McGill initiative in Computational Medicine

Introduction to Git and GitHub

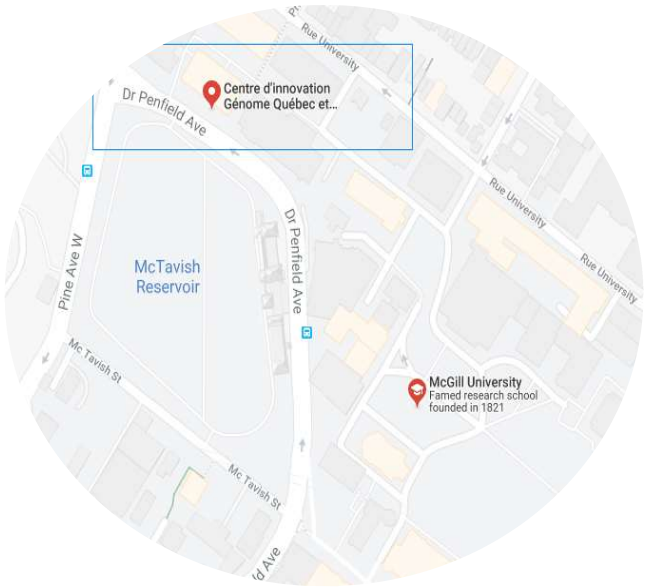
Workshop Lead: Adrien Osakwe

February 9th, 2024

McGill initiative in Computational Medicine

McGill initiative in Computational Medicine

Mission statement: deliver quality workshops designed to help biomedical researchers develop the skills they need to succeed.



Location: 740 Dr. Penfield Avenue, Montreal, Quebec

Scan the QR code to sign up for our **mailing list**

Contact: workshop-micm@mcgill.ca

Winter 2024 MiCM Workshops Series

Sign up for our mailing list for updates!

MyInvolvement Page

Workshop	Date	Location	Registration
How to think in Code	Feb. 2 9AM-11AM	McIntyre Room 325	Open
Intro to UNIX and HPC	Feb. 7 9AM-1PM	McIntyre Room 325	Open
Git and GitHub	Feb. 9 1PM-5PM	Arts Room 150	Open
Intro to R (Part 1)	Feb. 14 9AM-1PM	Macdonald Engineering Building Room 10	Open
Data Analysis in R (Part 2)	Feb. 19 1PM-5PM	680 Sherbrooke Room 1279	Open
Intro to Python (Part 1)	Feb. 20 9AM-1PM	McIntyre Room 325	Open
Data Analysis in Python (Part 2)	Feb. 23 1PM-5PM	Arts Room 150	Open
Meta-analysis of Genetic Association Results	Mar. 13 10AM-12PM	Education Room 113	Open
WGS Data and Variant Calling	TBA	TBA	TBA
GWAS and PRS	TBA	TBA	TBA
Transcriptomics	TBA	TBA	TBA

https://www.mcgill.ca/micm/training/workshops-series

Outline for this workshop:

1. Intro to Git and Version Control Systems
2. Basic Git functions
3. Advanced Git functions

4. Working with GitHub
5. Group Activity
6. Miscellaneous Features

7. Conclusions

Hands-on Activities:

- Quick Review Polls
- Git/GitHub exercises

What you will need:

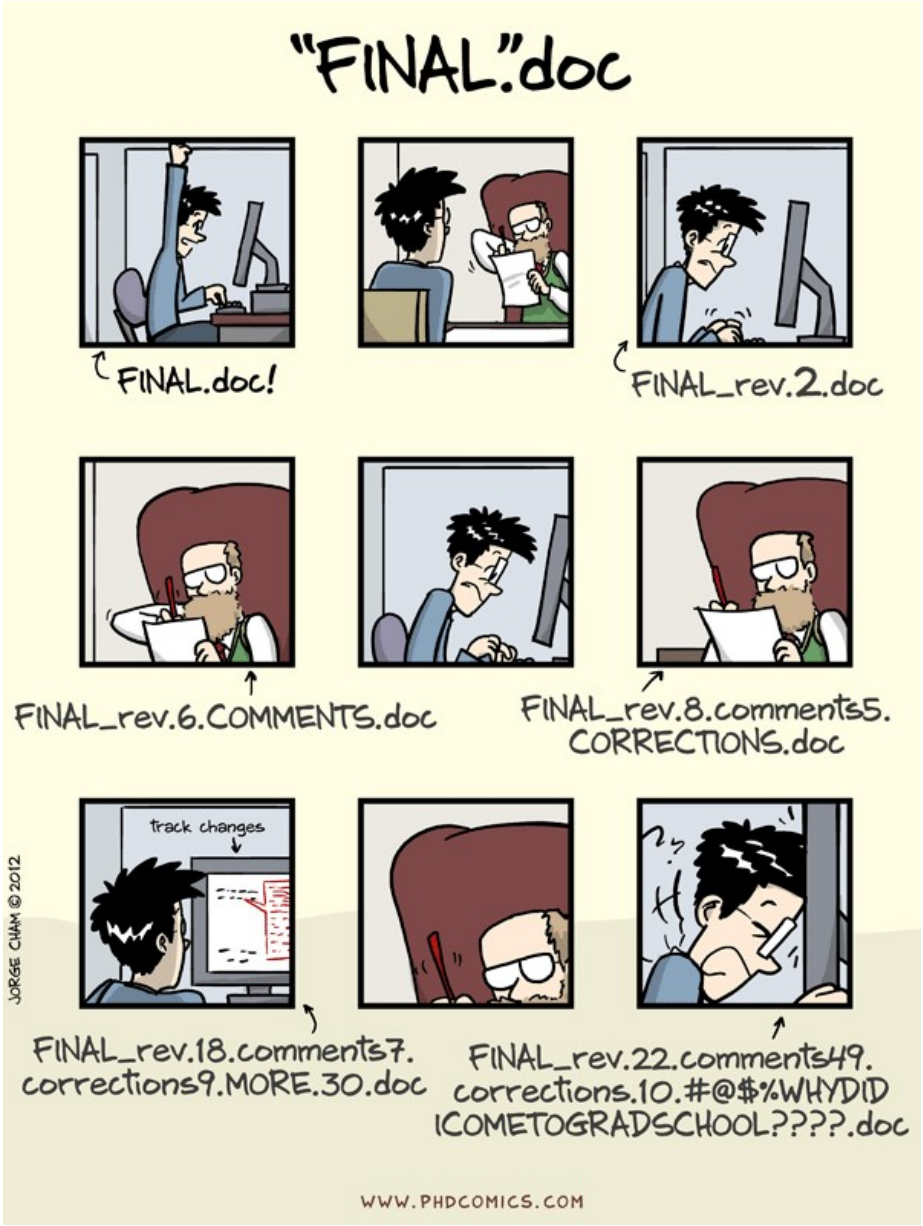
1. GitHub Desktop App and GitHub Account
2. A Text editor
3. UNIX-based Command Line (time-allowing)

What you WILL learn

- Basic theory and features behind Git and GitHub
- How to manage local and remote repositories

