An Overview of GitHub for Data Analysis Projects

Emerging Technologies Information Sessions 2025 Webinar Series
Modeling and Analysis Session
January 8, 2025

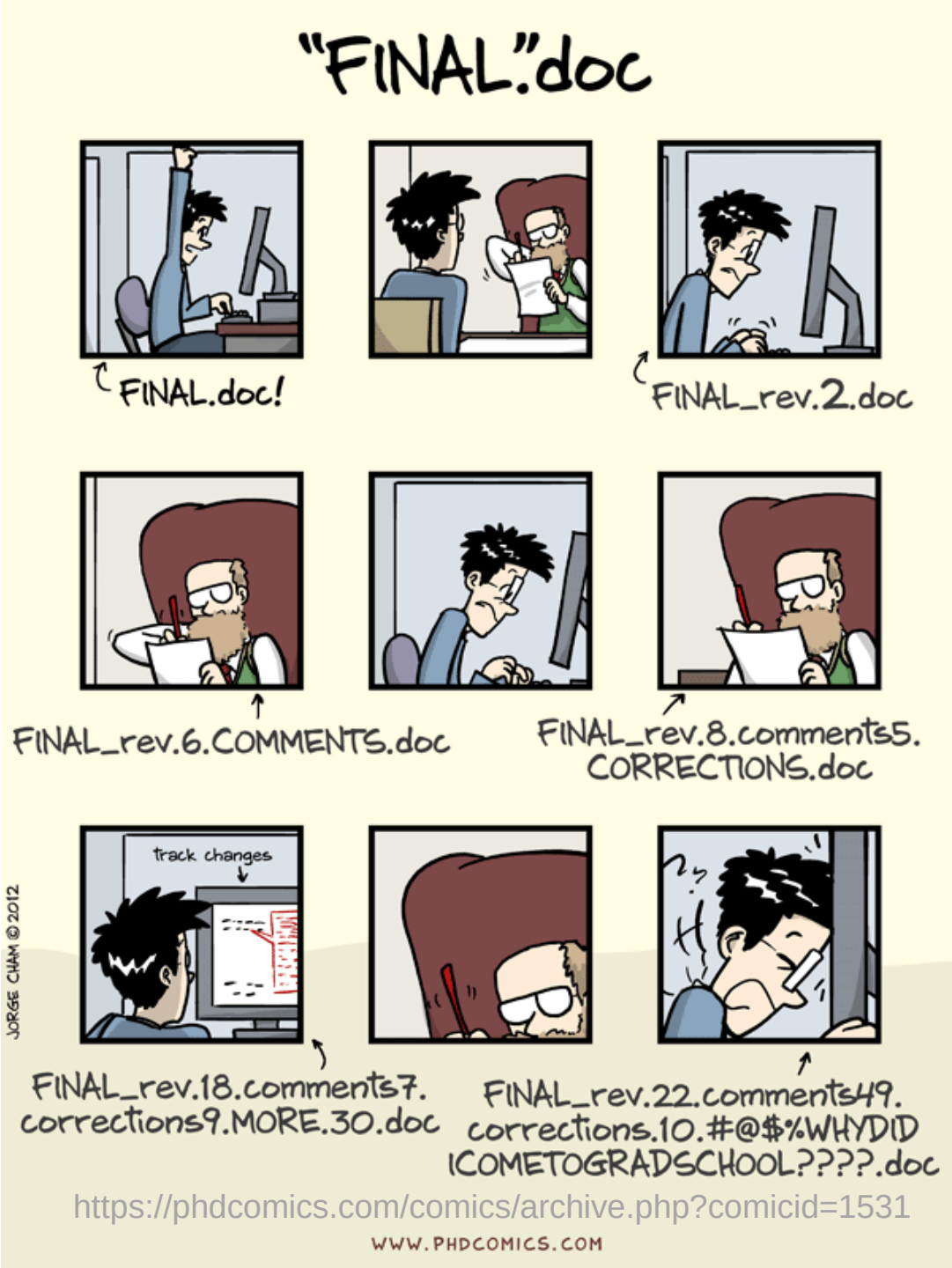
Ben Staton
Quantitative Fisheries Scientist
Columbia River Inter-Tribal Fish Commission
bstaton1
bstaton@critfc.org

Disclaimer: This overview is much more "what and why" of GitHub
See upcoming PNAMP FMWG Webinar Series for "How-To"

Why Version Control?

Especially for Data Analysis Projects?

Avoid this ➡

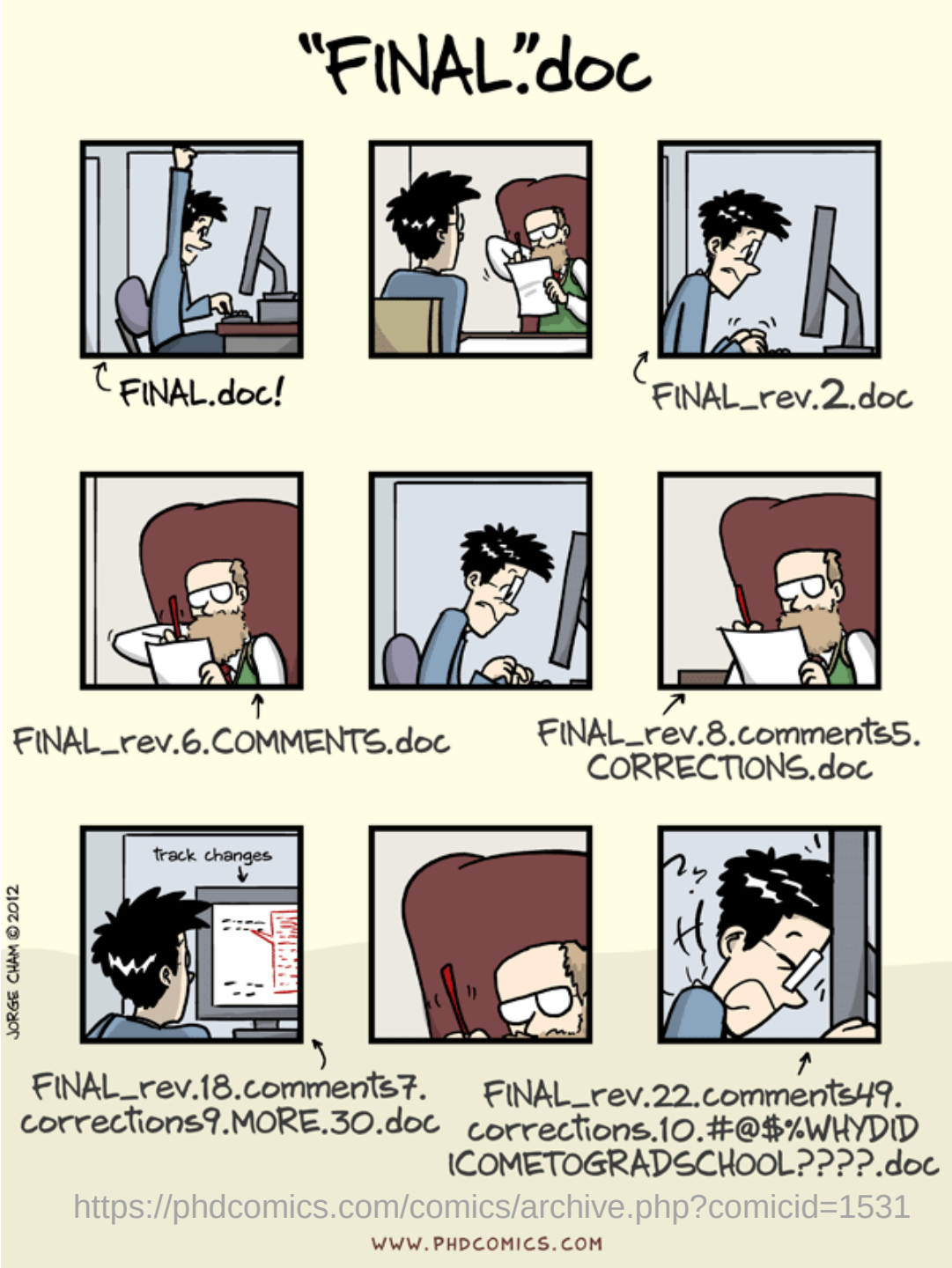


Why Version Control?

