Version control with Git and GitHub

Oxford Biomedical Data Science Training Programme

Overview

- Why version control?
- What version control offers
- Git and GitHub
- Working with Git
  - ➢ Creating a repository
  - ➢ Staging and committing files
  - ➢ Branching and merging
  - ➢ Working with remote repositories
  - ➢ Undoing changes

Why version documents?

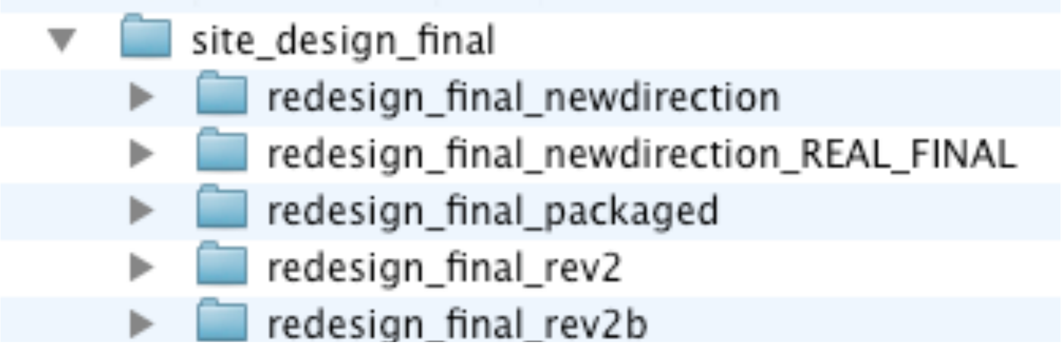
- To try out new ideas
- To try out new structures
- To gather contributions from multiple authors
- To reuse text for new purposes
- To keep a historical record
- To backup important work

The old way

```
$ nano script.py               # Add code until it works
$ cp script.py script_v1.py    # Make a copy
$ nano script_v1.py            # Add new feature
$ diff script.py script_v1.py  # View changes
... and repeat
```

The old way

- What if you want to change more than one file in a project?
- What if you are working on multiple projects?
- What if multiple people are working on the same file?



site_design_final
redesign_final_newdirection
redesign_final_newdirection_REAL_FINAL
redesign_final_packaged
redesign_final_rev2
redesign_final_rev2b

Version control

- Wouldn't it be nicer to:
  - Always work on the same script (don’t keep changing the name)
  - Tag changes with notes/explanations
  - Attribute ownership to individual changes
  - Record changes to all your files in a directory/project
  - Enable multiple developers to work on the same codebase
- **Version control** to the rescue!
  - Version control is a system that records changes to a set of files over time so that you can recall specific versions later

Git and GitHub

- What is Git?
    - It’s a version control system
    - Distributed, graph-based
- What is GitHub?
    - A web service that makes working with version control easier
        - Online code hub – access anywhere
        - Share, publish and release your code
        - User friendly interface to see changes in your code
        - Easily collaborate with others in a common project
        - Easily perform continuous integration in your code

Create a repository folder

```
$ cd /ifs/obds-training/apr20/<user>
$ mkdir -p devel/<repo-name>     # choose a name for the repository e.g. obds_training, you will use this throughout the course for storing your personal code
$ cd devel/obds_training
$ ls –al
```

Initialise repository

```
$ which git       # check git is available
$ git status      # check directory status
$ git init        # initialise repository
$ ls -al          # see .git folder
$ git status      # check repo status
```

Use this repository for your own code and notes throughout the course

Configure Git

- Global configuration (~/.gitconfig file) – check current configuration

```
$ git config --global user.name
$ git config --global user.email
$ git config --global core.editor
$ git config --global --list  # prints out above info
```

- Modify your configuration

```
$ git config --global user.name “Your Name”
$ git config --global user.email “email@email.com”
$ git config --global core.editor nano
```

Workflow

- Create/edit file
- `git add`
  - ➢ Tell Git you want to track it
- `git commit`
  - ➢ Save those differences with a description of what you have done
- `git diff`
  - ➢ See changes between commits