What you will NOT learn

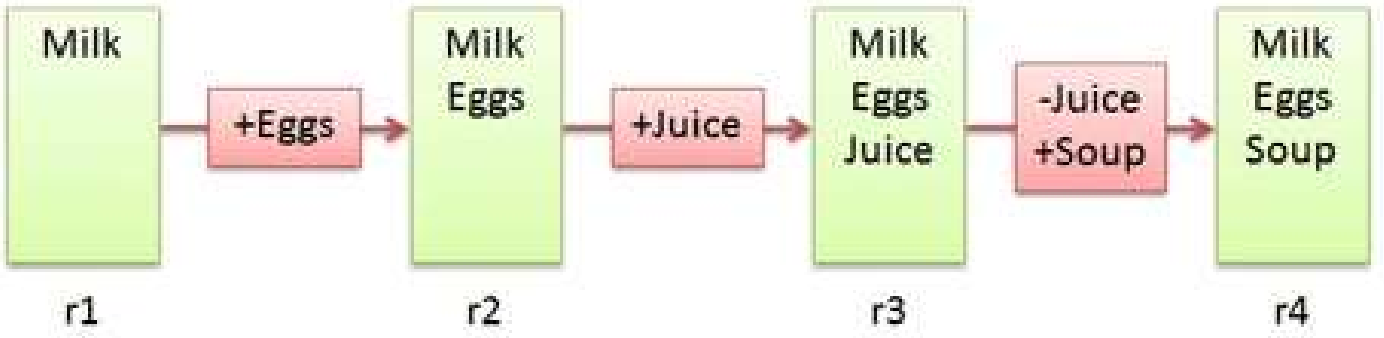
- How to code
- Data Analysis Pipelines


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Version Control Systems

- Make it easier to document changes
- Less confusion
- Like an unlimited “undo” button

Groceries.txt



Git

- A type of version control system
- Open-source
- Most popular VCS currently in use
- Creates a “Git Repository” which stores all the modified instances of your project

What kind of files does Git track?

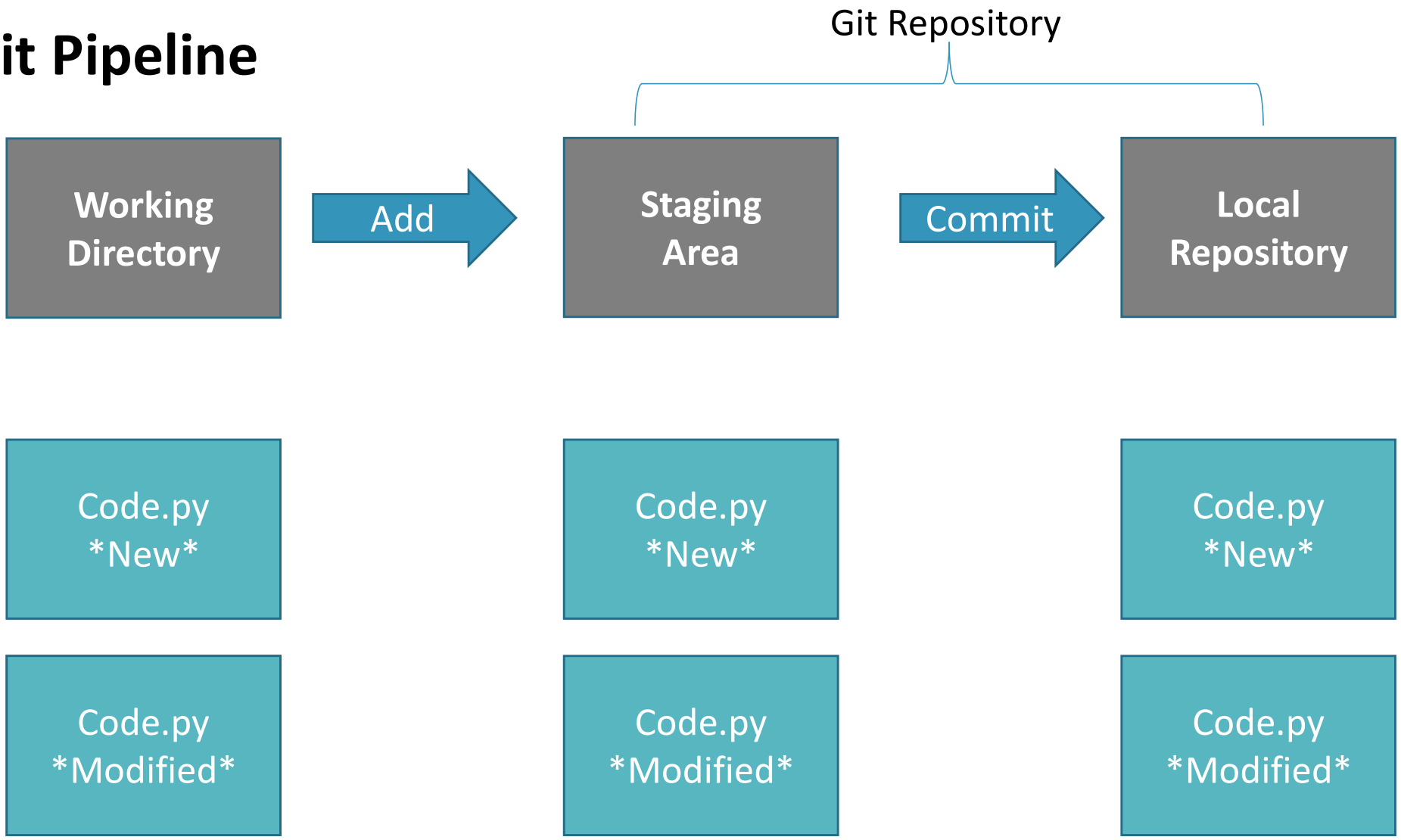
- Can track any file but works best for **pure text files** (.txt,.R,.py etc.)
- Word/PowerPoint are in binary; Git cannot display the changes occurring in these files
- Still useful for storing such files, just cannot use all of Git's features
- Markdown and LaTeX can be used instead of these binary files