Especially for Data Analysis Projects?

Avoid this ➡

- Data analyses have more files
  - Code & data files
  - Projects are directories



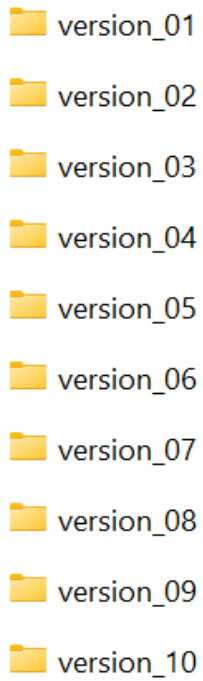
https://phdcomics.com/comics/archive.php?comicid=1531

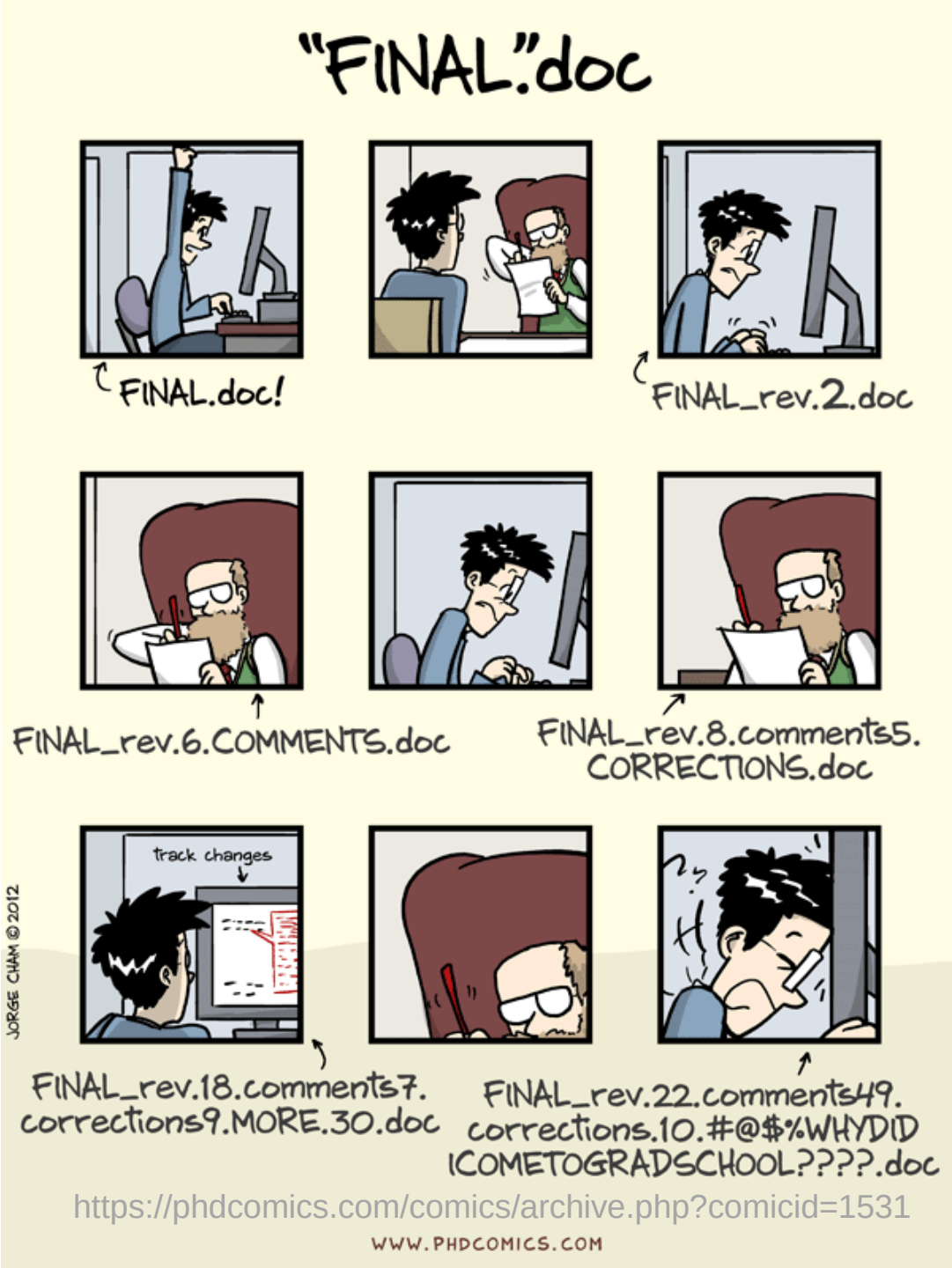
Why Version Control?

Especially for Data Analysis Projects?

Avoid this ➡

- Data analyses have more files
  - Code & data files
  - Projects are directories ➡





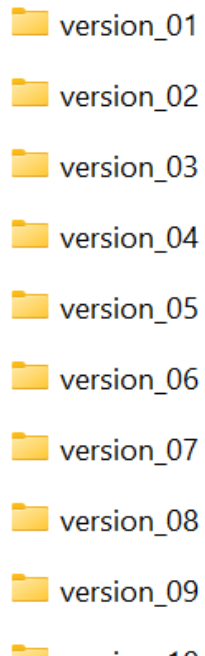
https://phdcomics.com/comics/archive.php?comicid=1531

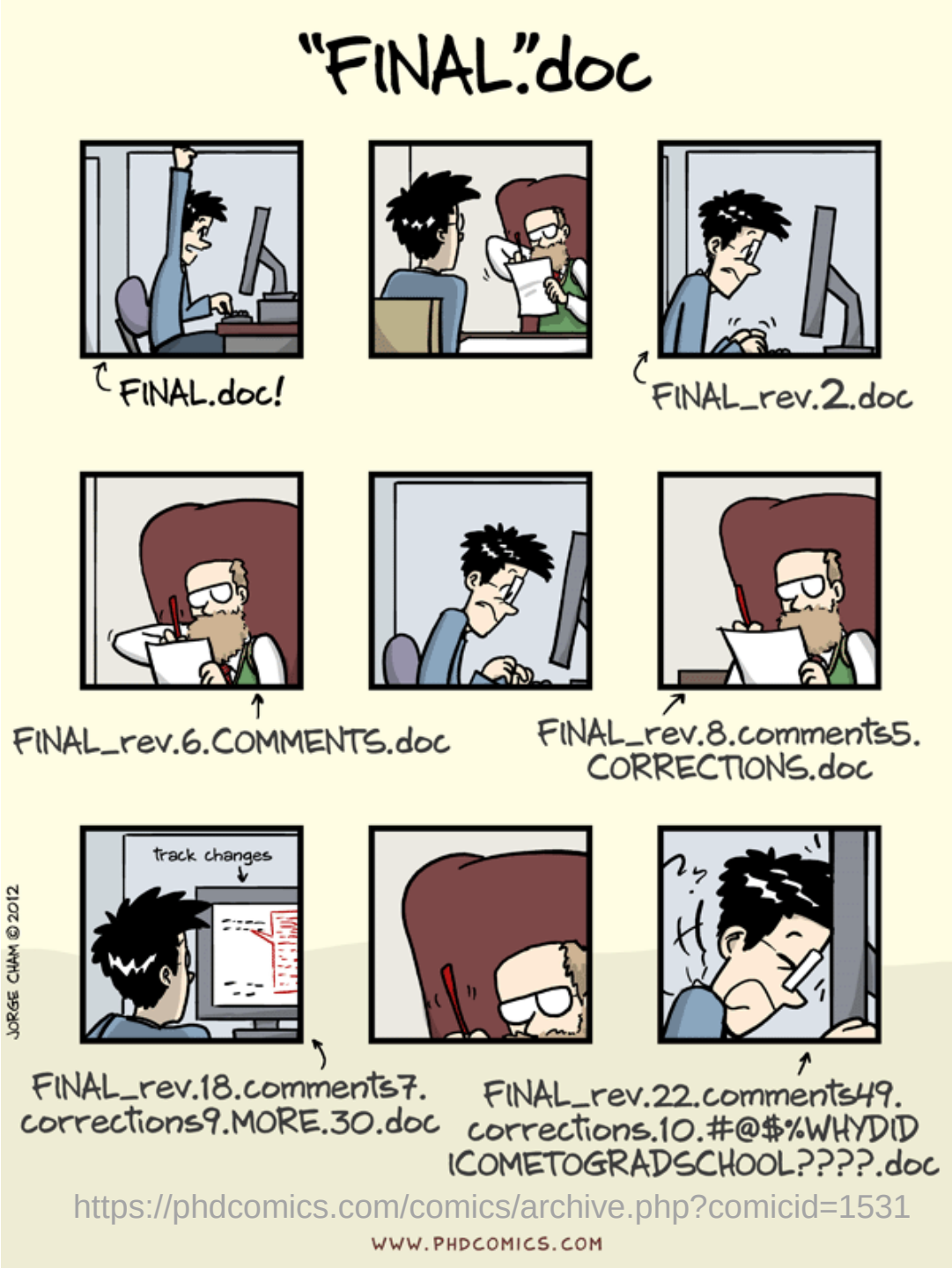
Why Version Control?

