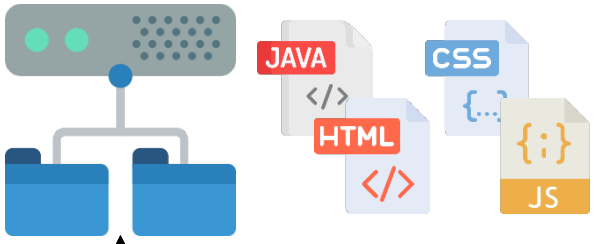
Introduction to Git

Distributed Working Environment

What could possible go wrong?

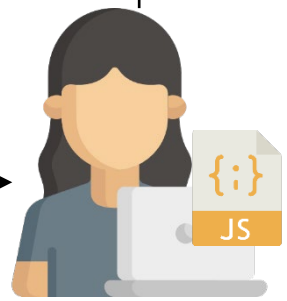
Shared Storage with source code



Copy and update source files shared drive

Fred

Share files via email and USB drive



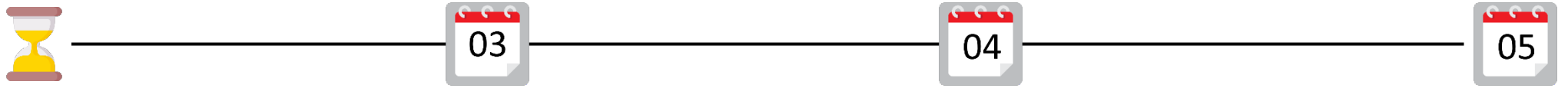
Alice

Barney

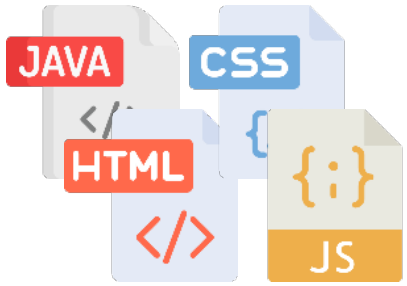
What is Git?

- Git is a version control system, used to keep track of the changes in your source code
- Distributed, allows many developers to collaborate and work on the same code base
  - Provide a way to manage conflicts eg. if 2 person edited the same file, whose work has precedence?
  - Merge changes from different sources
- Command line as well as GUI
  - git - command line tools works with all git repository
  - gh - command line tool for GitHub
    - Interacts with GitHub, create PR, clone repository, etc.
  - GitHub desktop - official GitHub GUI, works with any repo
    - Alternatives - GitKraken, SourceTree

What is Git?