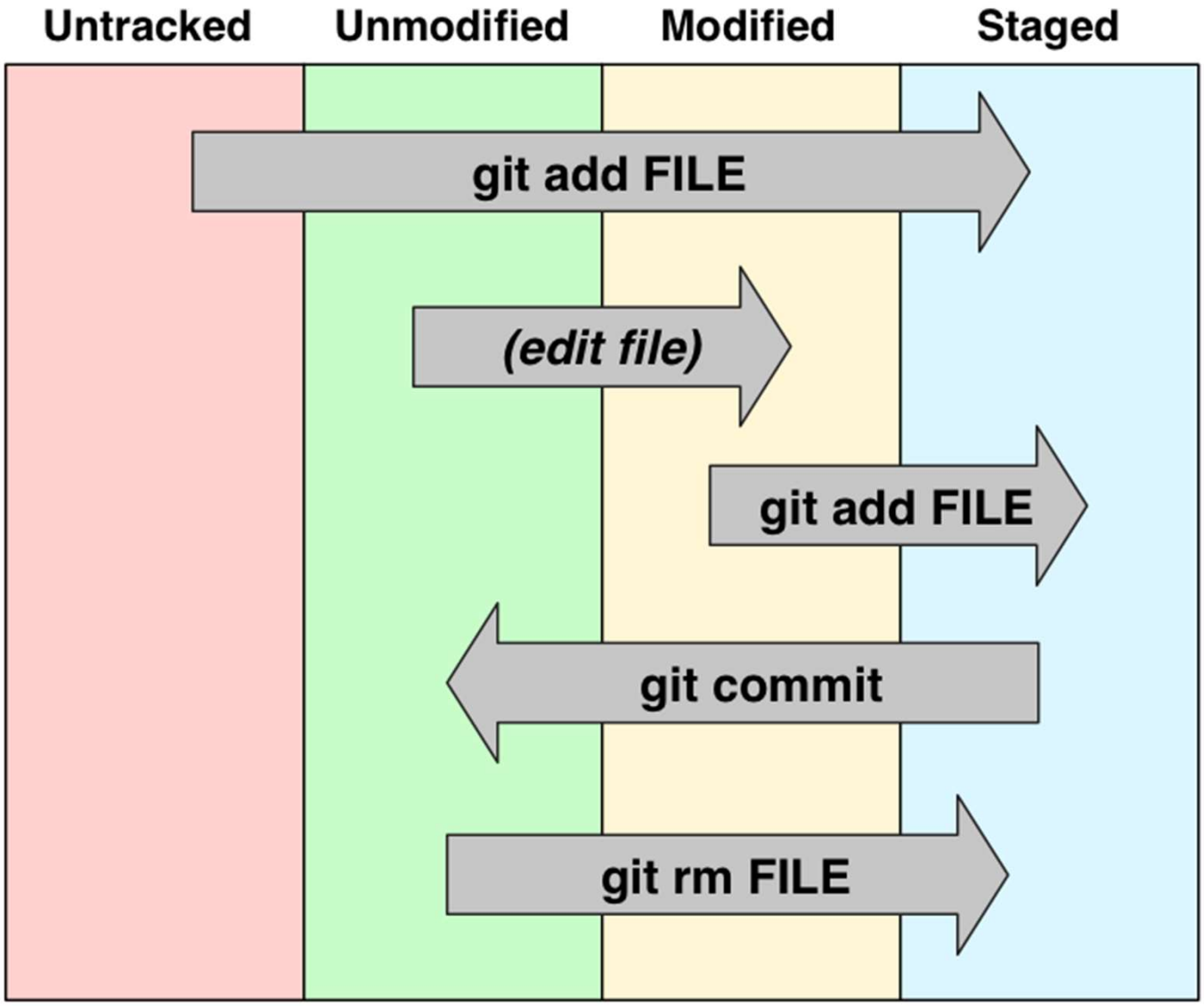
Git	vs.	GitHub
• A version control system		• A server that **hosts** Git repositories
• Works locally and remotely		• Works solely as a cloud-based service
• Is open-source		• Owned by Microsoft

Working Directory ≠ Git Repository

- Working directory is your **project folder**
- Git Repository is a **hidden file** stored inside your working directory
- The Git Repository contains the **staging area** and your **commit history/local repository**

Git Pipeline




Untracked
Unmodified
Modified
Staged
git add FILE
(edit file)
git add FILE
git commit
git rm FILE

Commit Messages

- Every commit MUST have an attached message/description
- This helps others identify what changes have been made and why

- [filename added/changed ; what change ; why]

BAD:

```
git commit -m 'add user.rb'
```

GOOD:

```
git commit -m 'create user model to
  store user session information'
```

Poll #1

T/F – Both untracked and modified files must be **added** to the staging area

Where are pending changes kept prior to committing them?

T/F - GitHub is a type of VCS

Summary

- Untracked or modified files are first **added** to the **staging area**
- Changes that have been staged can then be **committed** to the **local repository**
- Commits can then be used to **revert** to previous instances of the project or simply to **keep track** of what changes have been made

Part II – Basic Git Features

Activity #0 – Initialize a Git Repository

Activity #0 – Initialize a Git Repository

1. Create a new Git Repository (File → new repo OR "create a new repository on your hard drive")

2. Open the generated project folder, what do you see?

Activity #1 – Create and track a Text File with Git

Activity #1 – Create and track a Text File with Git

1. Create a text file (or code script) in the project folder

2. Open the GitHub Desktop App and add + commit the file to track it.

3. Look at your commit history to see your commit!

4. Modify the file and commit the changes. Try and make informative commit messages!

Activity #2 – Revert a file to a previous commit

Activity #2 – Revert a file to a previous commit

1. Make an unwanted change to your file
2. Use your commit history to revert to the previous commit

3. Make another unwanted change and commit it
4. Now use your commit history to revert the commit!

Poll #2

T/F – After initialisation, a project folder becomes a git repository

T/F – After making a commit, you cannot revert to a previous version of your repository

Is it possible for two commits to have the same ID?

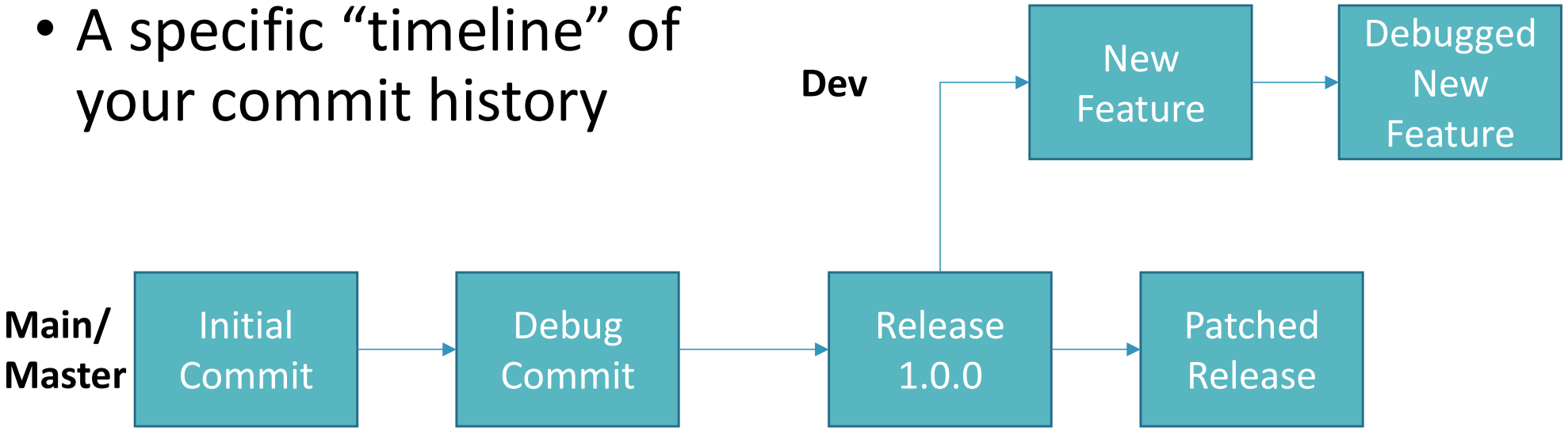
Part III – Advanced Git Features

What if you want to test out a new feature without affecting your current version of the code?

- Branching

What is a Branch?