Especially for Data Analysis Projects?

Avoid this ➡

- Data analyses have more files
  - Code & data files
  - Projects are directories ➡
- What if you need to (*easily*):
  - Try out something new?
  - See the history of changes?
  - Compare differences in versions?
  - Roll back to an earlier version?





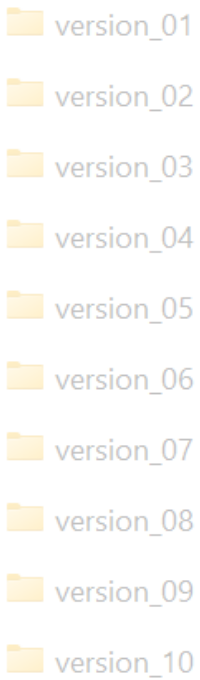
https://phdcomics.com/comics/archive.php?comicid=1531

Why Version Control?

Especially for Data Analysis Projects?

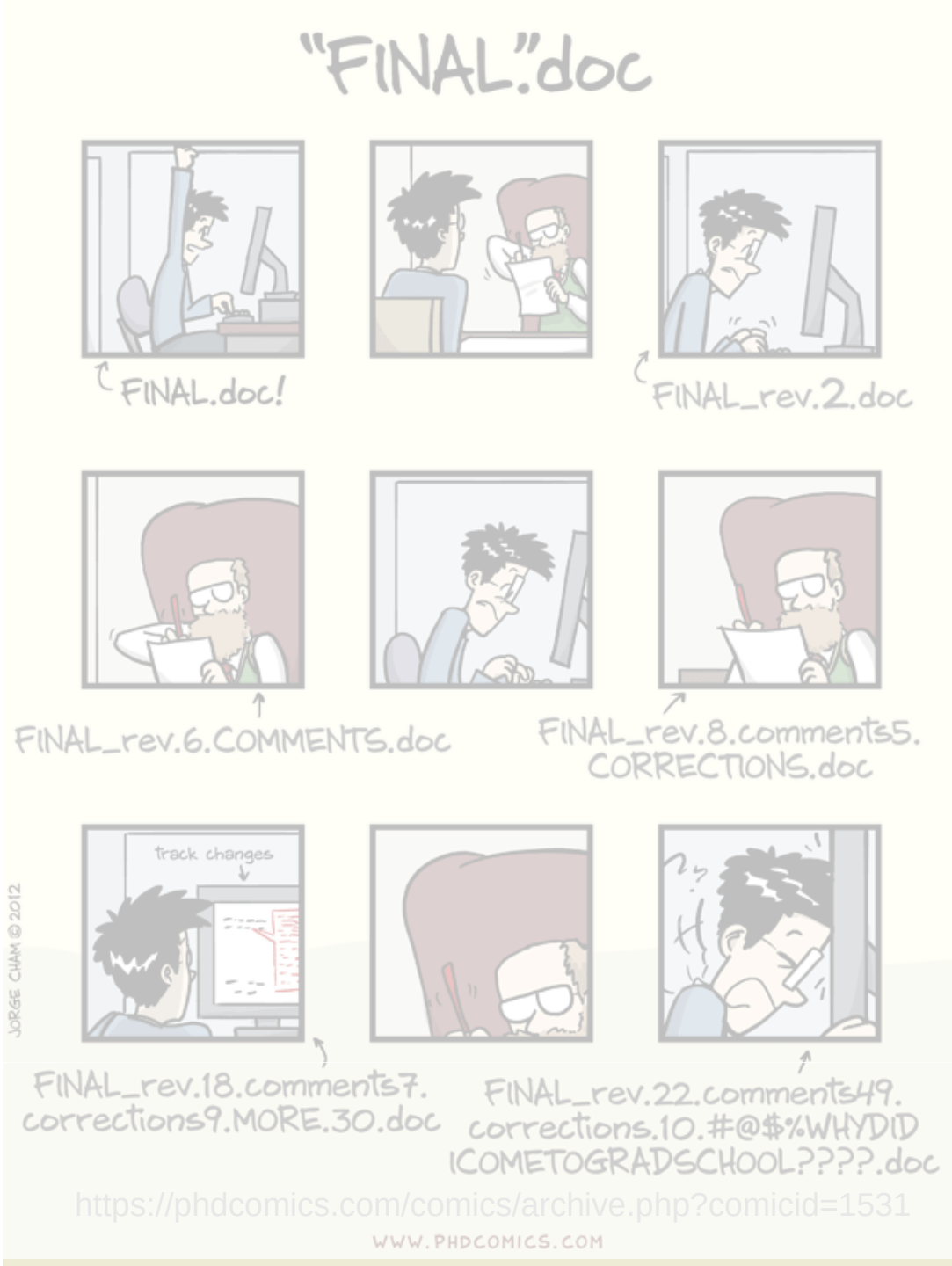
Avoid this ➡

- Data analyses have more files
  - Code & data files
  - Projects are directories ➡



- What if you need to (*easily*):
  - Try out something new?
  - See the history of changes?
  - Compare differences in versions?
  - Roll back to an earlier version?

<u>This</u> is what version control systems (VCS) *are for* (and more!)



Why GitHub?

As the Version Control Platform?

- Old school git (Unix-like CLI) is hard
  - But this is where the VCS tools live
- GitHub offers:
  - All the VCS benefits of Git
  - Desktop GUI to interact with Git (*point/click*)
  - Online hosting (*centralized, most current version*)
  - Collaborative development
  - Project management
  - Project documentation
  - Free & pro versions (*no need for pro generally*)

VCS software "under the hood"

User-friendly, GUIs, collaboration, remote hosting

Other options: e.g., GitLab

Why GitHub?

As the Version Control Platform?

- Old school git (Unix-like CLI) is hard
  - But this is where the VCS tools live
- GitHub offers:
  - All the VCS benefits of Git
  - Desktop GUI to interact with Git (*point/click*)
  - Online hosting (*centralized, most current version*)
  - Collaborative development
  - Project management
  - Project documentation
  - Free & pro versions (*no need for pro generally*)

VCS software "under the hood"

User-friendly, GUIs, collaboration, remote hosting

Tired of sending code/data back and forth, each time with different file names and describing what you changed?

Other options: e.g., GitLab

History of Changes

Who Made the Change, When, and What was the Purpose?