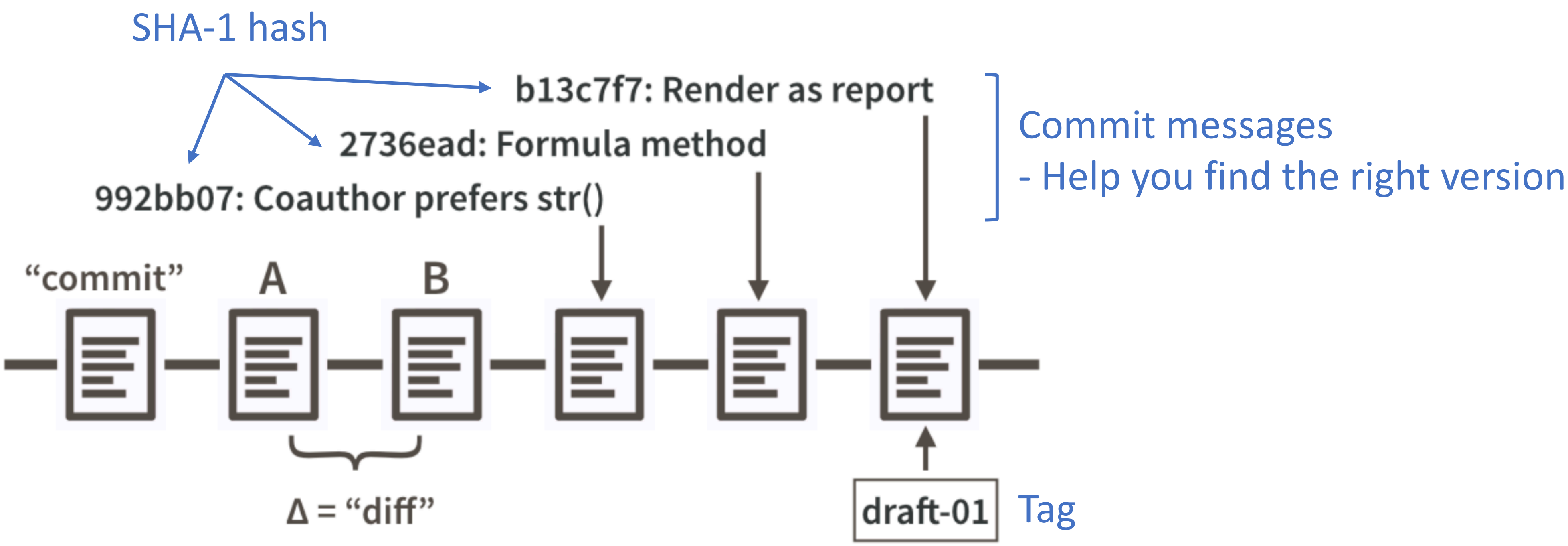
> “If you have ever versioned a file by adding your initials or the date, you have effectively made a commit, albeit only for a single file. It is a version that is significant to you and that you might want to inspect or revert to later”

https://peerj.com/preprints/3159.pdf

Adding files to a repository

```
$ nano file1.txt                   # Create file & add line
$ git status                       # Check repo status
$ git add file1.txt                # Track file
$ git status                       # Check repo status
$ git diff --cached                # Examine changes
$ git commit –m “added first line” # Save changes
$ git status                       # Check repo status
$ git log                          # Check commit history
$ git log --oneline                # Abbreviated history
```

Commit history

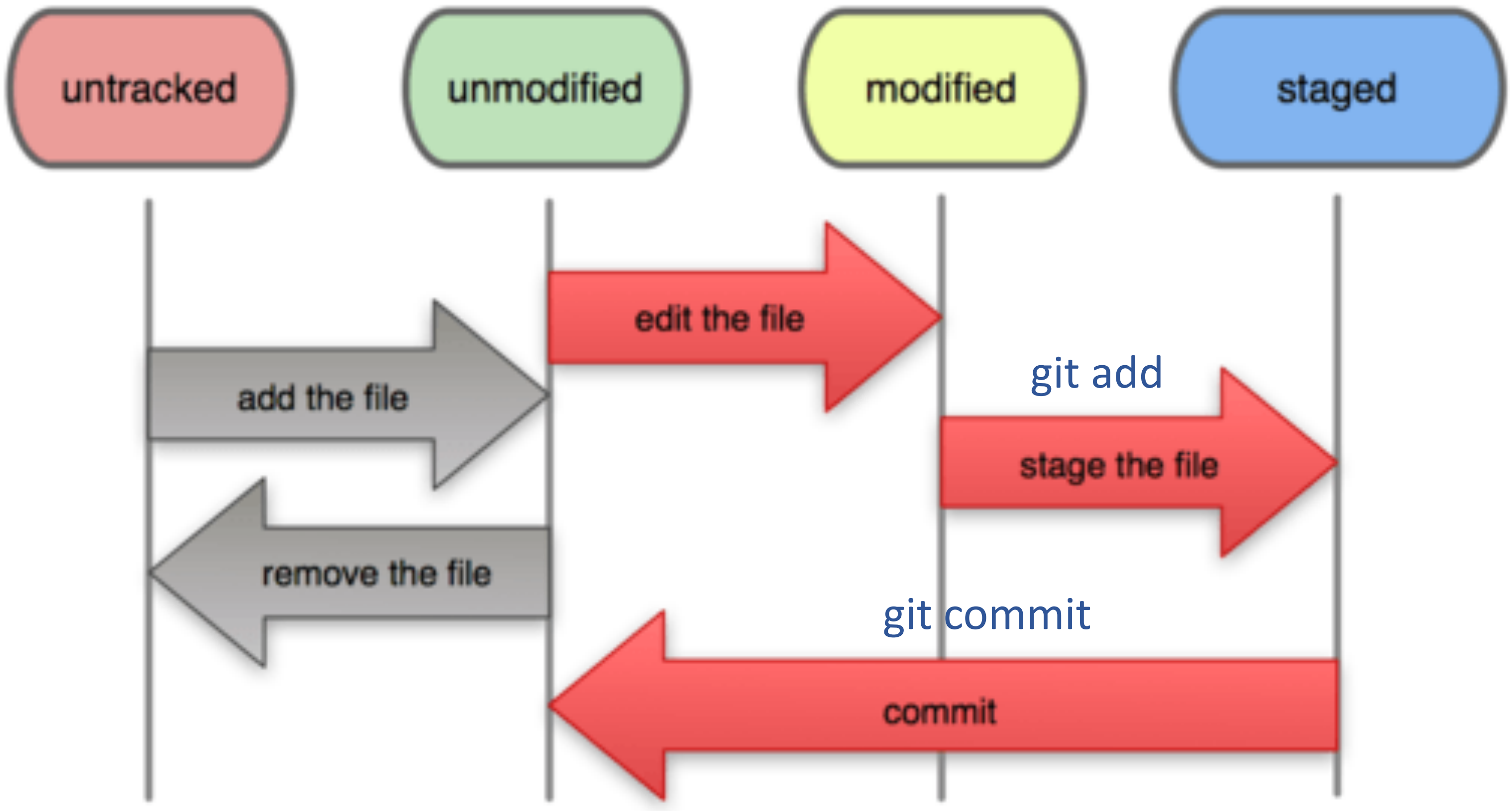

SHA-1 hash
b13c7f7: Render as report
2736ead: Formula method
992bb07: Coauthor prefers str()
Commit messages
- Help you find the right version
"commit"
A
B
Δ = "diff"
draft-01
Tag

