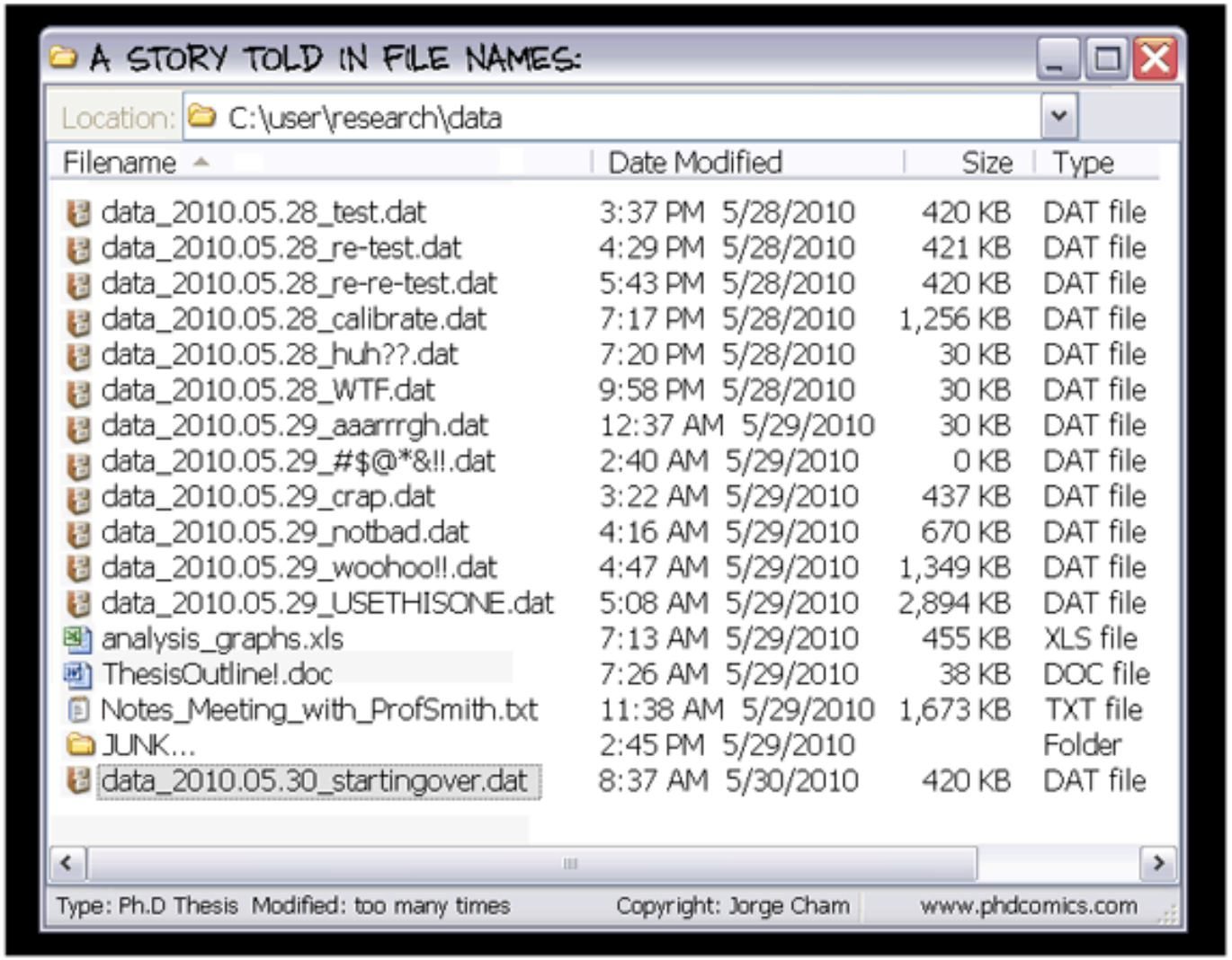
Introducing Version control Collaborative development Continuous integration