Commits on May 11, 2023

Merge pull request #195 from bstaton1/constant-prespawn-surv
bstaton1 authored on May 11, 2023

Remove time-varying prespawn mort from sim-vs-obs
bstaton1 committed on May 11, 2023

Remove time-varying prespawn mort from all output plots
bstaton1 committed on May 11, 2023

Update initial value generator
bstaton1 committed on May 11, 2023

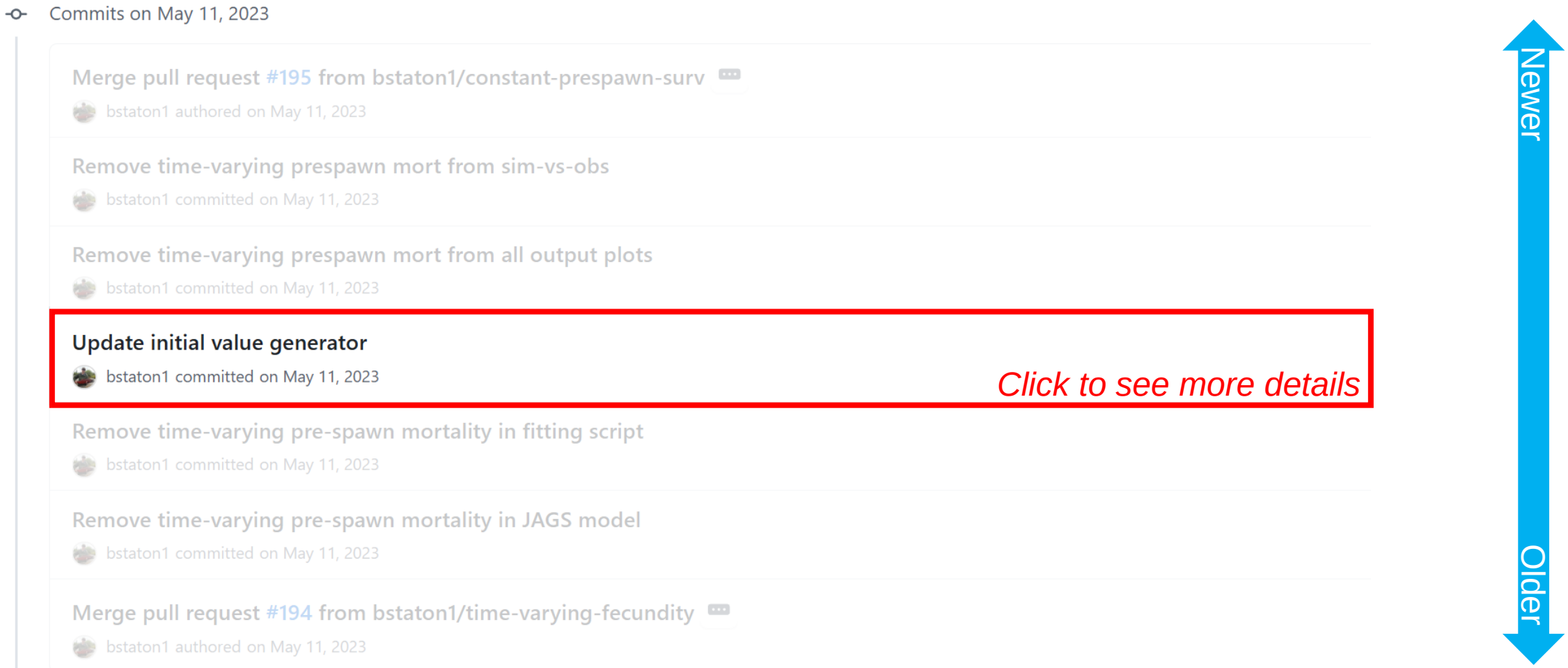
Remove time-varying pre-spawn mortality in fitting script
bstaton1 committed on May 11, 2023

Remove time-varying pre-spawn mortality in JAGS model
bstaton1 committed on May 11, 2023

Merge pull request #194 from bstaton1/time-varying-fecundity
bstaton1 authored on May 11, 2023

History of Changes

Who Made the Change, When, and What was the Purpose?



File Diffs

Easily View File Changes

Original Version

```
77  -     prespawn_surv = x_carcass_spawned/x_carcass_total
78  -     prespawn_surv[is.na(prespawn_surv)] = 0.75
79  -     prespawn_surv[prespawn_surv < 0.5] = 0.5
80
81        # get the age/origin compositon of the total adult return
82        # assume 60% pHOS and 20%, 60%, 20% of age 3, 4, and 5
    @@ -96,7 +94,7 @@ get_E_obs = function(jags_data, fill_missing = FALSE, append_sim_yrs = FALSE) {
96
97        # remove prespawn morts
98        age_origin_survived = lapply(1:nj, function(j) {
99  -       apply(age_origin_above_weir[,,j], 2, function(a) a *
    prespawn_surv[,j])
100       })
101       age_origin_survived = do.call(abind, append(age_origin_survived,
    list(along = 3)))
```

Changed Version

```
77  +     prespawn_surv = colSums(x_carcass_spawned, na.rm =
      TRUE)/colSums(x_carcass_total, na.rm = TRUE)
78
79        # get the age/origin compositon of the total adult return
80        # assume 60% pHOS and 20%, 60%, 20% of age 3, 4, and 5
94
95        # remove prespawn morts
96        age_origin_survived = lapply(1:nj, function(j) {
97  +       apply(age_origin_above_weir[,,j], 2, function(a) a *
      prespawn_surv[j])
98        })
99        age_origin_survived = do.call(abind, append(age_origin_survived,
      list(along = 3)))
```

What do you need to git started?

Beyond Patience…

- GitHub account (*free; info*)
- GitHub Desktop installed (*info*)
- A beginner's knowledge of markdown syntax (*info*)
- Working knowledge of terminology…

Term	Description
1. Repository	A directory (folder) with files under version control *(aka “repo”)*
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	
10.	
11.	
12.	
13.	

Term	Description
1. Repository	A directory (folder) with files under version control *(aka “repo”)*
2. Local (Repository)	A repository found on your hard drive
3.	
4.	
5.	
6.	
7.	
8.	
9.	
10.	
11.	
12.	
13.	

Term	Description
1. Repository	A directory (folder) with files under version control *(aka “repo”)*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4.	
5.	
6.	
7.	
8.	
9.	
10.	
11.	
12.	
13.	

Term	Description
1. Repository	A directory (folder) with files under version control *(aka “repo”)*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4. Clone	Create a local repo from remote (*one time only*)
5.	
6.	
7.	
8.	
9.	
10.	
11.	
12.	
13.	

Term	Description
1. Repository	A directory (folder) with files under version control *(aka "repo")*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4. Clone	Create a local repo from remote (*one time only*)
5. Commit	A small set of changes that accomplish an "atomic task"
6.	
7.	
8.	
9.	
10.	
11.	
12.	
13.	

Commit Messages

They Matter

https://xkcd.com/1296

Term	Description
1. Repository	A directory (folder) with files under version control *(aka “repo”)*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4. Clone	Create a local repo from remote (*one time only*)
5. Commit	A small set of changes that accomplish an “atomic task”
6. Fetch	Check if local is up-to-date with remote
7.	
8.	
9.	
10.	
11.	
12.	
13.	

Term	Description
1. Repository	A directory (folder) with files under version control *(aka "repo")*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4. Clone	Create a local repo from remote (*one time only*)
5. Commit	A small set of changes that accomplish an "atomic task"
6. Fetch	Check if local is up-to-date with remote
7. Pull ⬇	Sync commits to local from remote (*if local is behind remote*)
8. Push ⬆	Sync commits from local to remote (*if remote is behind local*)
9.	
10.	
11.	
12.	
13.	