https://doi.org/10.7287/peerj.preprints.3159v2

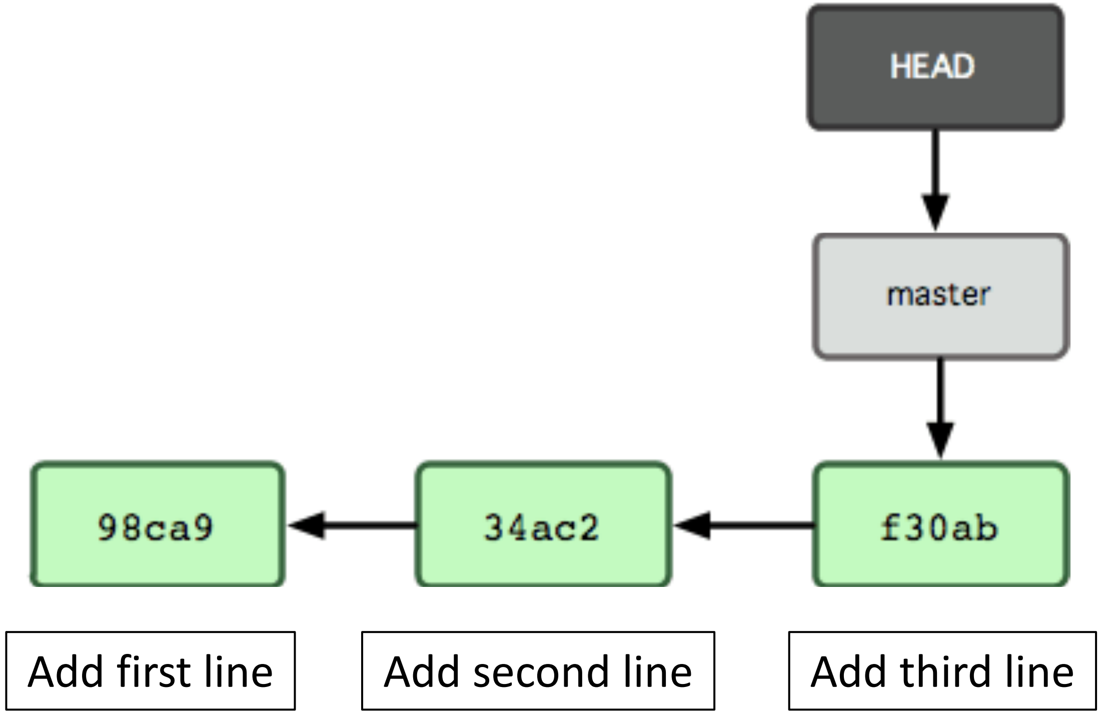
Modifying files

```
$ nano file1.txt                     # Edit file
$ git status                         # Check repo status
$ git add file1.txt                  # Track file
$ git status                         # Check repo status
$ git commit –m “added second line”  # Save changes
$ git status                         # Check repo status
$ git log --oneline                  # Check commit history
```