Rene Gassmoeller,
UC Davis

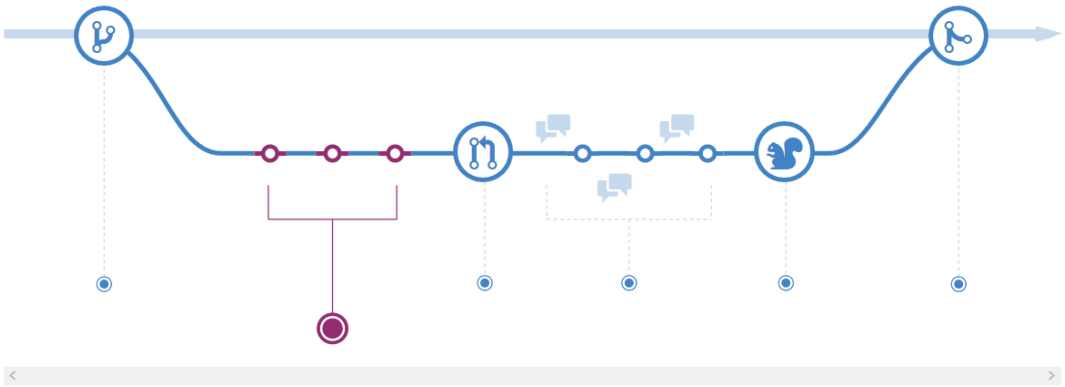
CIG COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

Version control – A story told in file names

- Archive your history: What was the state at last AGU?
- Archive your decisions: Why did you change this?
- Archive connections: What else was affected by this change?
- Safely test inside the same directory



Version control – Collaborative development



Github Flow, https://guides.github.com/introduction/flow/

- Allows parallel, safe, reproducible development of code

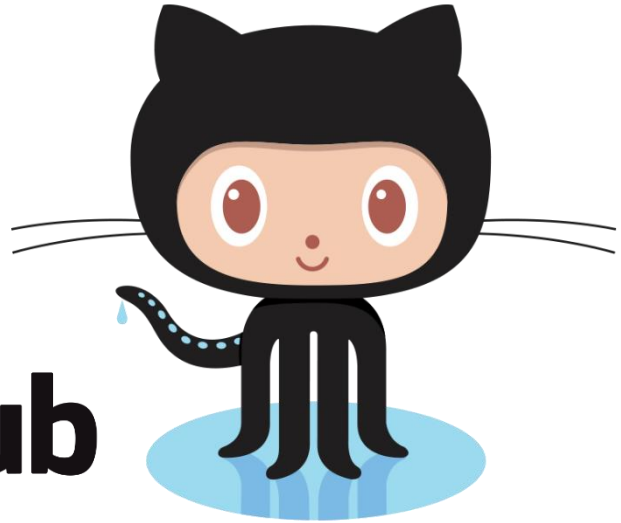
- Create branch
- Make commits
- Create pull request
- Wait for review
- Address comments
- Pull request is merged

Version control – Setting up Git

- Install git
- Set name and email address:
  - git config --global user.name “[name]”
  - git config --global user.email “[email]”
- Set favourite text editor:
  - git config --global core.editor "vim“