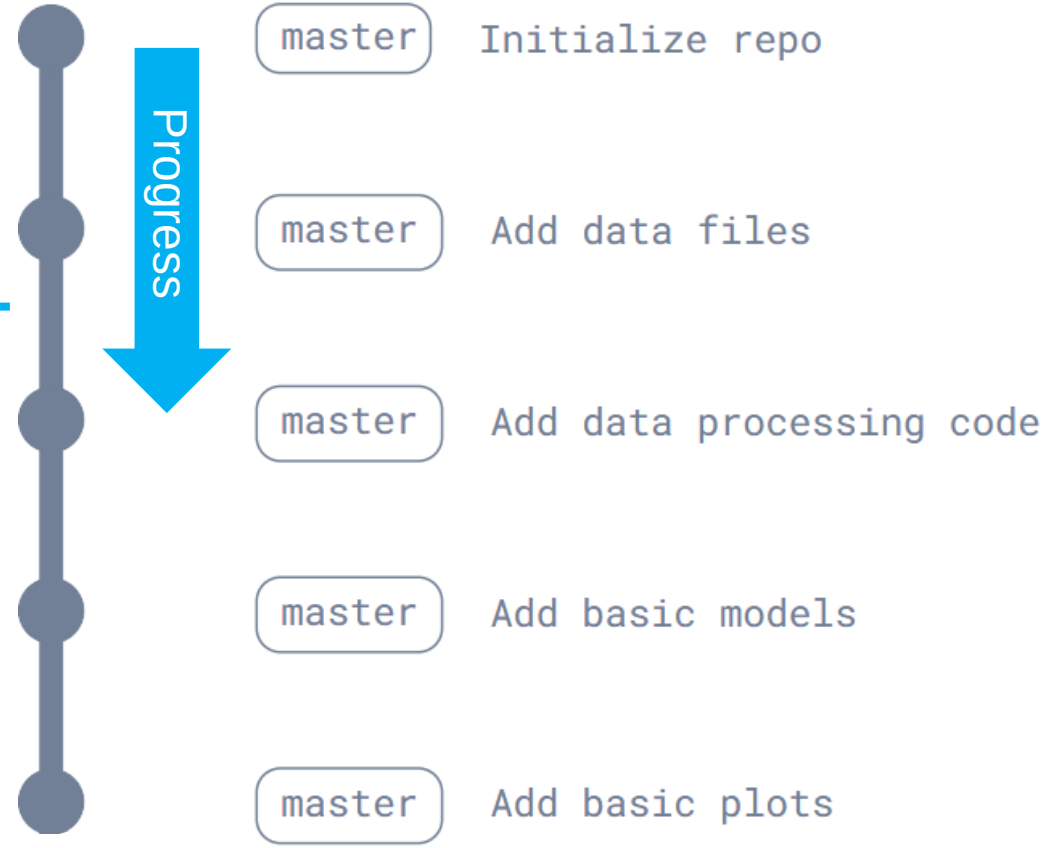
Term	Description
1. Repository	A directory (folder) with files under version control *(aka "repo")*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4. Clone	Create a local repo from remote (*one time only*)
5. Commit	A small set of changes that accomplish an "atomic task"
6. Fetch	Check if local is up-to-date with remote
7. Pull ⬇	Sync commits to local from remote (*if local is behind remote*)
8. Push ⬆	Sync commits from local to remote (*if remote is behind local*)
9. Branch	An isolated version of the repo (*doesn't affect other branches*)
10.	
11.	
12.	
13.	

Term	Description
1. Repository	A directory (folder) with files under version control *(aka "repo")*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4. Clone	Create a local repo from remote (*one time only*)
5. Commit	A small set of changes that accomplish an "atomic task"
6. Fetch	Check if local is up-to-date with remote
7. Pull ⬇	Sync commits to local from remote (*if local is behind remote*)
8. Push ⬆	Sync commits from local to remote (*if remote is behind local*)
9. Branch 	An isolated version of the repo (*doesn't affect other branches*)
10. Main Branch	A special branch storing the "best" version
11.	
12.	
13.	

A “Git Graph”

Visualizes Development History

- Commits constitute progress
- “Master” = “Main” branch

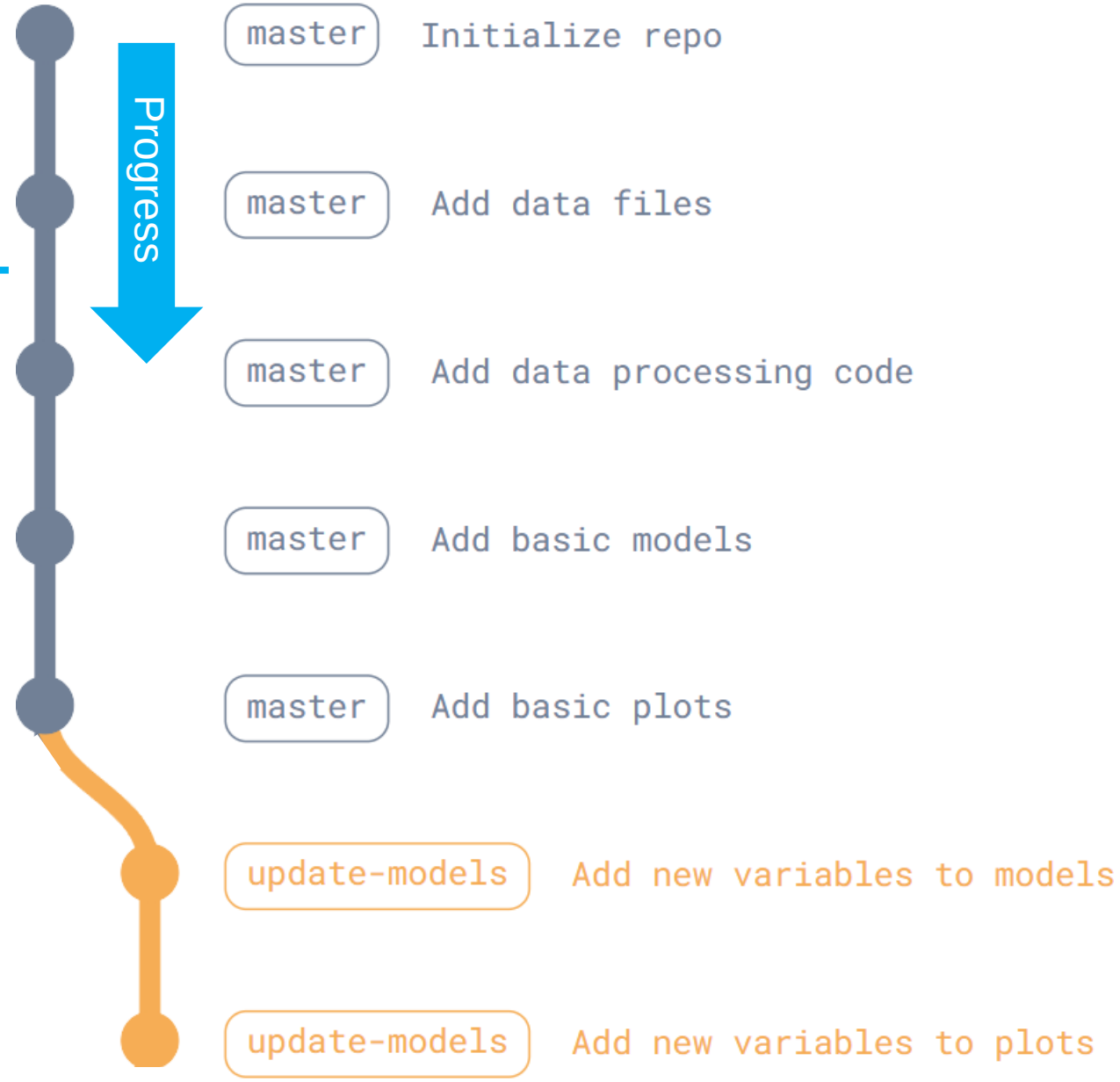


Term	Description
1. Repository	A directory (folder) with files under version control *(aka “repo”)*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4. Clone	Create a local repo from remote (*one time only*)
5. Commit	A small set of changes that accomplish an “atomic task”
6. Fetch	Check if local is up-to-date with remote
7. Pull ⬇	Sync commits to local from remote (*if local is behind remote*)
8. Push ⬆	Sync commits from local to remote (*if remote is behind local*)
9. Branch 	An isolated version of the repo (*doesn’t affect other branches*)
10. Main Branch	A special branch storing the “best” version
11. Feature Branch	A branch storing a version under development
12.	
13.	