File status lifecycle



Git HEAD



Add first line

Add second line

Add third line

Pointer to current branch & commit

Branch

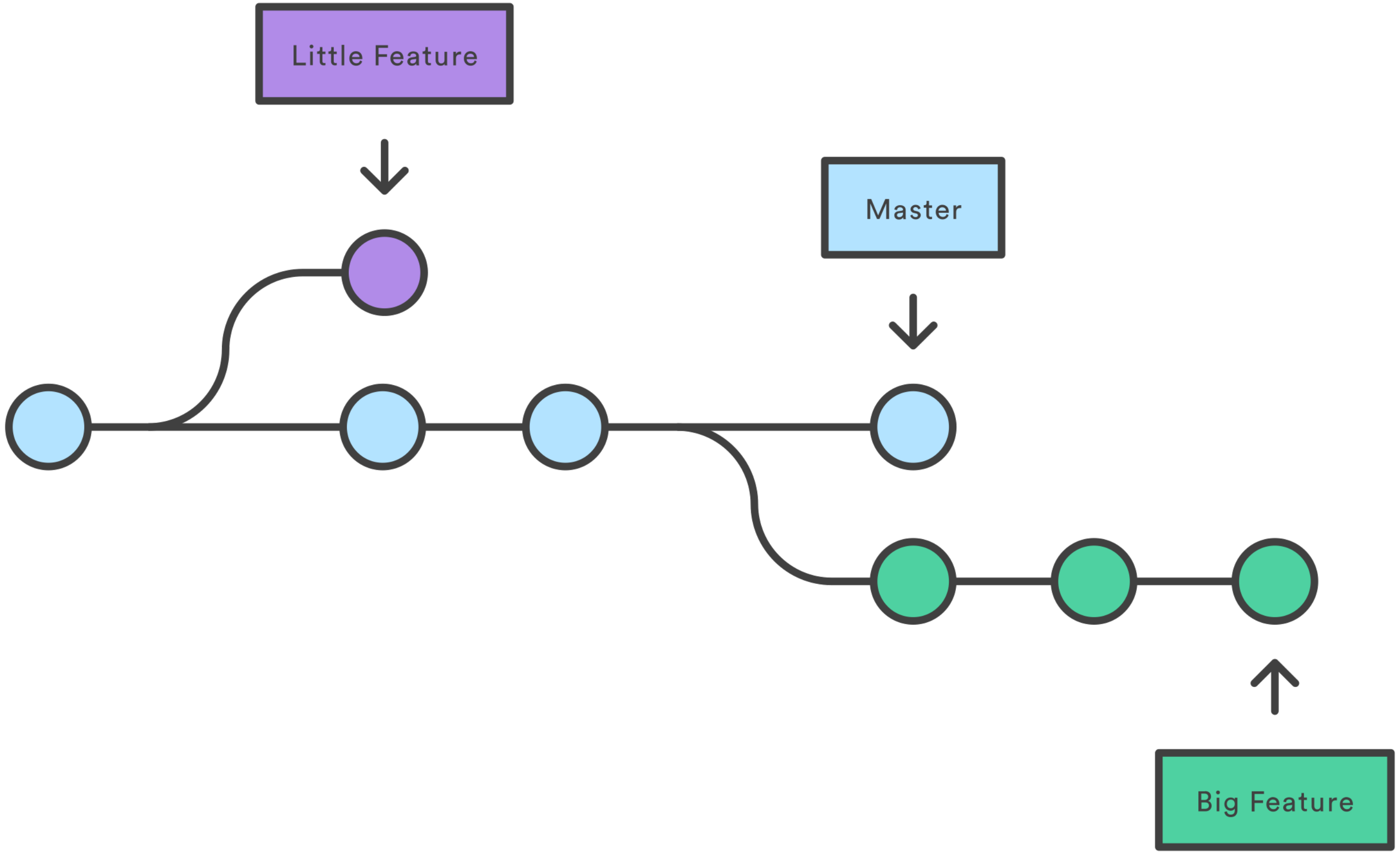
Commits

Commit messages

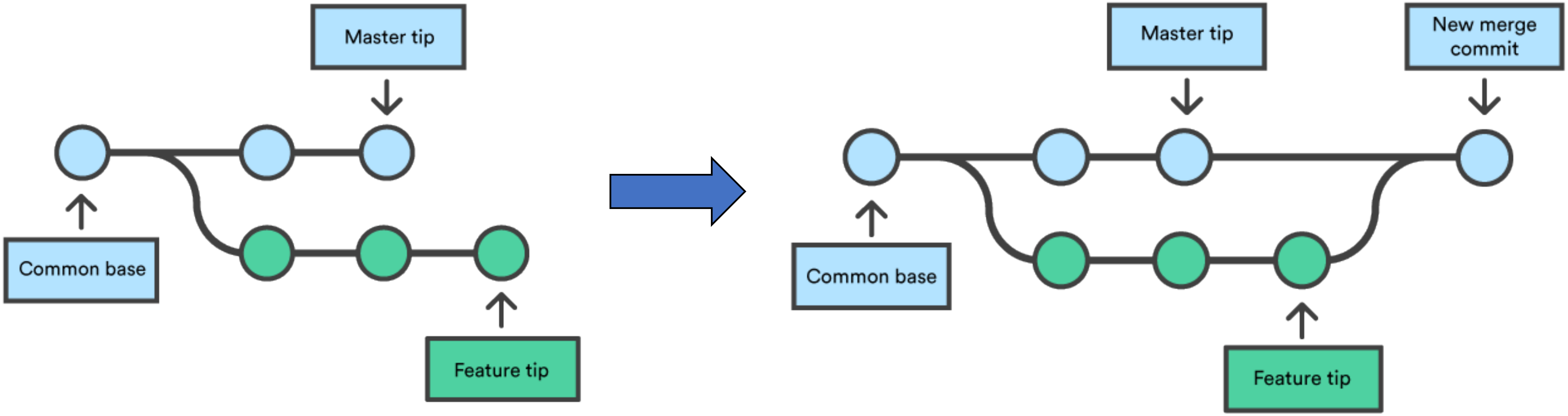
Branching and merging

- Branches enable you to test out new things without changing master
  - ➢ Keep a working version of the code
- Allow multiple users to modify the same code at the same time
  - ➢ Each users can work on their own branch
- Branches are lightweight
  - ➢ Just pointers, so quick and easy to make

Branching and merging



Branching and merging



git branch

```
$ git branch                 # list branches
$ git branch fix-1           # create new branch called fix-1
$ git checkout fix-1         # switch to branch fix-1
$ git branch                 # list branches - * has moved
$ nano file1.txt             # Edit file
$ git status                 # Check repo status
$ git add file1.txt          # Stage file
$ git commit –m “fixed typo in file1.txt”   # Save file
```

