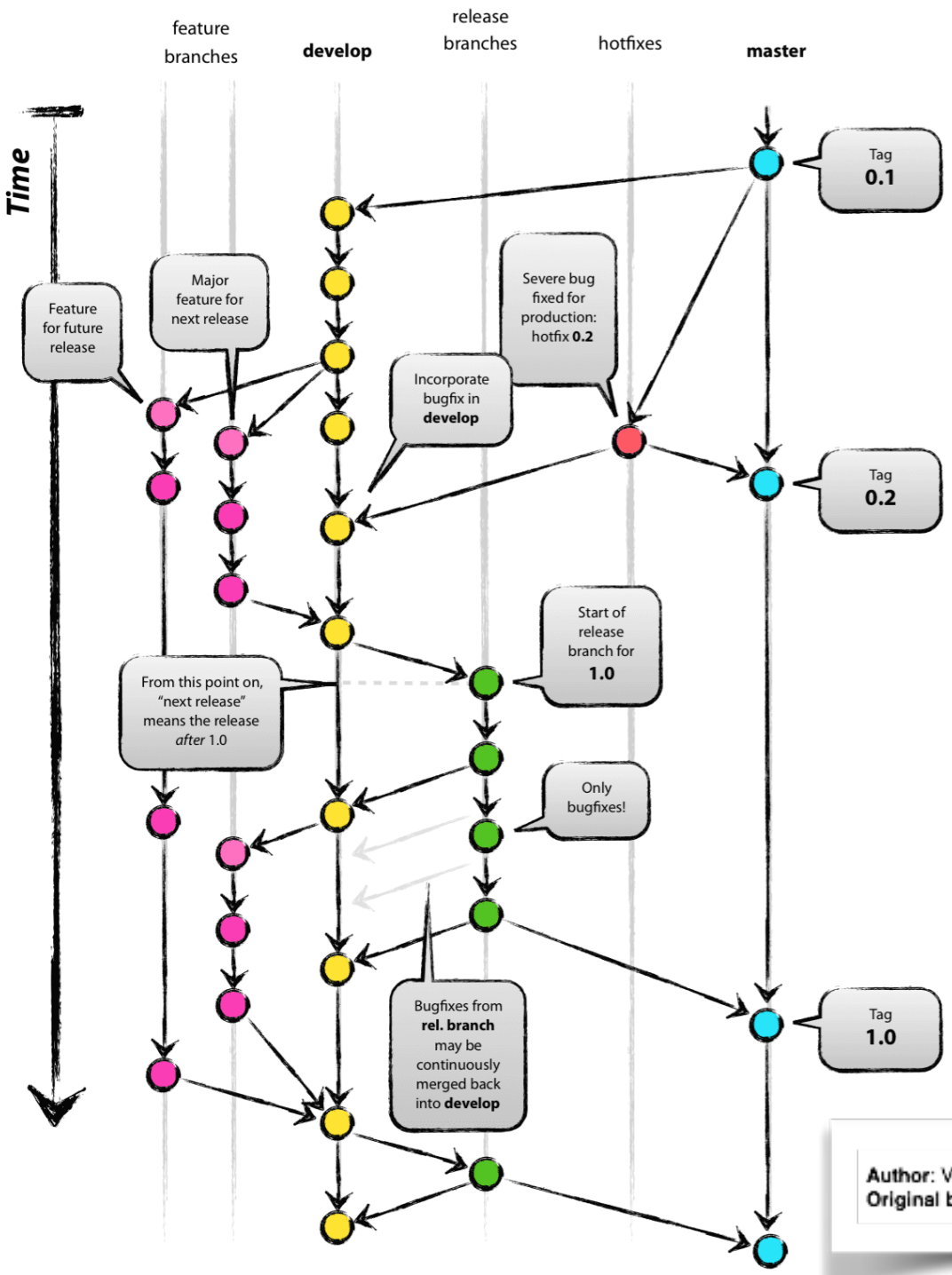


Branch Protection



Release Strategies

- Flexibility to do anything
- Develop on master, make release branches
- Develop on feature branches, release on master
- Trickle up pull requests (Linux kernel)
- Branches corresponding to each stage of development



Author: Vincent Driessen
Original blog post: http://nvie.com/archives/

Handy tips

- Add this to your `~/.gitconfig`:

```
[alias]
        lg = log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s
%Cgreen(%cr) %C(bold blue)%Creset' --abbrev-commit
```

```
[thrylos:lsst-texmf (master) $ git lg | head -12                                    ]
*   9d5630b - (HEAD -> master, origin/master) Merge pull request #54 from lsst/u/timj/table-color (10
 days ago)
|\
| * a0cefd7 - (origin/u/timj/table-color, u/timj/table-color) Enable table option in xcolor package (
10 days ago)
|/
*   709588d - Merge pull request #53 from lsst/u/timj/for-lse-61 (10 days ago)
|\
| * f9322a4 - (origin/u/timj/for-lse-61, u/timj/for-lse-61) Fix sort order (10 days ago)
| * fc876dc - Add LSE-39, -68 and -69 (10 days ago)
|/
*   29fbc78 - Merge pull request #51 from lsst/u/timj/for-ldm-482 (13 days ago)
|\
| * 1491683 - (origin/u/timj/for-ldm-482, u/timj/for-ldm-482) Add LPM-162, DMTN-035, Document-15097,
and LDM-482 (13 days ago)
```

What's the current branch?

Add this to your `~/.profile`:

```
source ~/etc/git-prompt.sh
export PS1='\[\e[1;32m\]\h\[\e[0;39m\]:\[\e[1;34m\]\W\[\e[1;31m\]$(__git_ps1)\[\e[0;1m\]
\$ \[\e[0;39m\]'
```

Helps a lot when you are about to force push on the wrong branch.

```
[thrylos:lsst-texmf (master) $ git checkout tickets/DM-9941
Switched to branch 'tickets/DM-9941'
thrylos:lsst-texmf (tickets/DM-9941) $
```

Further Reading

- LSST Developer Guide: https://developer.lsst.io
- Pro Git Book: https://git-scm.com/book/en/v2

Acknowledgement

- H2020-Astronomy ESFRI and Research Infrastructure Cluster (Grant Agreement number: 653477).