A “Git Graph”

Visualizes Development History

- Commits constitute progress
- “Master” = “Main” branch
- Create a branch and add commits
- Feature branch has all upstream commits on main branch
- But main remains unchanged

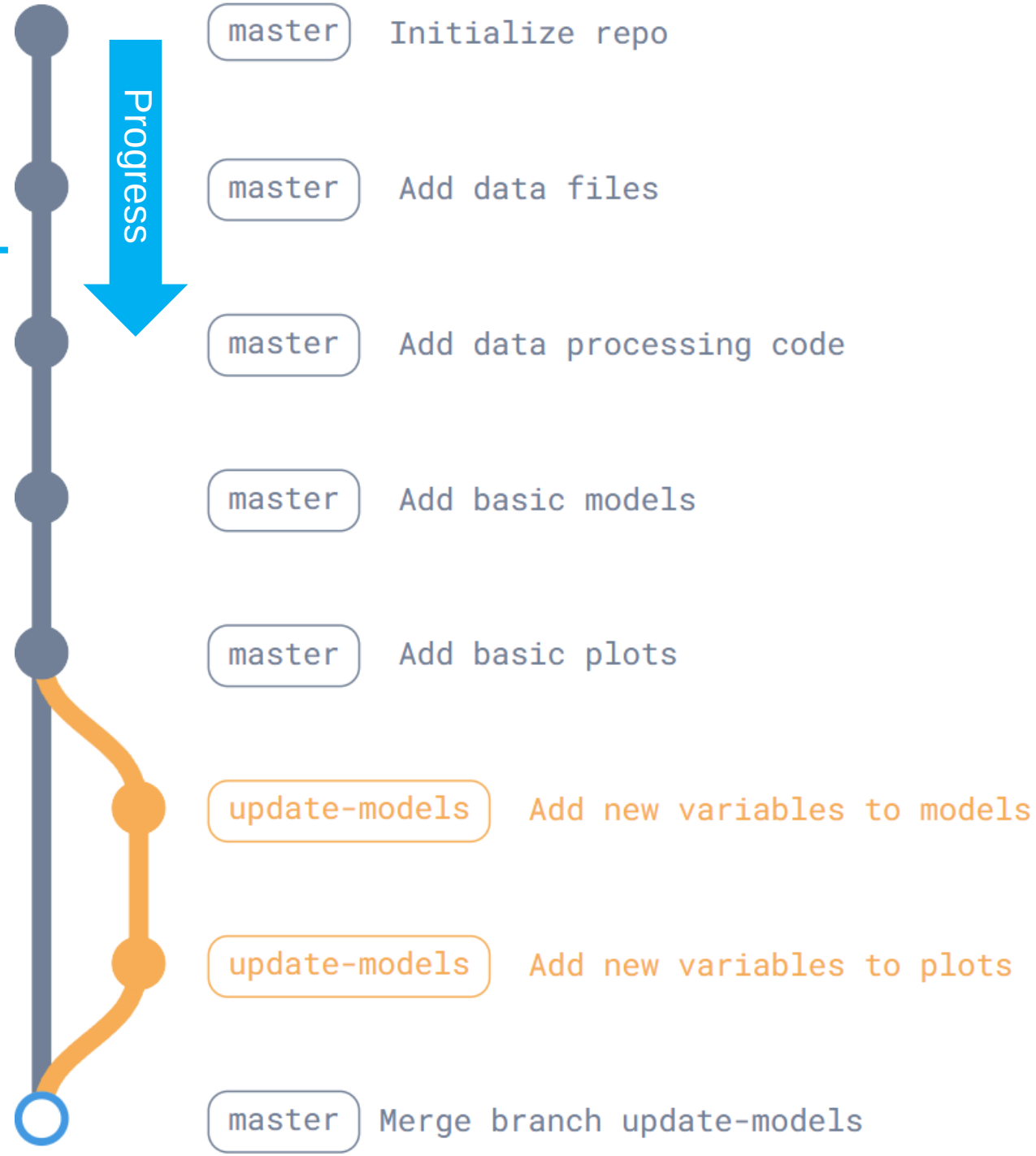


Term	Description
1. Repository	A directory (folder) with files under version control *(aka “repo”)*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4. Clone	Create a local repo from remote (*one time only*)
5. Commit	A small set of changes that accomplish an “atomic task”
6. Fetch	Check if local is up-to-date with remote
7. Pull ⬇	Sync commits to local from remote (*if local is behind remote*)
8. Push ⬆	Sync commits from local to remote (*if remote is behind local*)
9. Branch	An isolated version of the repo (*doesn’t affect other branches*)
10. Main Branch	A special branch storing the “best” version
11. Feature Branch	A branch storing a version under development
12. Merge	Add commits from one branch to another
13.	

A “Git Graph”

Visualizes Development History

- Commits constitute progress
- “Master” = “Main” branch
- Create a branch and add commits
- Feature branch has all upstream commits on main branch
- But main remains unchanged
- If feature commits are beneficial, add them to main with merge
- After merge, delete feature branch



Term	Description
1. Repository	A directory (folder) with files under version control *(aka "repo")*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4. Clone	Create a local repo from remote (*one time only*)
5. Commit	A small set of changes that accomplish an "atomic task"
6. Fetch	Check if local is up-to-date with remote
7. Pull	Sync commits to local from remote (*if local is behind remote*)
8. Push	Sync commits from local to remote (*if remote is behind local*)
9. Branch	An isolated version of the repo (*doesn't affect other branches*)
10. Main Branch	A special branch storing the "best" version
11. Feature Branch	A branch storing a version under development
12. Merge	Add commits from one branch to another
13. Pull (merge) Request	Perform a merge in a documented/collaborative fashion