- A specific “timeline” of your commit history



- Allows us to implement new features without harming viable releases

Activity #3 – Create a new branch

Activity #3 – Create a new branch

1. Click the “current branch” button and select “new branch”

2. Make changes to your file in the new branch and commit it

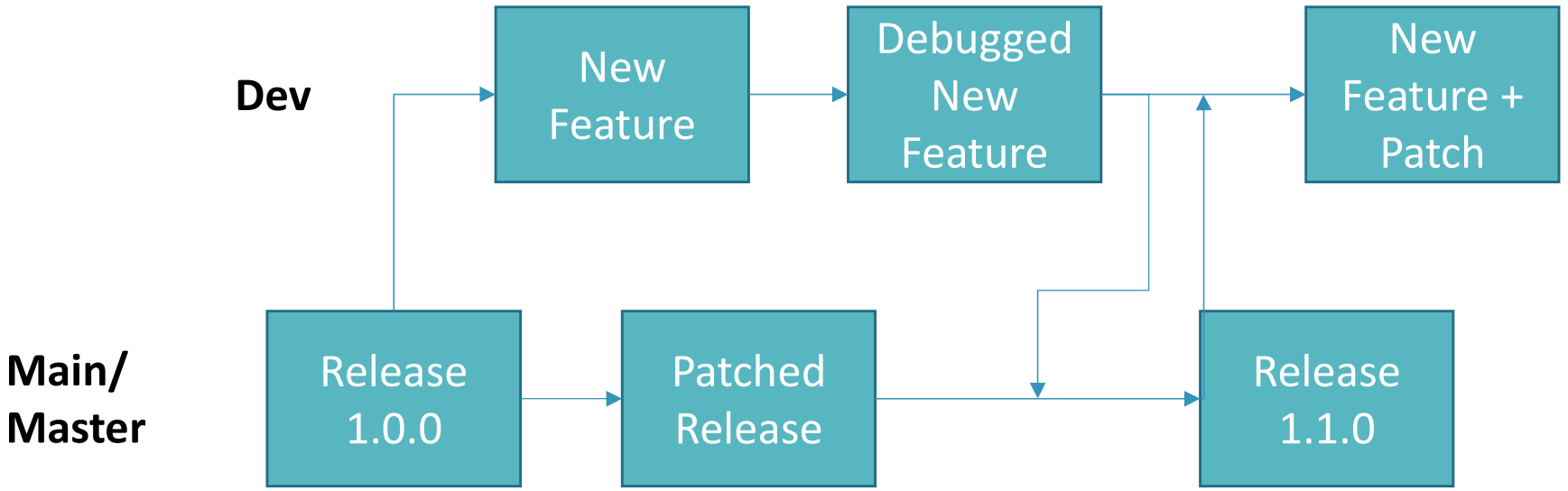
3. Switch back to the main branch and open the file, what do you notice?

Now that your test feature works, you might want to make it part of your official release

- Merge

What is a Merge?

Action that will “merge” the commits from one branch into another



Activity #4 – Merging Branches

Activity #4 – Merging Branches

1. Switch to the main branch

2. Open the branch menu

3. Select the “choose a branch to merge into main” button and select your second branch.

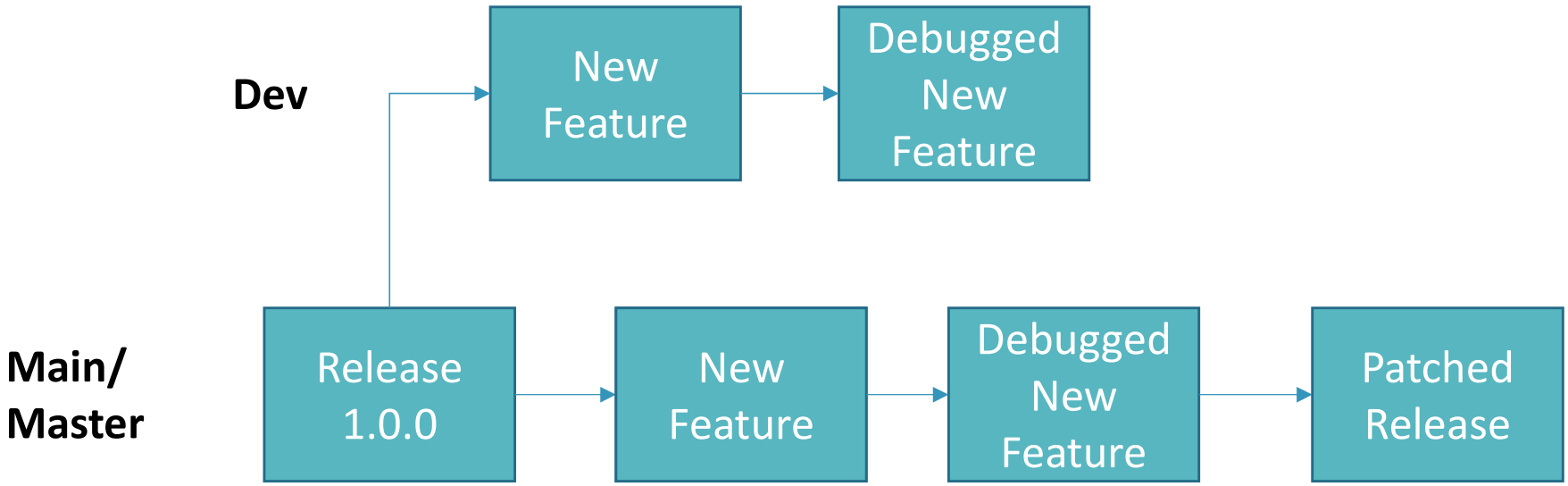
4. Compare the branches again, the file should now be identical in both!

Merging is not the only way to combine branches

- Rebasing

Rebasing

Instead of merging the other branch's changes as a new commit, rebasing rewrites the target branch's commit history to add the commits directly



Conflicts

If branches have competing commits, git will raise a **conflict** and not be able to merge/rebase

Can occur if:

- Both branches edit the same line
- Both added a file of the same name
- One branch edited a file, the other deleted it

Activity #5 – Resolving Conflicts

Activity #5 – Resolving Conflicts

1. In both branches, make a change on the **same line** and commit it

2. Try and merge your second branch into the main

3. When the merge conflict occurs, open the text file and resolve the issue

4. Try to merge again

5. Look at your commit history, how is your resolved merge shown in the commit?

Poll #3

What is the purpose of a merge/rebase?

T/F – Merging branches with changes in the same file will ALWAYS create a merge conflict

How could we prevent conflicts from occurring?

Summary

- Branches can be used to develop new features without affecting a previous, viable version of your code
- Merges or Rebases can be used to combine the commits of two different branches
- Conflicts prevent branches from being combined