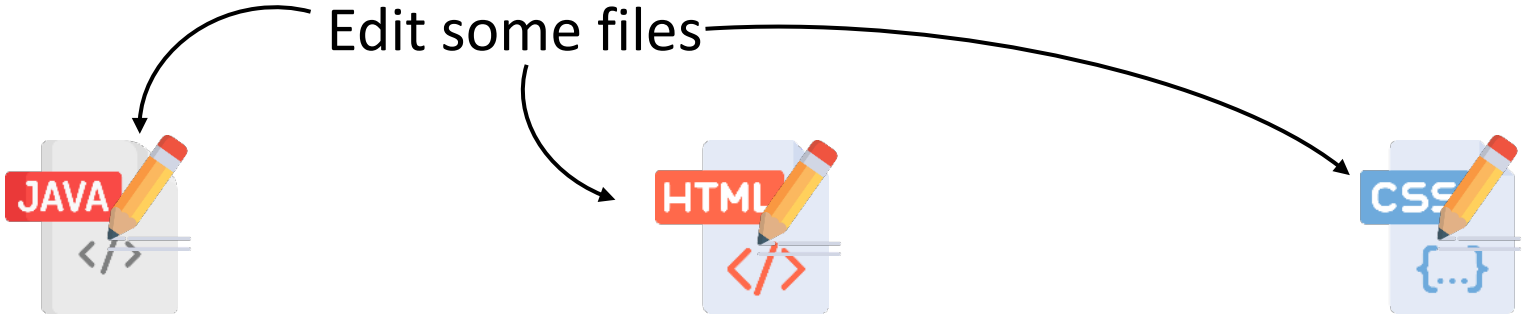


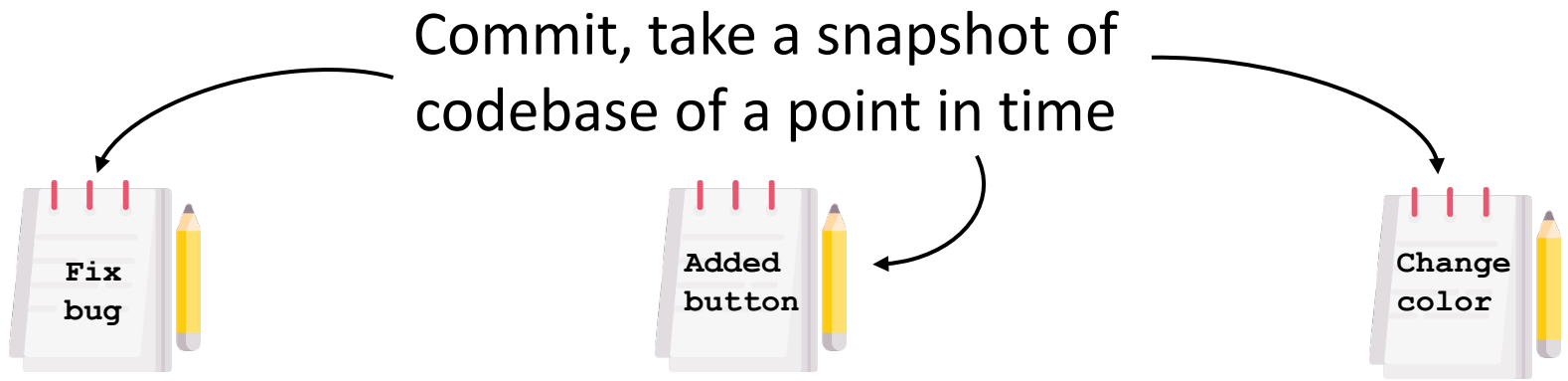
Timeline



Source code

Git repository





Working on Local Machine

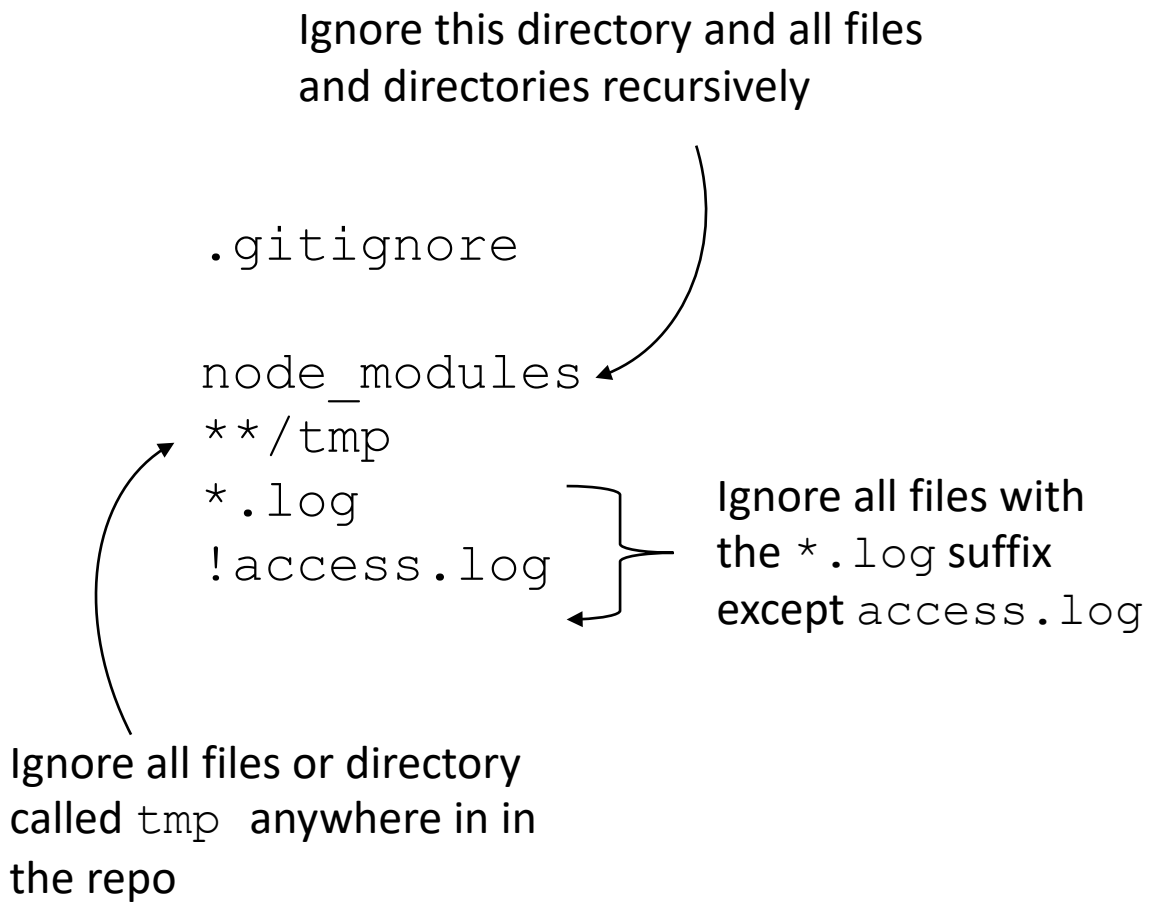
- Git initialize a local repository
  - Do once at the start of each project

```
mkdir myapp
cd myapp
git init
```