Miscellaneous Topics

Time Allowing

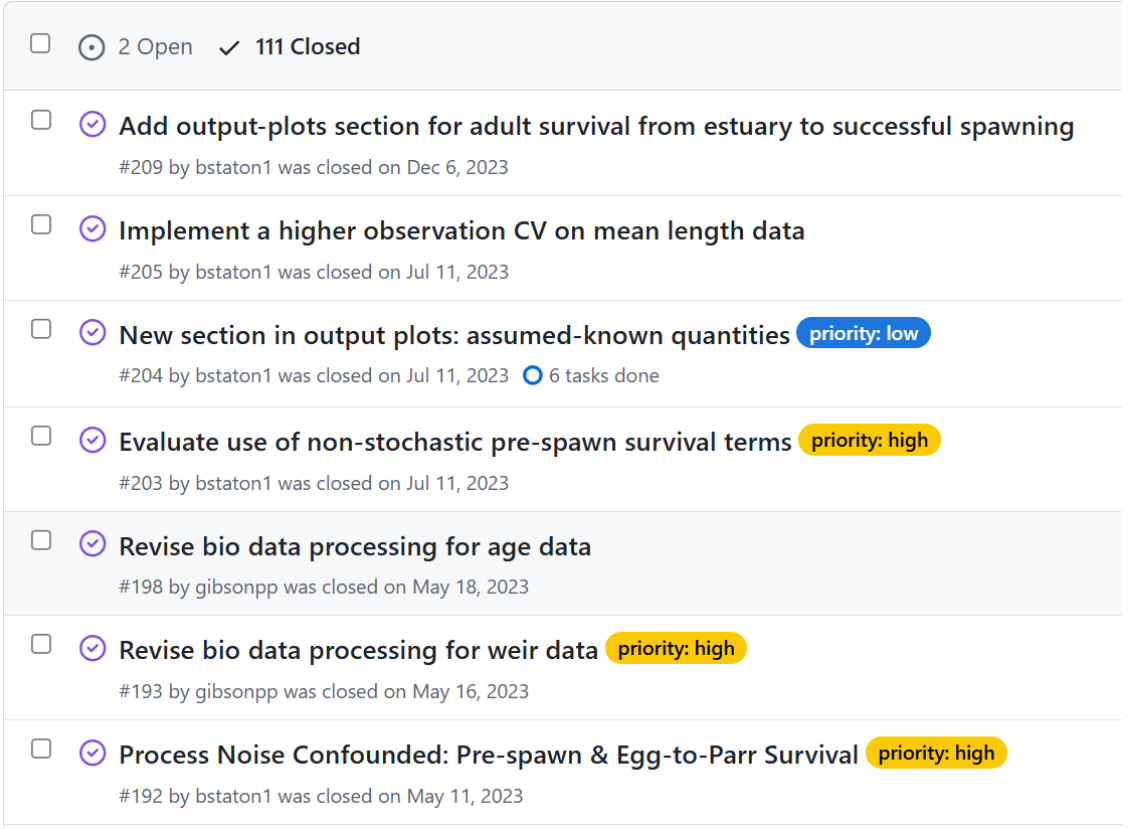
These Topics are More Advanced

Will Require Exploring Some on Your Own

Issue Tracker

Organize Ideas, Next Steps, Discuss Problems

- Online feature only
- “# system” allows cross-referencing
- Can use as something of a notebook
- Comment threads
- Can group issues, tag people
- Uses markdown syntax
- Open vs. closed issues
- Info



`.gitignore`

Exclude some files from tracking

- E.g.,
  - Large output files
  - Things that get re-written often
- Example `.gitignore` file ➡
- Place in main repo directory
  - Subdirectories can have separate ones
- Edits to ignored files cannot be committed, files cannot be pushed

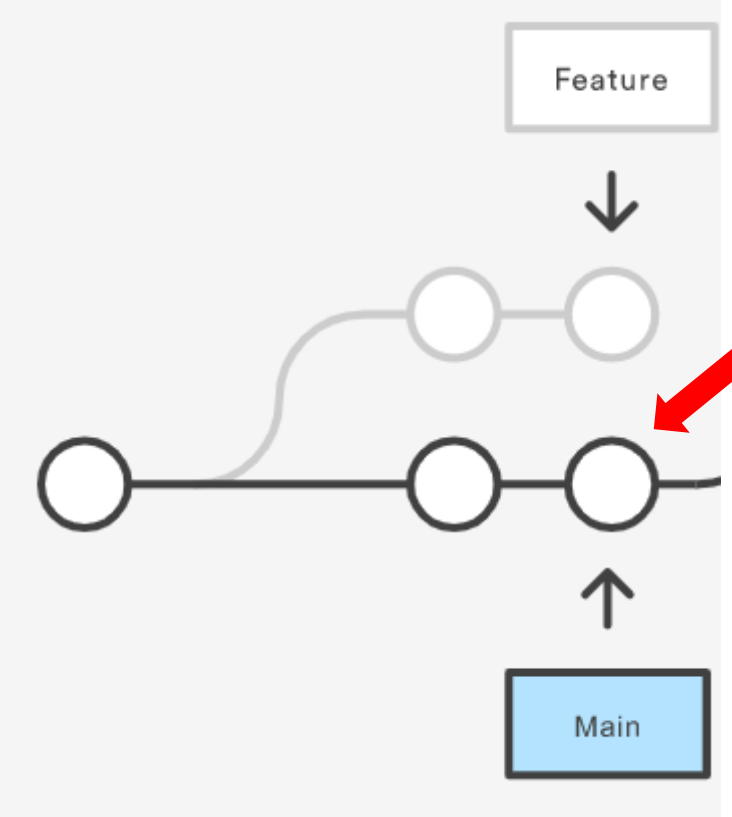
```
# Files associated with R sessions/projects
.Rproj.user/
.Rhistory
.RData
.Ruserdata

# Saved R objects (e.g., large model output)
*.rds

# Temporary files created by Rmarkdown
*.utf8.md
*.knit.md

# HTML files (generated from Rmarkdown)
*.html

# Figure files (can be recreated from source)
*.pdf
*.png
*.jpg
*.jpeg

# Anything in scratch/ subdirectory
scratch/*
```

Merge Conflicts

When Things Go Wrong

- Occurs when merging branches with inconsistent commit history

In these cases, merging feature into main will often cause merge conflicts



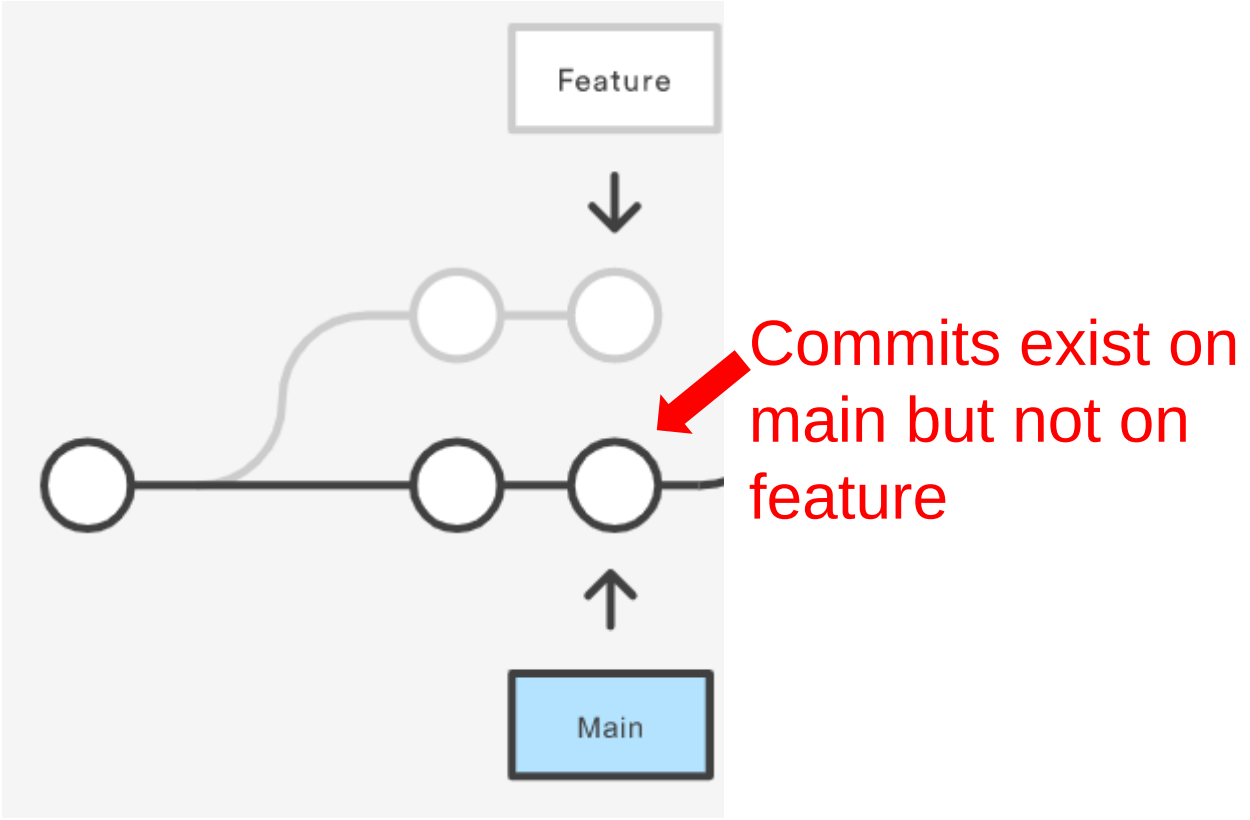
Commits exist on main but not on feature

Merge Conflicts

When Things Go Wrong

- Occurs when merging branches with inconsistent commit history
- Avoid by:
  - Don’t commit to main (*esp. with active branches*)
  - 1 branch = 1 person
  - Otherwise, communicate & pull (*often*)
  - Rebase (*details here*)