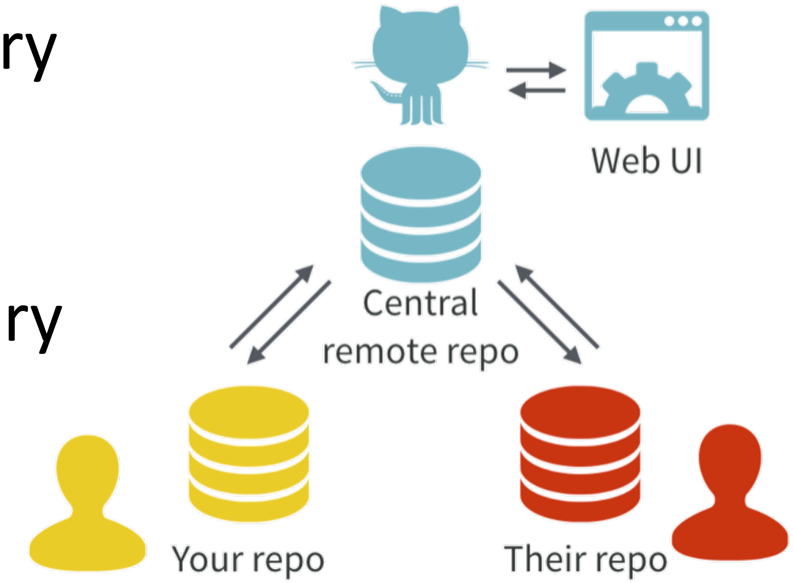
git merge

```
$ git branch            # confirm branch is fix-1
$ git checkout master   # Switch to master branch
$ less file1.txt        # file1.txt not modified on master
$ git merge fix-1       # Merge branch with master
$ less file1.txt        # file1.txt modified on master
$ git branch –d fix-1   # Delete merged branch
```

N.B. if multiple users modify the same code in the same file, then manual resolution of conflicts will be required

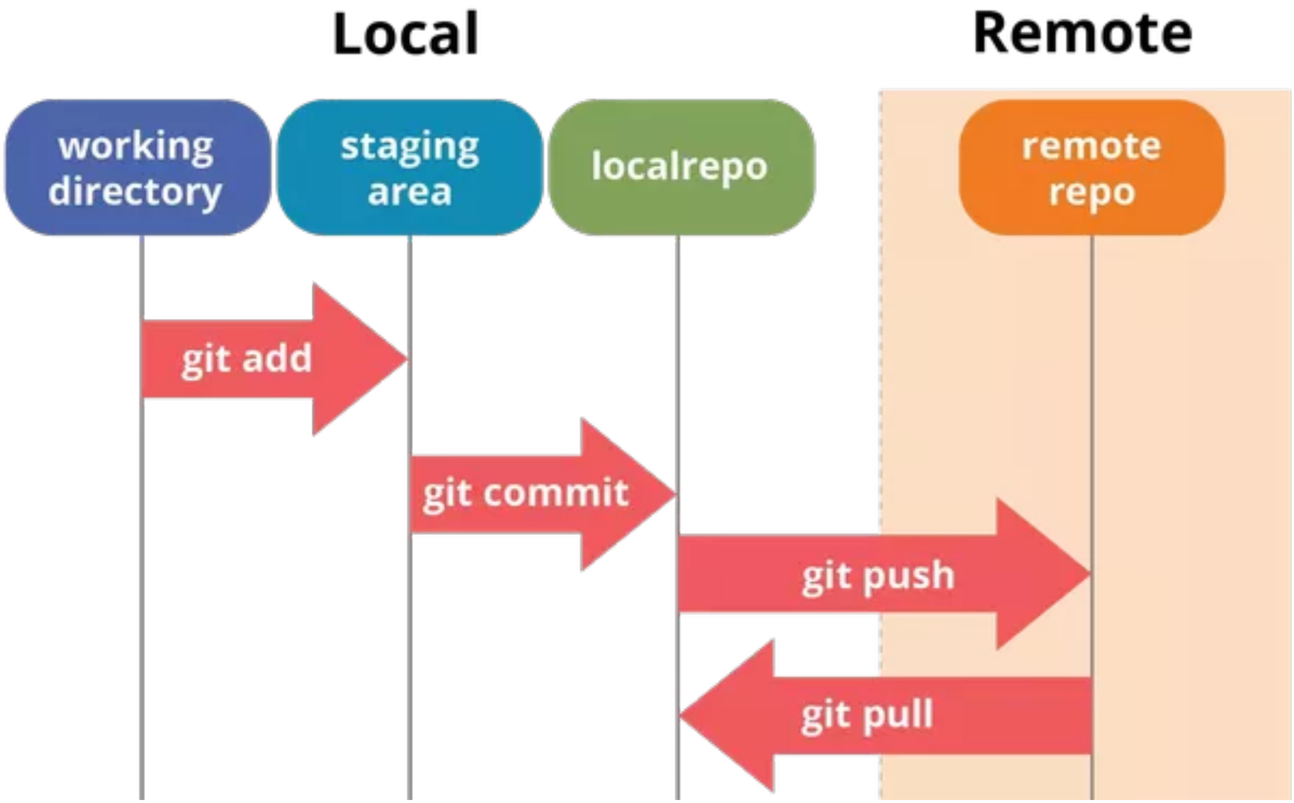
GitHub

- Team development
- Everyone has their own copy of the repository
  - ➢ Can work offline
  - ➢ Can work simultaneously
- Stay in sync through remote central repository



https://doi.org/10.7287/peerj.preprints.3159v2

Working with remotes



GitHub setup

- Register for GitHub
  - https://github.com/join
- SSH key – make sure you follow the instructions for Linux
  - https://help.github.com/en/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account

git remote

- First, create your new repo on github.com (don’t initialise with README) – best to give same name as local repo
- `$ git remote -v       # see remotes`
- `$ git remote add origin ...git (use SSH) e.g. git remote add origin git@github.com:lc822/obds-repo.git`
- `$ git remote –v       # should see your remote`
- `$ git push –u origin master    # push to GitHub repo`
- You should now see file1.txt in your GitHub repository

git remote

- Go to github.com and make some changes to your file online

-
  ```
  $ git pull origin master       # check file1.txt in
  local repository for changes
  ```

Pull requests

Branch

Develop features on a branch and create a pull request to get changes reviewed.