Part IV – Hosting Projects on GitHub

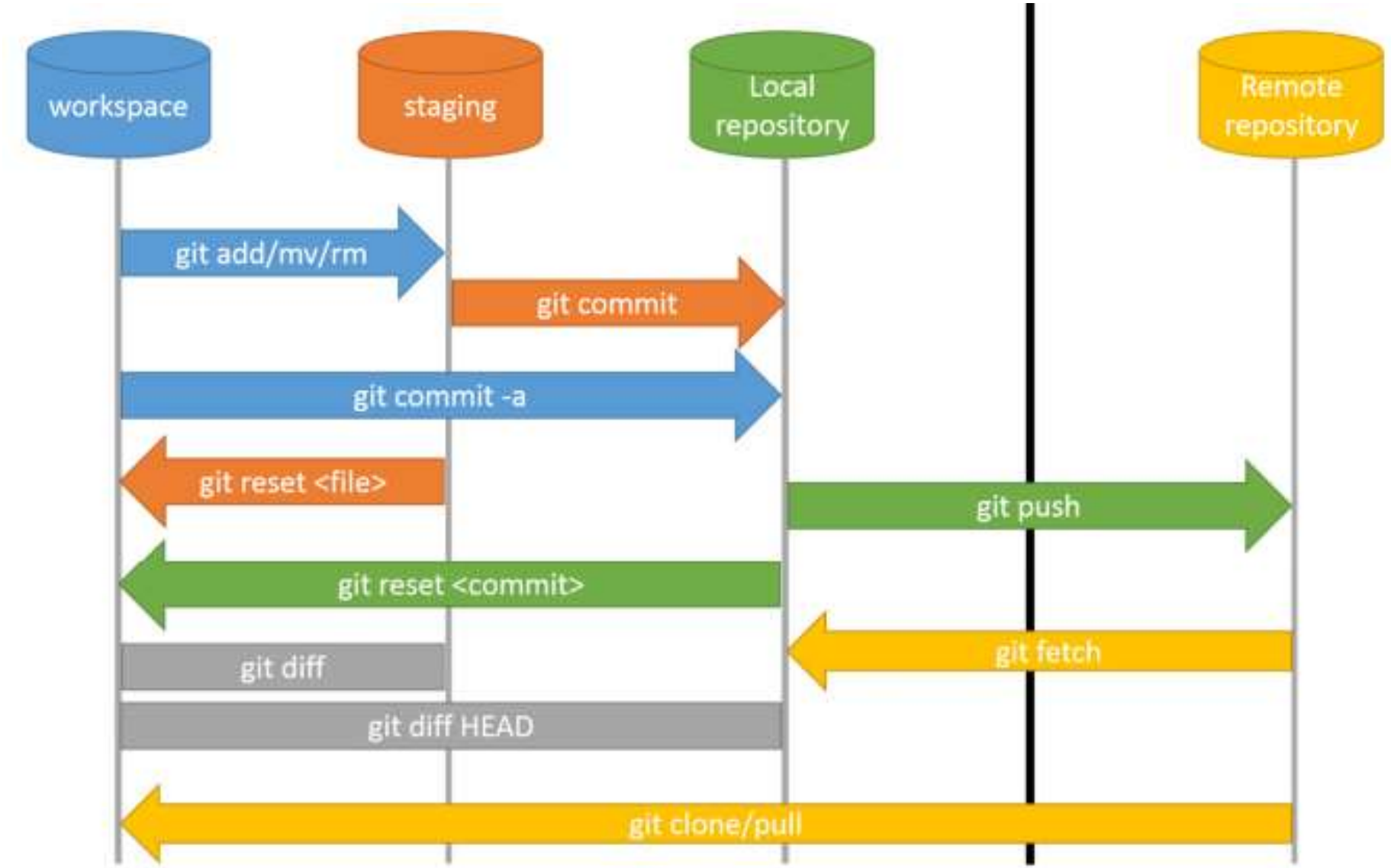
GitHub Interface

Why use remote repositories like GitHub?

- Backup for your code
  - Can access the repository from anywhere

- Collaborations
  - Easy to keep track of changes made by you and collaborators

- Open Science
  - Easy to access, download and use others' code
  - Easy to discuss with **users** of your code

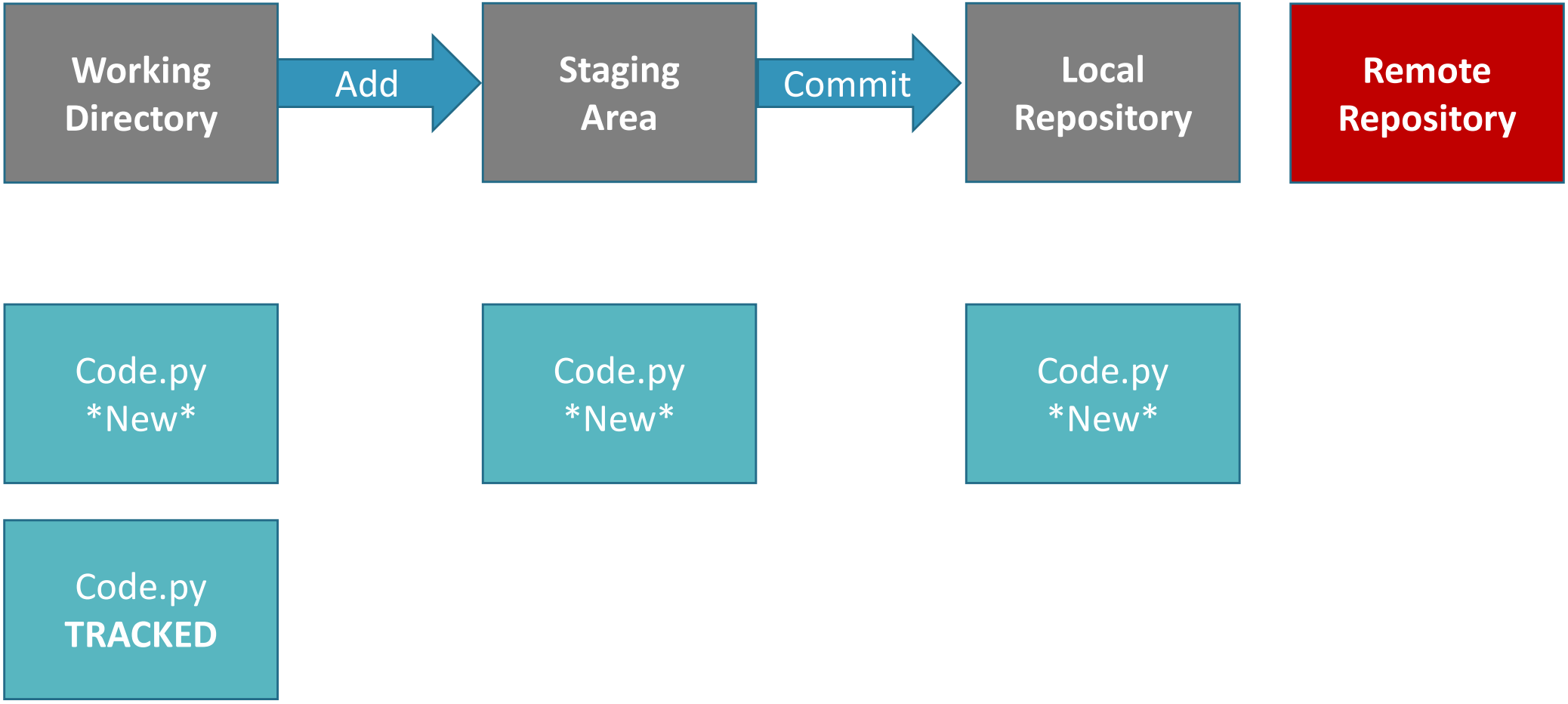

workspace
staging
Local repository
Remote repository
git add/mv/rm
git commit
git commit -a
git reset <file>
git push
git reset <commit>
git fetch
git diff
git diff HEAD
git clone/pull