In these cases, merging feature into main will often cause merge conflicts



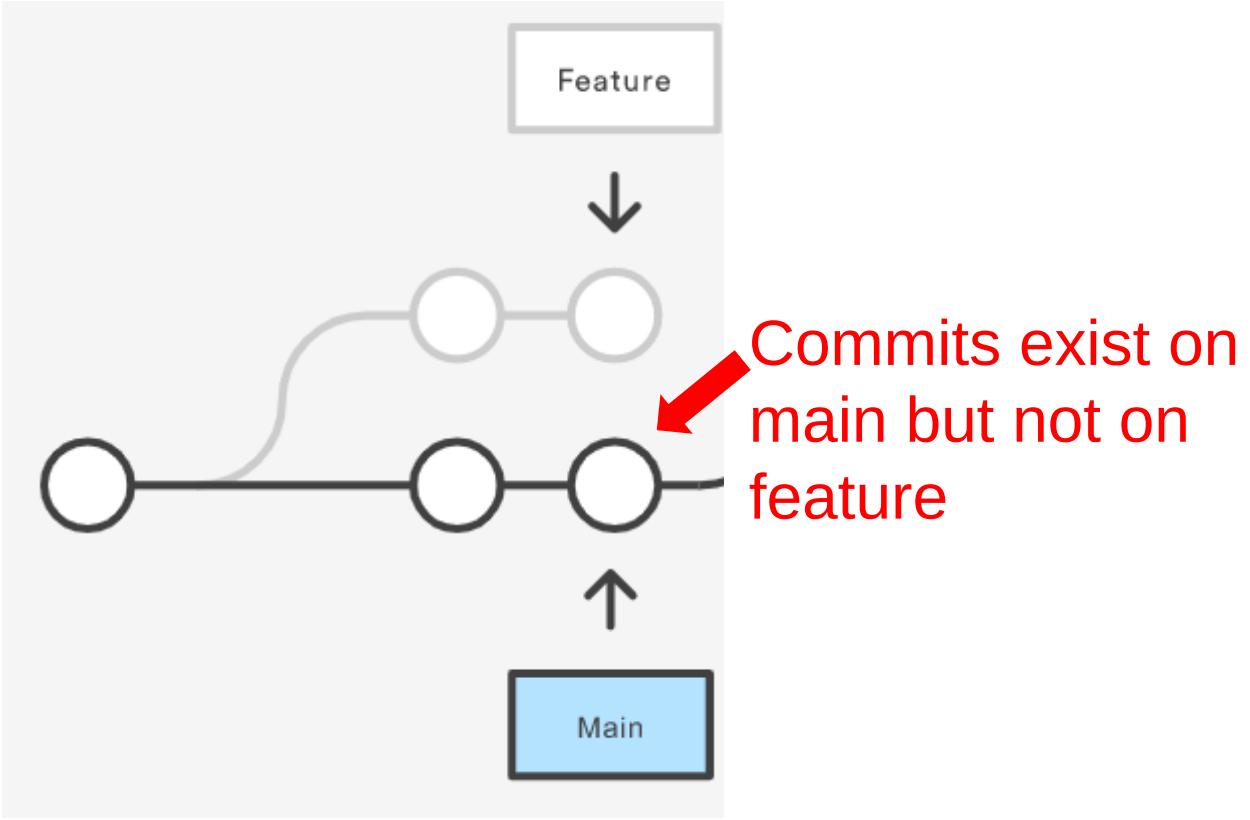
Commits exist on main but not on feature

Merge Conflicts

When Things Go Wrong

- Occurs when merging branches with inconsistent commit history
- Avoid by:
  - Don’t commit to main (*esp. with active branches*)
  - 1 branch = 1 person
  - Otherwise, communicate & pull (*often*)
  - Rebase (*details here*)
- Resolve by selecting which version of conflicting file(s) to keep after merge

In these cases, merging feature into main will often cause merge conflicts

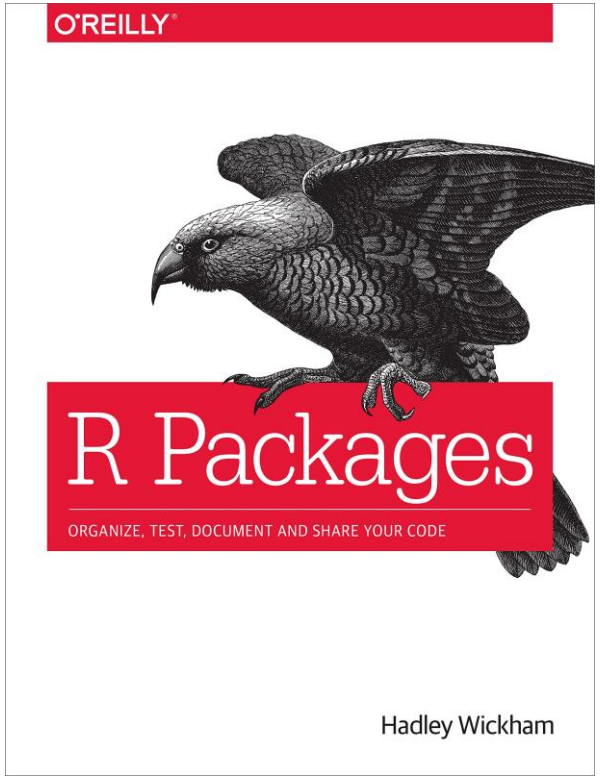


R Packages

Have an R Package? Host it on GitHub!

- R packages are a clean way of sharing code that will be called repeatedly by multiple users
- You could
  - Host it on CRAN (non-trivial)
  - Share the compiled package source files
  - Host it on GitHub!
- Users install via 1-2 lines of code:

```
# install remotes package
install.packages("remotes")

# install package from GitHub
remotes::install_github("user/packageName")
```



Free ebook: https://r-pkgs.org/

Cite Your Code

Need to make a version permanent?

- E.g., for journal article supplementary material
- Zenodo: archival service

Example of Staton et al. (2021)

Accounting for uncertainty when estimating drivers of imperfect detection: An integrated approach illustrated with snorkel surveys for riverine fishes

Article 10.1016/j.fishres.2021.106209

Code 10.5281/zenodo.3928691

Cite Your Code

Need to make a version permanent?

- E.g., for journal article supplementary material
- Zenodo: archival service
- Steps (*more details*):
  1. Link Zenodo to GitHub account
  2. Make your GitHub repo public
  3. Sync Zenodo & turn on tracking
  4. Publish a GitHub release
  5. Each release will receive a DOI, archiving that exact version

Example of Staton et al. (2021)

Accounting for uncertainty when estimating drivers of imperfect detection: An integrated approach illustrated with snorkel surveys for riverine fishes

Article 10.1016/j.fishres.2021.106209

Code 10.5281/zenodo.3928691

Practice Makes Perfect!

(And gives confidence that you won't mess something up)

But don't be too perfect.

Time spent documenting changes is time spent not making changes (progress).

Strongly *advise practicing on a toy analysis before applying to real new or existing analyses. Get the feel for how this works.*

Upcoming PNAMP FWMG Webinar
A "How-To" Tutorial

Term	Description
1. Repository	A directory (folder) with files under version control *(aka "repo")*
2. Local (Repository)	A repository found on your hard drive
3. Remote (Repository)	A repository found online
4. Clone	Create a local repo from remote (*one time only*)
5. Commit	A small set of changes that accomplish an "atomic task"
6. Fetch	Check if local is up-to-date with remote
7. Pull ⬇	Sync commits to local from remote (*if local is behind remote*)
8. Push ⬆	Sync commits from local to remote (*if remote is behind local*)
9. Branch	An isolated version of the repo (*doesn't affect other branches*)
10. Main Branch	A special branch storing the "best" version
11. Feature Branch	A branch storing a version under development
12. Merge	Add commits from one branch to another
13. Pull (merge) Request	Perform a merge in a documented/collaborative fashion