Discuss

Discuss and approve code changes related to the pull request.

Merge

Merge the branch with the click of a button.

Pull requests

```
$ git pull origin master     # Make sure you are up-to-
date
$ git checkout -b fix-2      # Create and switch to a
new branch
$ nano file1.txt             # Edit file
$ git add file1.txt          # Stage changes
$ git commit –m “Fixed another typo”
$ git push origin fix-2      # Push branch to remote
```

Pull requests

- Go to github.com and see result
- Make pull request (Compare and pull request → Create pull request)
- Check pull request and merge to master

https://www.atlassian.com/git/tutorials/merging-vs-rebasing

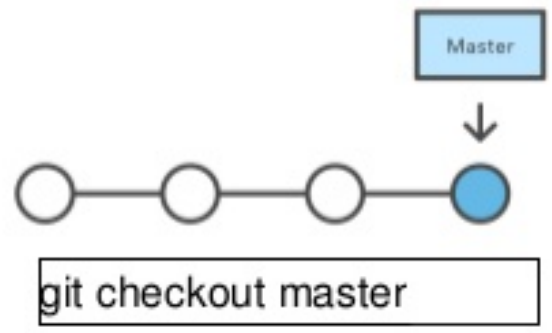
Pull requests

- Delete remote branch on GitHub
- Update local master branch:
  - ➢ `git checkout master`
  - ➢ `git pull origin master`
- Delete local branch:
  - ➢ `git fetch --prune      # remove any reference to fix-2 branch on remote (no longer exists)`
  - ➢ `git branch -d fix-2   # delete local fix-2 branch`