Pushing and Pulling

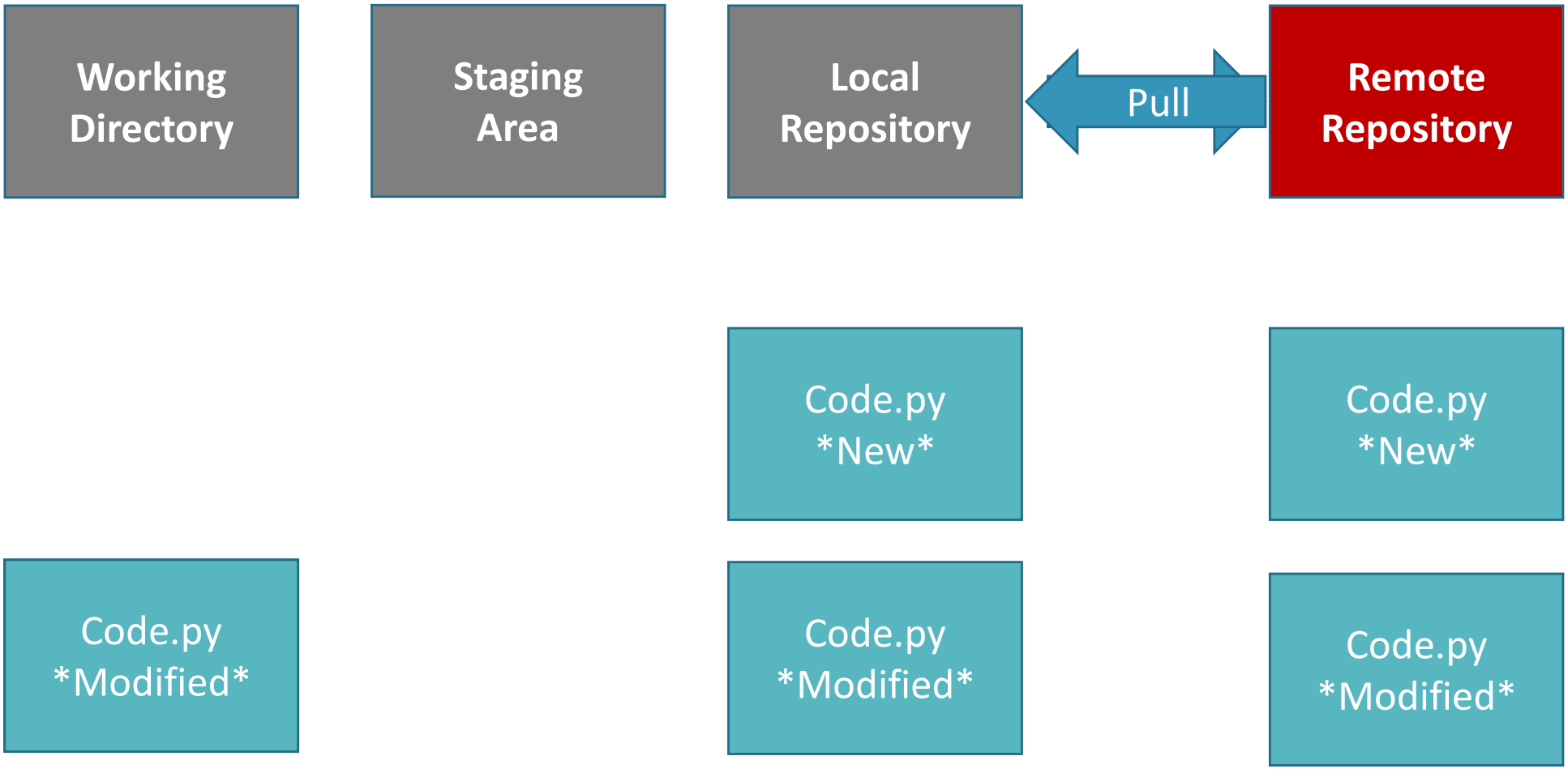
- **Push:** Updates your **remote** repository with recent commits from your **local** repository

- **Pull:** Updates your **local** repository with recent commits from your **remote repository**

Local-Remote Pipeline



Local-Remote Pipeline



Activity #6 – Pushing commits to GitHub

Activity #6 – Pushing commits to GitHub

1. Click the “Publish Repository” button

2. Now open your GitHub account in the browser and find your repository. See if your commit history is the same as in your local repo.

Activity #7 – Pulling Commits from GitHub

Activity #7 – Pulling Commits from GitHub

1. From the GitHub website, open your file and select the edit option

2. Make a change to the file and commit it.

3. Return to the desktop app and do Repository -> Pull

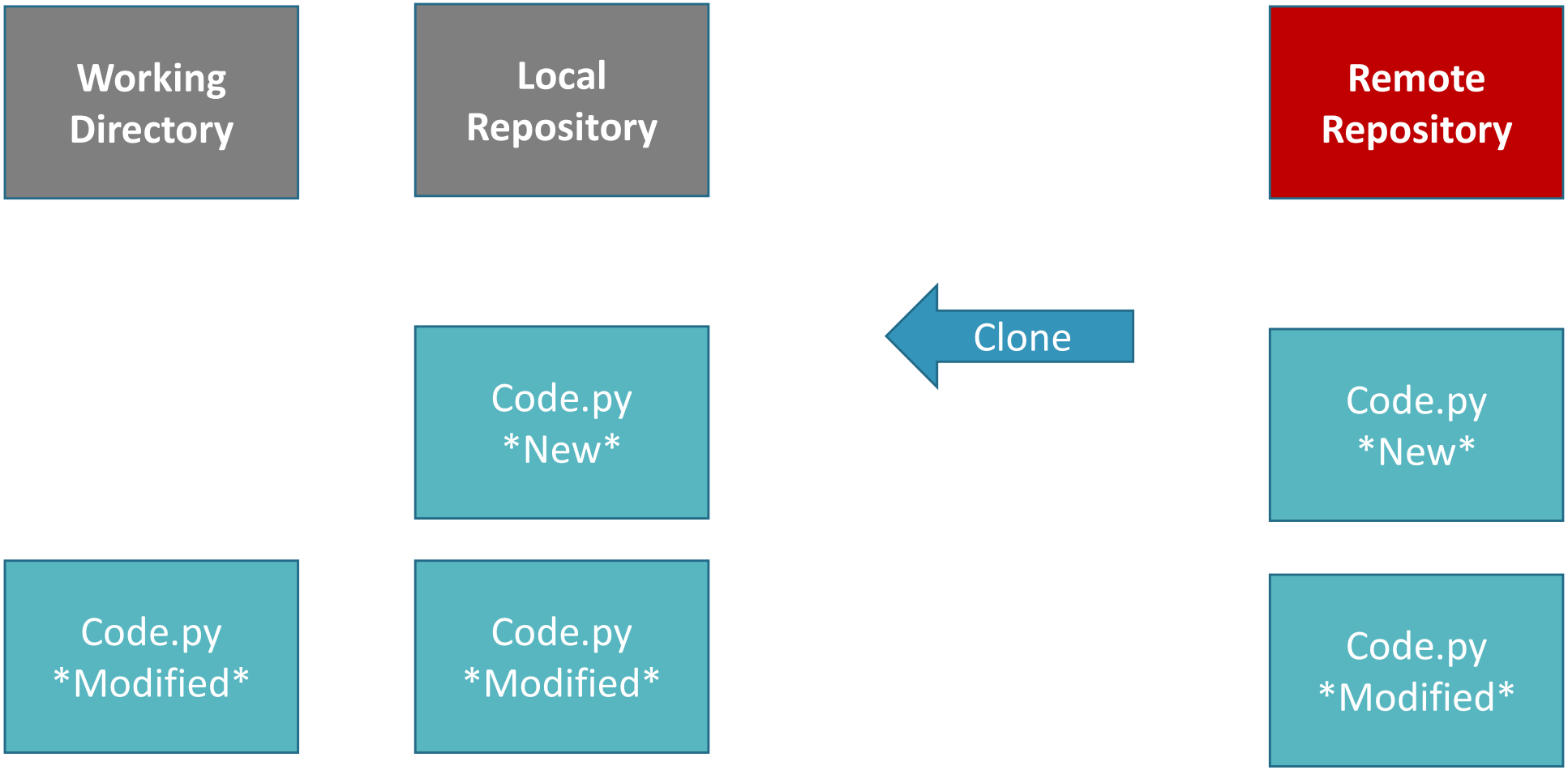
4. Look at your commit history and your file, what do you see?