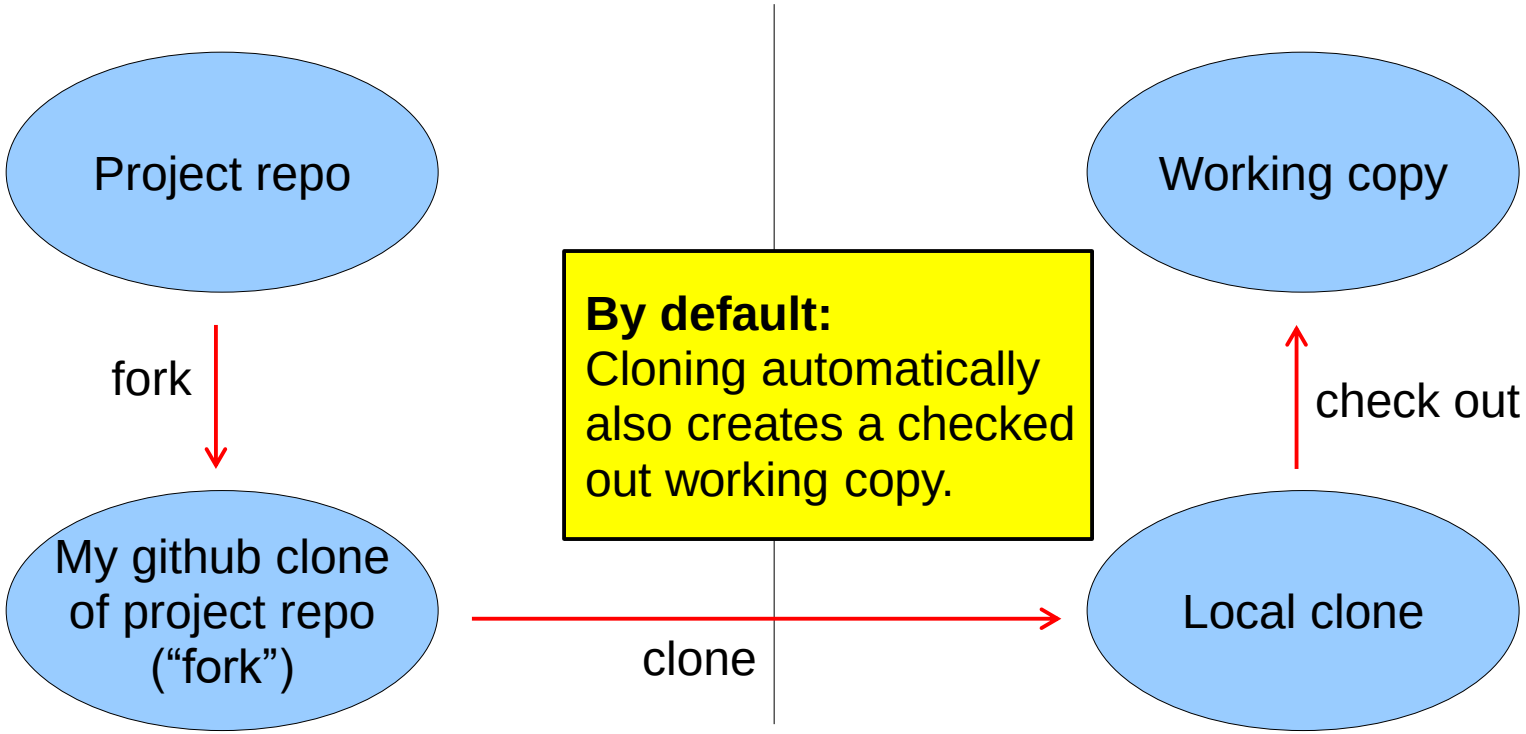
Version control – Setting up Github

- Create a github account
- Browse to the geodynamics/Rayleigh repository
- Click the “Fork”button (top right)
- Create a ssh-key on your laptop (if you do not have one already), e.g. by running `ssh-keygen` in your terminal
- Link the public SSH key ($HOME/.ssh/id_rsa.pub) with your github account under https://github.com/settings/keys by copying the content of the key to the webpage



Version control – Creating a local copy

- Github.com:
- Local computer:

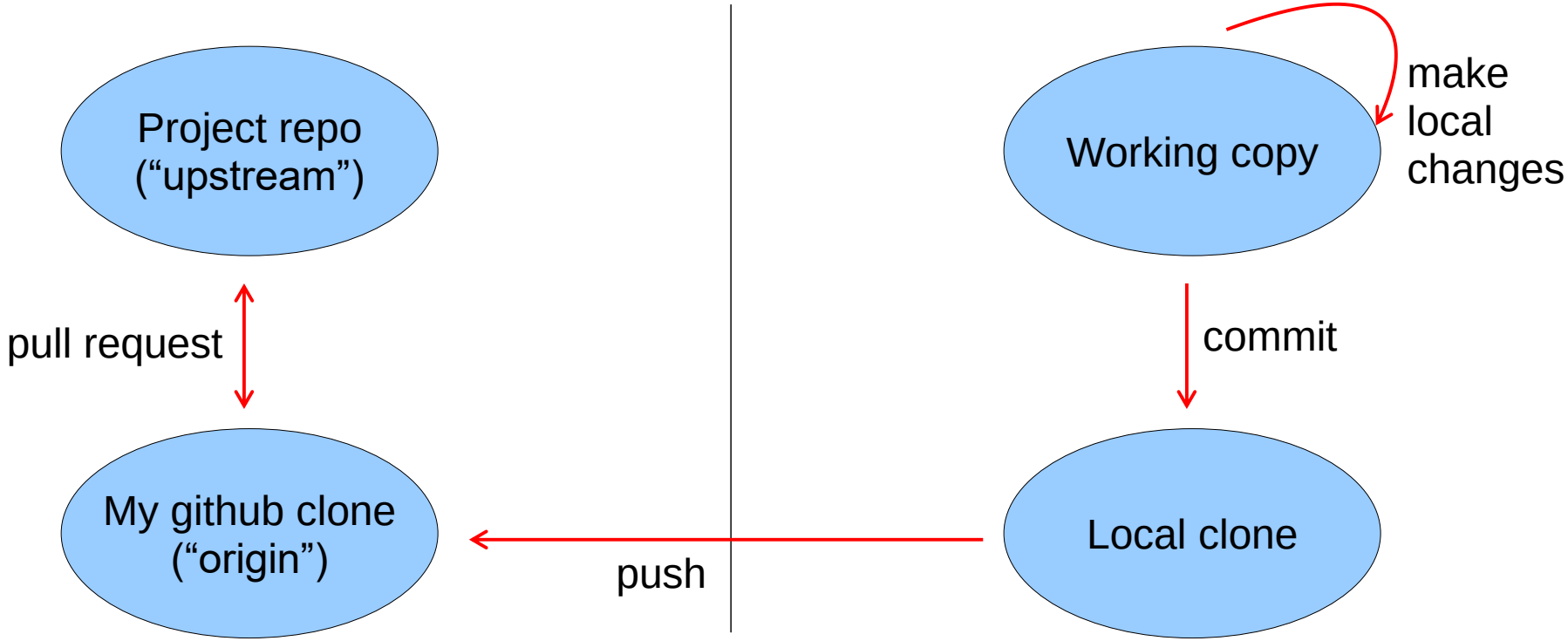


Version control – Creating a local copy

1. Workflow in on Github:
   - Browse to the geodynamics/Rayleigh repository
   - Click the “Fork”button (top right)
2. In a terminal of the local computer:
   - git clone \
     git@github.com:username/Rayleigh.git
   - git remote add upstream \
     https://github.com/geodynamics/Rayleigh.git
   - We have created 2 remotes, “origin” (username/Rayleigh), and “upstream” (geodynamics/Rayleigh)

Version control – Making a pull request

- Github.com:
- Local computer:



Version control – Making a pull request

In practice:

- Because there is a delay between
  – creating a pull request
  – getting it accepted
  it is useful to do almost all development on branches

- Workflow then:
  – create small “feature branch”
  – do development, commit changes
  – “push” the branch to origin
  – create a “pull request” for the changes between
    . the base of the branch
    . the tip of the branch