Undoing changes

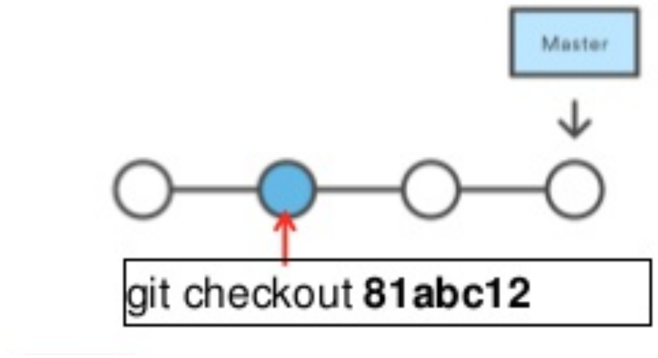
- Just looking around: `git checkout`

Attached HEAD



```
git checkout master
```

Detached HEAD

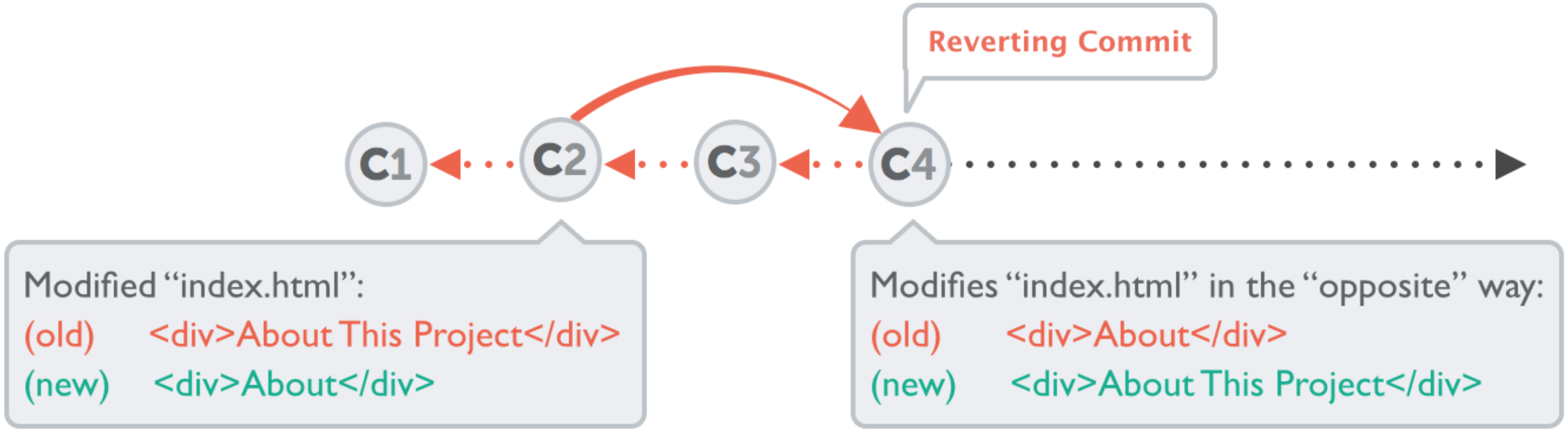


```
git checkout 81abc12
```

Head not pointing to latest commit

Undoing changes

- Undo changes and keep history: `git revert`

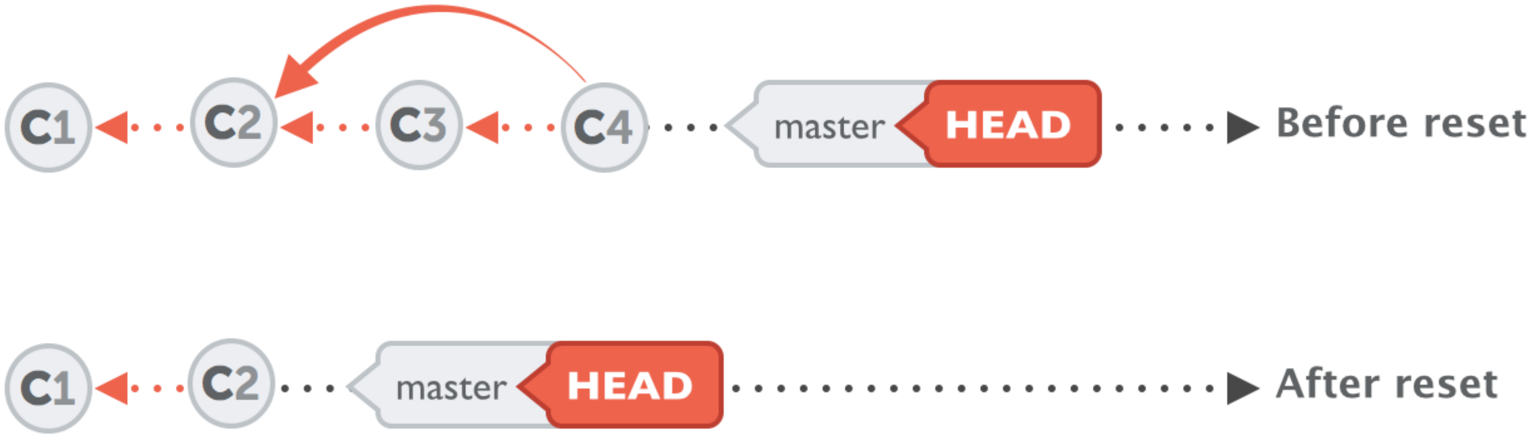


Modified "index.html":
(old) <div>About This Project</div>
(new) <div>About</div>

Modifies "index.html" in the "opposite" way:
(old) <div>About</div>
(new) <div>About This Project</div>

Undoing changes

- Undo changes and remove history: `git reset`



git checkout

```
$ git log --oneline          # Check log to find commit
$ git checkout <commit>      # Choose commit
$ less file1.txt             # Take a look at the old version
$ git checkout master        # Return to the current commit
```

git revert (one commit)

```
$ git log –oneline        # Show commit logs
$ git revert HEAD         # Revert latest commit
$ git log --oneline       # Check effect of revert
$ git status
$ less file1.txt
$ git push origin master   # push local changes to remote repo
```

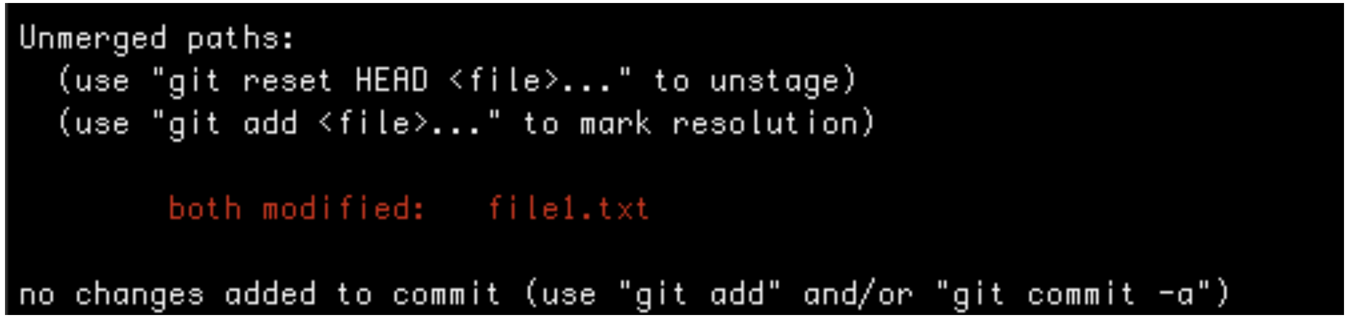
git revert (multiple commits)

```
$ git log --oneline     # choose a previous commit (not
most recent)
$ git revert 9e0ad3a    # git revert to that commit
```

```
error: could not revert 9e0ad3a... edit on fix-4
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'
```

git revert (multiple commits)

```
$ git status          # tells you which file has a problem
(will be file1.txt for us)
```