Create a local copy of a GitHub Repository

- "Cloning"
- Usually done to **use** published code
  - Reproduce results
  - Use code on a dataset of interest

- Creates a **direct link** to the original repository
  - Anything changes you **push** will affect the original repository
  - Any changes in the original repo can be **pulled** into your clone

Cloning a Repository



Activity #8 – Clone a Repository

- Find a GitHub repo that interests you and clone it!

Activity #8 – Clone a Repository

1. Go to the GitHub website and search for a repo that interests you.

2. Copy the repo's URL

3. In the Desktop App, do File → Clone a repository and paste the URL.

4. Open the cloned repo and explore!

Contributing to projects

- We saw earlier that we can create merge conflicts
- How can we regulate this when working in teams?

Forks & Pull Requests

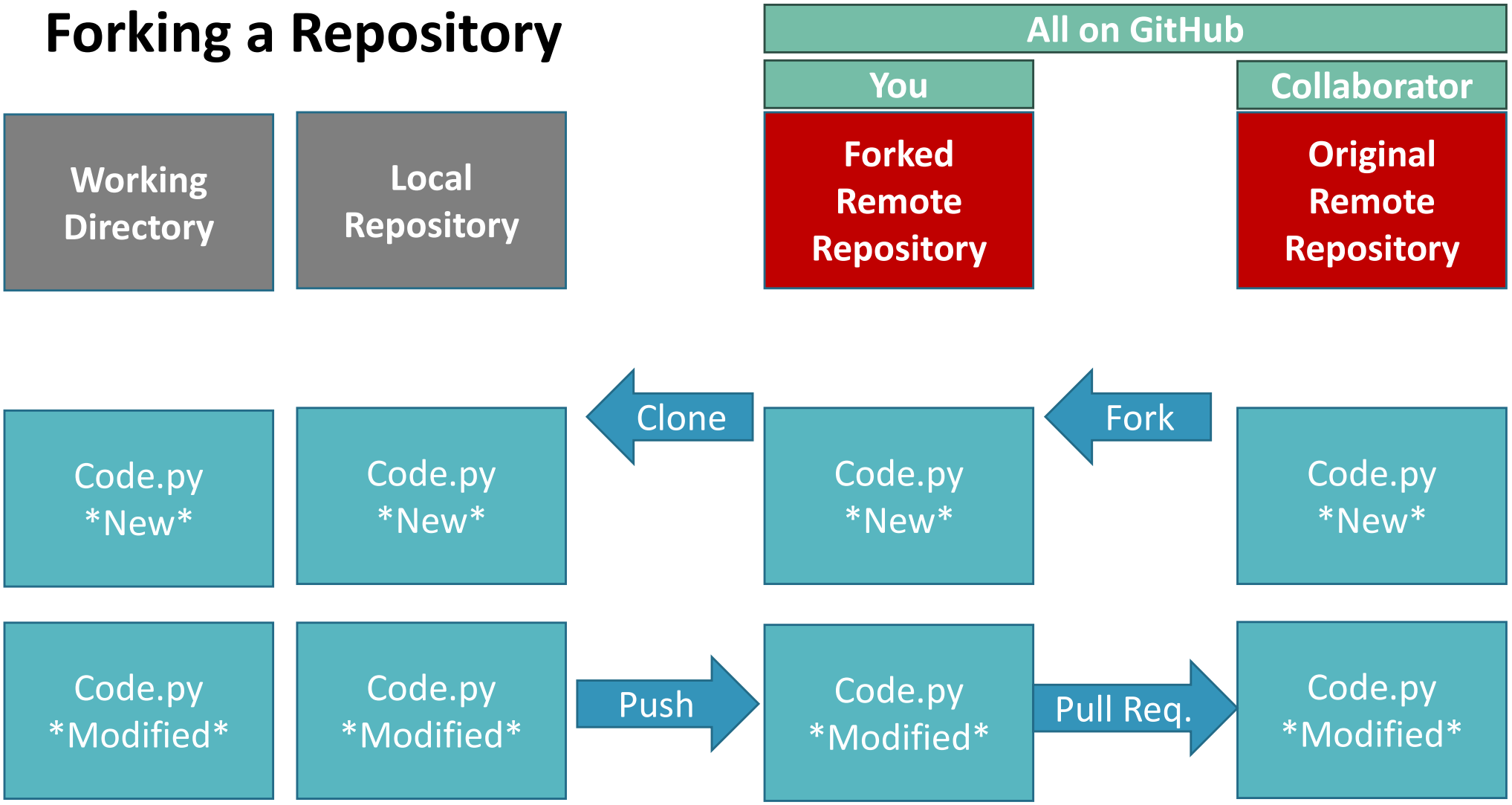
Create a remote copy of a GitHub Repository

- "Fork"

- Usually used to **contribute** to published code
  - Can then edit the forked repository locally
  - Can then **push** local changes to your forked copy
  - When your changes are done, you can make a **pull request** to add your changes to the original repo

- Much safer way to contribute to source code

Forking a Repository



Activity #9 – Fork a Repo and make a Pull Request

- Fork the repo I made for this course and add a text file to it. Then, make a pull request!

Activity #9 – Fork a Repo and make a Pull Request

1. Open the link in chat to access the repo I made.

2. Click the “fork” button

3. Clone your forked copy to your computer and add a text file with one thing you have learned today. Commit the file.

4. Push the commit to your forked repo.

5. On the GitHub website. Look at your forked repo and select the “create Pull Request” button

Poll #4

What command lets you update your remote repo with local changes?

T/F - Cloning a repository is the best way to contribute new features

T/F – You will always be able to push changes from a cloned repo to the original

What is the advantage of using a Pull Request prior to merging changes from a contributor?