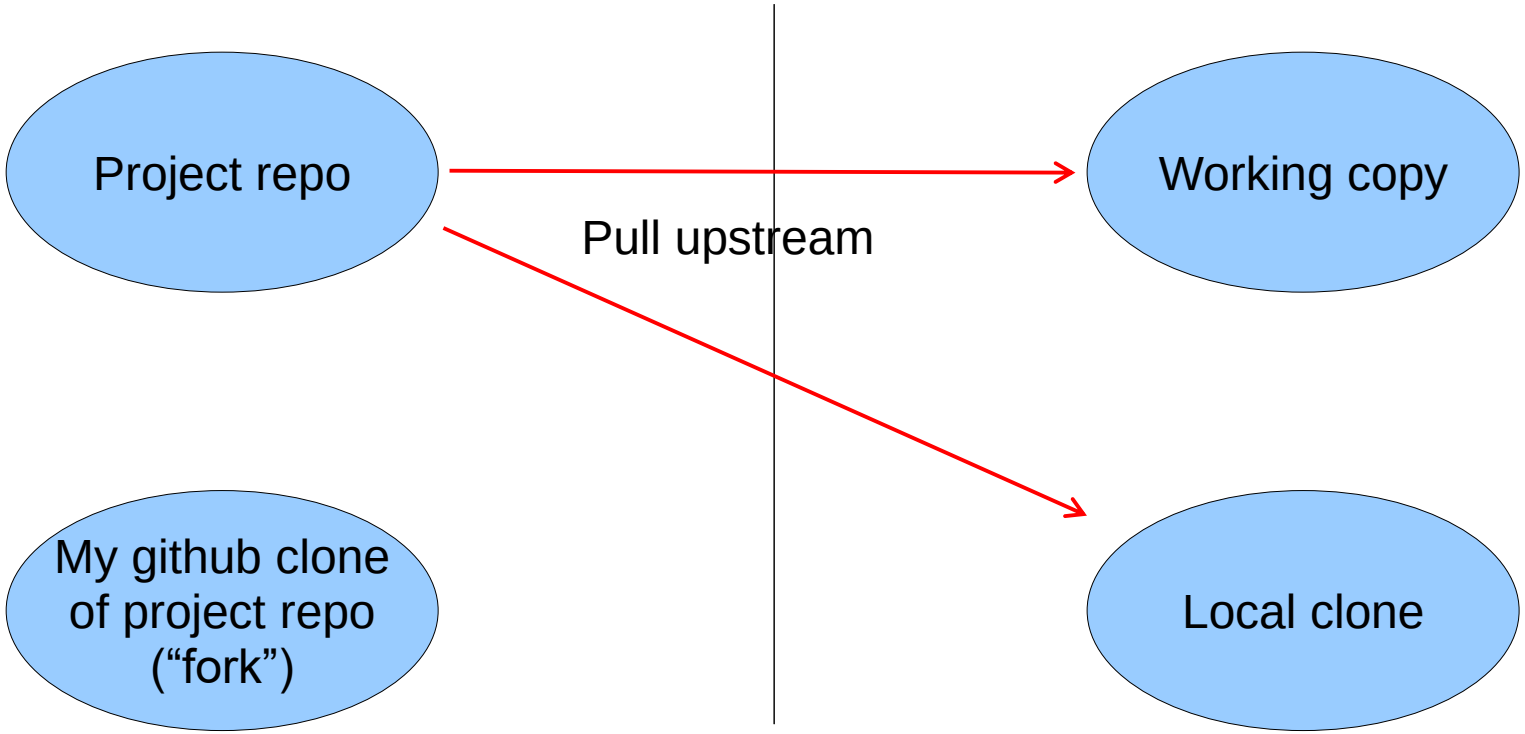
Version control – Making a pull request

- Workflow on Github:
  - If you just pushed: An option to create a pull request will appear
  - If later:
    - Go to the page of your fork
    - Click “Create pull request”
    - Select your branch add_feature_X
  - Describe your changes and open pull request
  - Wait for review and address comments by repeating the local steps

- Local computer:
  - git checkout –b add_feature_X
  - Implement feature X
  - Test feature X
  - git add filename
  - git commit –m ‘Add feature X’
  - git push origin add_feature_X

Version control – Updating from official repo

- Github.com:
- Local computer:



Version control – Updating from official repo

- Into your current branch:
  - `git branch`
  - `git pull upstream master`
- Into the local master branch:
  - `git checkout master`
  - `git pull upstream master`
  - `git push origin master`

Version control – Helpful git commands

- git help <command>: Get help on any git command
- git log: See history of changes
- git branch: See current branch and available branches
- git checkout <branchname>: Switch to different branch
- git add <filename>: Add file to the next (future commit)
- git commit: Create a new commit (=snapshot of the current state)
- git pull <remote> <branch>: Download changes from other repository
- git push <remote> <branch>: Upload changes to another repository
- https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf

Continuous integration

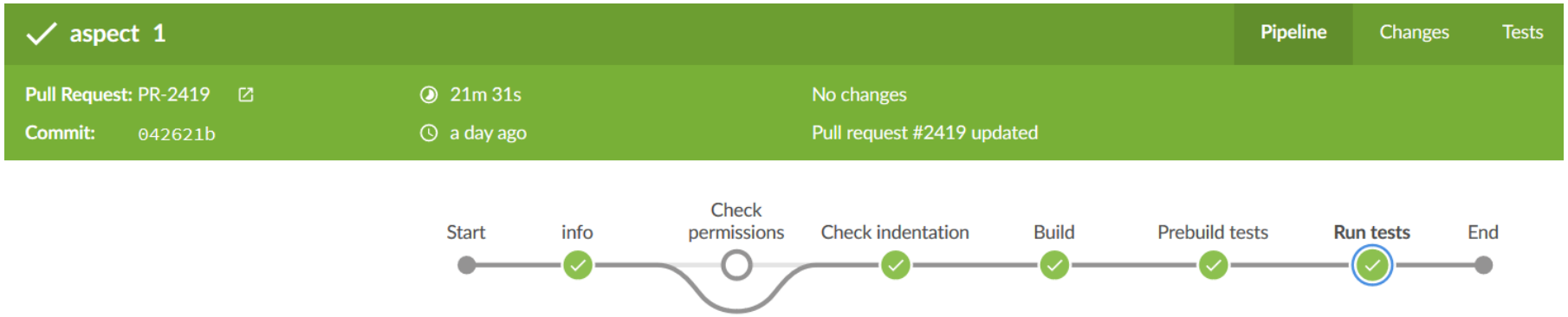
What does it mean?

- No separation in development and stable version
- Releases are just snapshots of the development version with a name
- Make development version stable by continuous (automatic) testing, and peer code review

What does it do?

- Generate trust in your software
- Find/Fix bugs early, fast, and cheap
- Lower barrier for big changes
- Save embarassment during review
- Example from ASPECT:

```
1236      100% tests passed, 0 tests failed out of 614
```



Collaborative development – Code review

- The auto-tester only checks existing tests, can not find all bugs
- Human code review significantly reduces bugs, and improves code quality (efficiently)
- Goal: To review all code before it is merged into Rayleigh (even from maintainers)
- Do this via Githubs review functionality: Hands-on public review
- Keep Code review friendly and constructive:
  - Only request necessary improvements (not nice to have features)
  - Consider level of contributor, but be strict about fundamentals
  - Teach code structure and guidelines