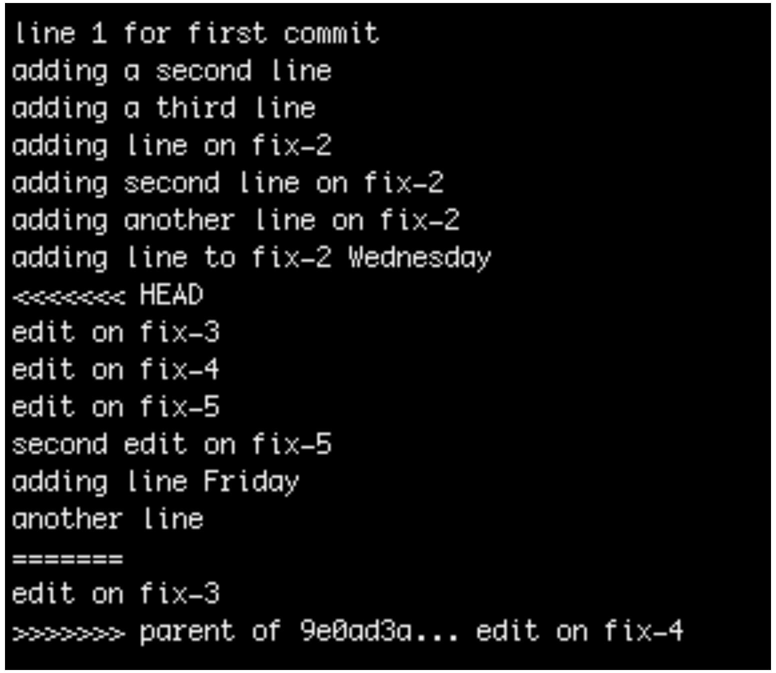
```
Unmerged paths:
  (use "git reset HEAD <file>..." to unstage)
  (use "git add <file>..." to mark resolution)

        both modified:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

git revert (multiple commits)

```
$ nano file1.txt    # manually resolve issues
```



```
line 1 for first commit
adding a second line
adding a third line
adding line on fix-2
adding second line on fix-2
adding another line on fix-2
adding line to fix-2 Wednesday
<<<<<<< HEAD
edit on fix-3
edit on fix-4
edit on fix-5
second edit on fix-5
adding line Friday
another line
=======
edit on fix-3
>>>>>>> parent of 9e0ad3a... edit on fix-4
```

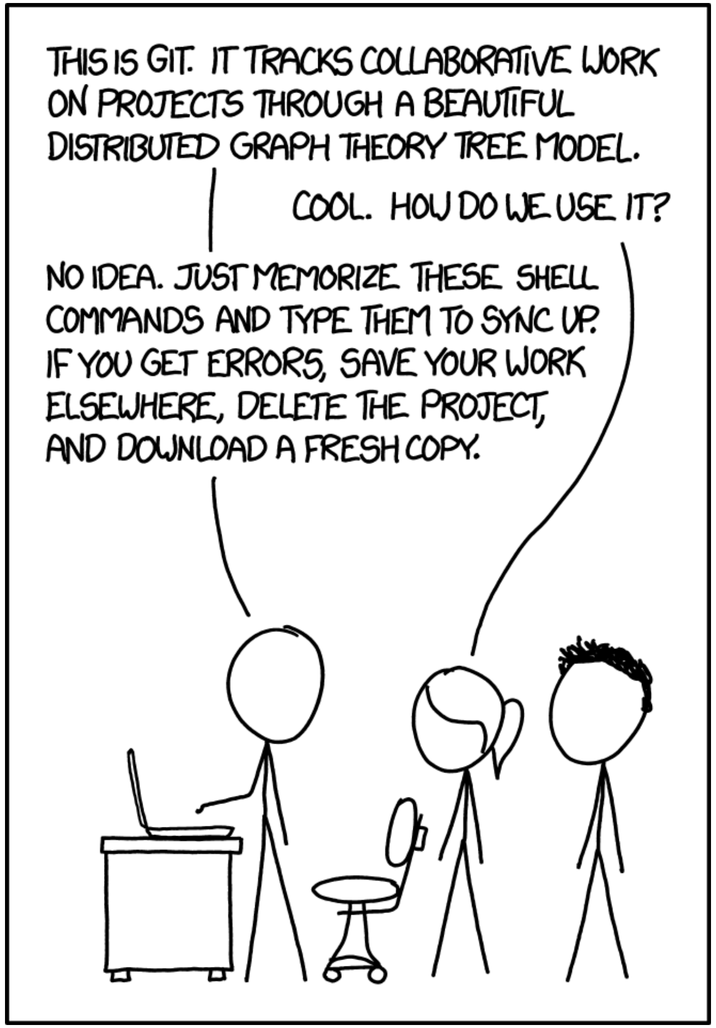
- Need to fix the part between <<<<<<<<< HEAD and >>>>>>> parent of 9e0ad3a
- Change to how you want it to be

git revert (multiple commits)

```
$ git add file1.txt
$ git commit –m “Manually resolved conflicts”
```

git reset

```
$ git log --oneline        # Find commit to go back to
$ git reset <commit-hash>  # Choose commit, removes
history after commit
$ git checkout file1.txt
$ git log --oneline        # Check effect of reset
$ git status
$ cat file1.txt
$ git push –f origin master    # force push
```


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Which files should you track?

- Code
- README.md
- Test data
- Configuration files
  - e.g. Conda yml files

- Personal webpages
- Wiki
- Gather and share info
  - Course info

Tips

- Be descriptive in your commit messages
  - ➢ Be kind to future you
- Always include a README.md file in your repository
- Dedicate a folder to Git
  - ➢ Do not have nested Git directories

git clone OBDS_Training_Apr_2020

We are now going to clone the OBDS-Training/OBDS_Training_Apr_2020 repository that we will use throughout the course

```
$ cd devel/
$ git clone ...git (use SSH)
$ git remote -v
$ git push origin master
$ git pull origin master
```

Exercise

- Everyone pull, edit one line of “test-file.txt” and push
- Resolve conflicts by opening file and manually resolving

Useful resources

- https://www.atlassian.com/git/tutorials
- https://doi.org/10.7287/peerj.preprints.3159v2
- https://guides.github.com
- Guided exercise – https://try.github.io