Summary

- GitHub provides a framework to store and document code
- Push/Pull commands allow you to transfer commits between local and remote repositories
- Forks/Pull Requests help manage code contributions

Important: Using GitHub is **NOT** a replacement for good communication:

- Ensures everyone is using the latest version of a file
- Helps keep track of what changes were made and by who

Good Practices:

- Have certain user(s) in charge of approving Pull Requests
- Set a standard format for commit messages

Part V – Group Activity

1. Split into teams based on preferred language (R, Python, Neither)

2. Made a base repository called Git_Collab
   - https://github.com/aosakwe/Git_Collab

3. Task – as a team. Using the base files in the repo, create a fork and collaborate to extend it using pull requests.

Part VI – Miscellaneous Features

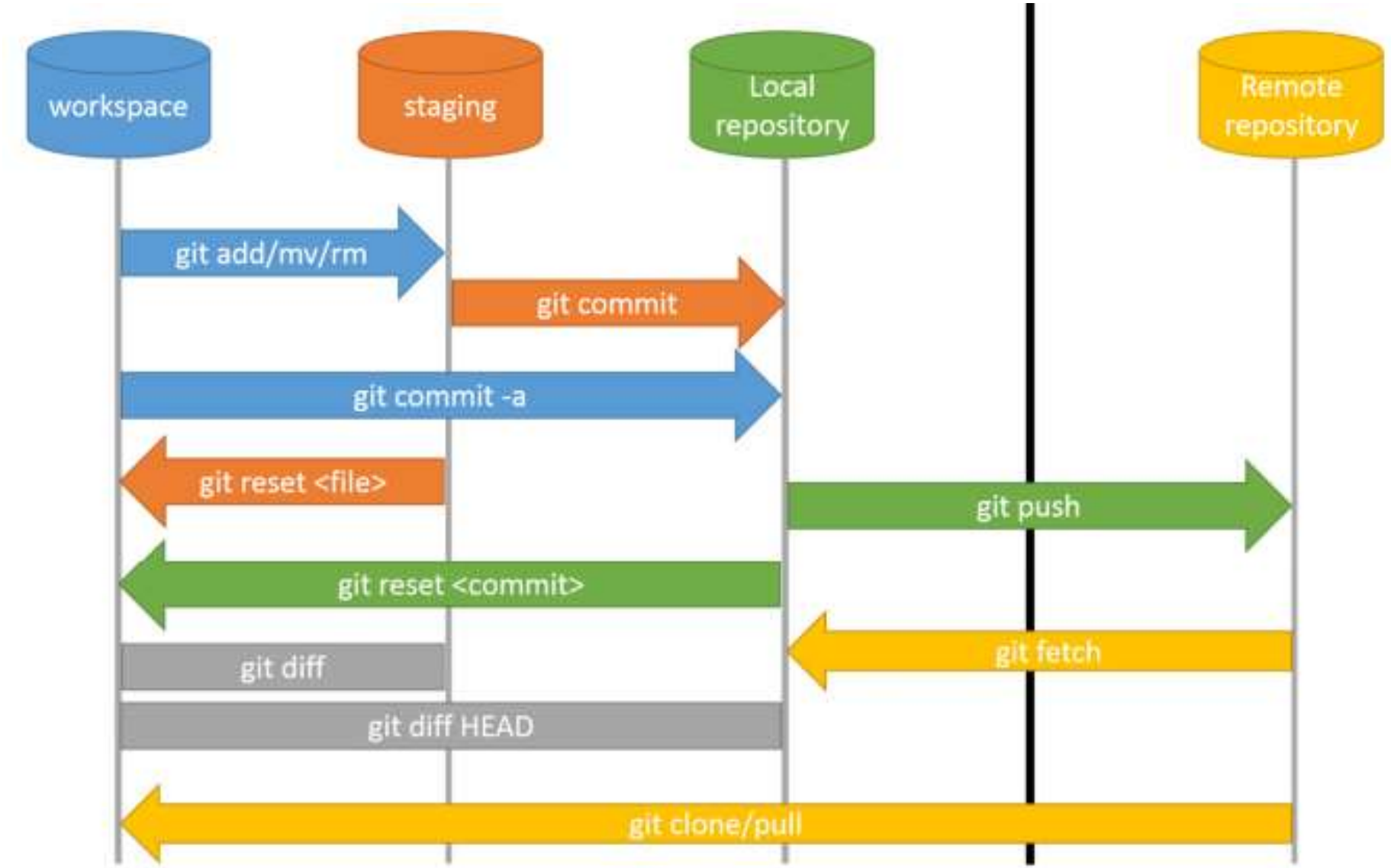
Using Git on the Command Line

Using the Command Line:

- Direct interaction with Git
- Allows you to use Git on an HPC server (e.g: ComputeCanada)
- Provides more tools than third-party software

But:

- Can have a steep learning curve depending on experience with UNIX


workspace
staging
Local repository
Remote repository
git add/mv/rm
git commit
git commit -a
git reset <file>
git push
git reset <commit>
git fetch
git diff
git diff HEAD
git clone/pull

Activity #10 – Initialize Repo with git commands

1. Open Command Line and switch to Desktop:
   cd Desktop/

2. Make a new directory: mkdir Git_Practice

3. Initialize git repository: git init

4. Check the status of your repository: git status

Activity #11 – Commit a new file with git commands

1. Create a new txt file in the repo with vim or through your text editor
2. From the command line, check your repo status: git status
   (do you see the new file?)

3. Stage the new file: git add file_name
4. Check your repo status: git status

5. Commit the file: git commit **–m** “Commit Message”
6. Check your repo status: git status
7. Check your Commit History: git log
8. Rename your branch: git branch –M main

Activity #12 – Create a new branch

1. Check your current branches: git branch
2. Create new branch: git branch new_branch_name
3. Check your current branches: git branch
4. Switch to new branch: git checkout new_branch_name
5. Edit your file and commit it
6. Check your commit history: git log

Activity #13.1 – Revert a Modification/Commit

1. Make a change to a file
2. Check the repo status: git status
3. Restore the previous version of file: git restore file_name

4. Make the change again, but stage it this time: git add file_name
5. Unstage the file: git reset file_name

Activity #13.2 – Revert a Modification/Commit

1. Make a change and commit it
    1. git add file_name
    2. git commit –m “commit message”

2. Get the commit ID from your commit history: git log

3. Revert your current commit: git revert commit_id

4. Check your file: have the changes been reverted?

Activity #14.1 – Push to Remote Repository

1. On the GitHub website, click on your profile in the top right corner and select “settings”
2. Select “Developer settings” at the bottom of the page

3. Under the “Personal access tokens” menu, click generate new token.
4. Select the “repo” scope and generate token.

5. SAVE the token somewhere (or you will have to make a new one)

Activity #14.2 – Push to Remote Repository

1. On the GitHub website, create a new repository
    1. Do NOT create a README file
    2. Copy paste the repo's URL

2. In the command line, link your local and remote repo:
   git remote add origin URL

3. Push your main branch: git push -u origin main
4. Check your remote repo: are the files there?

GitHub Pages

GitHub Actions

Part VII – Conclusion

What have we learned?

- What VCS, Git and GitHub are
- Why they are so useful
- How to undertake basic tasks in a local and remote repository
- How services like GitHub improve code collaboration and Open Science

What next?

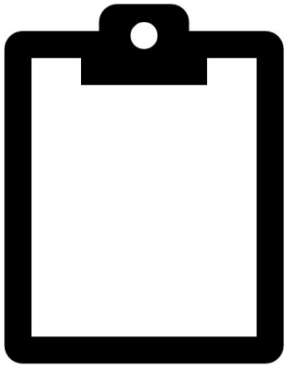
- Practice makes perfect!
  - Track your hobby projects
  - Track your research
  - Use open-access code or software from publications
  - Consider trying Git on the command line

Thank you for attending!

1

Scan the QR code to confirm you attended today's workshop.

2



Fill out the feedback survey in the next 72h.

3

Get recognition for this workshop on your co-curricular record.

Image Sources

https://betterexplained.com/articles/a-visual-guide-to-version-control/

https://github.com/qw3rtman/git-fire

https://walkingrandomly.com/?p=6653