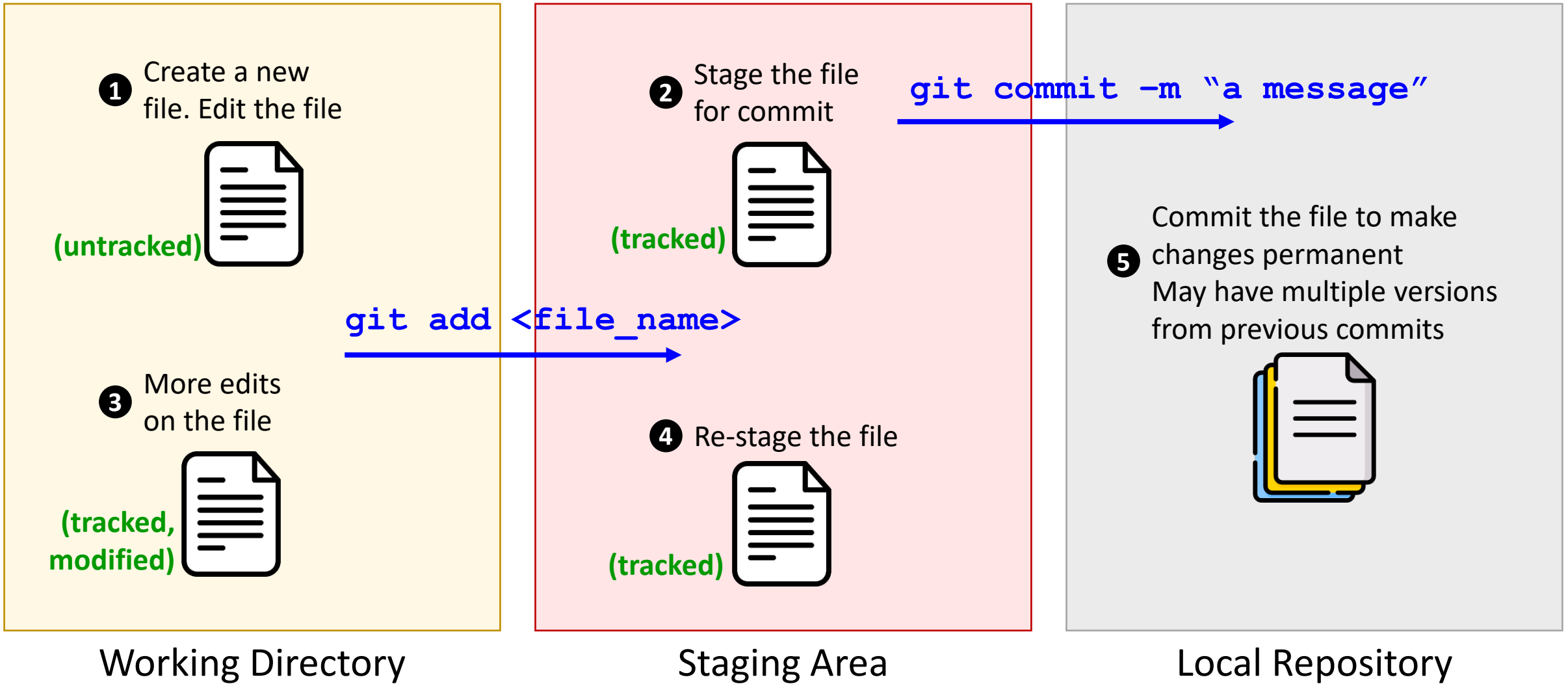
  - Places the contents of the directory under git's control
- Git uses 3 different areas to manage files
  - Working directory
  - Staging area
  - Local repository

Ignoring Files

- Some files should not be committed to repositories
  - Configuration files with secrets eg. password, security keys
  - Software development artifacts eg. `.class` files, `node_modules`,
  - Logs and temporary files eg. from editor, static assets
- List the files or directories for git in `.gitignore`
  - One file or directory per line



Life Cycle of a File



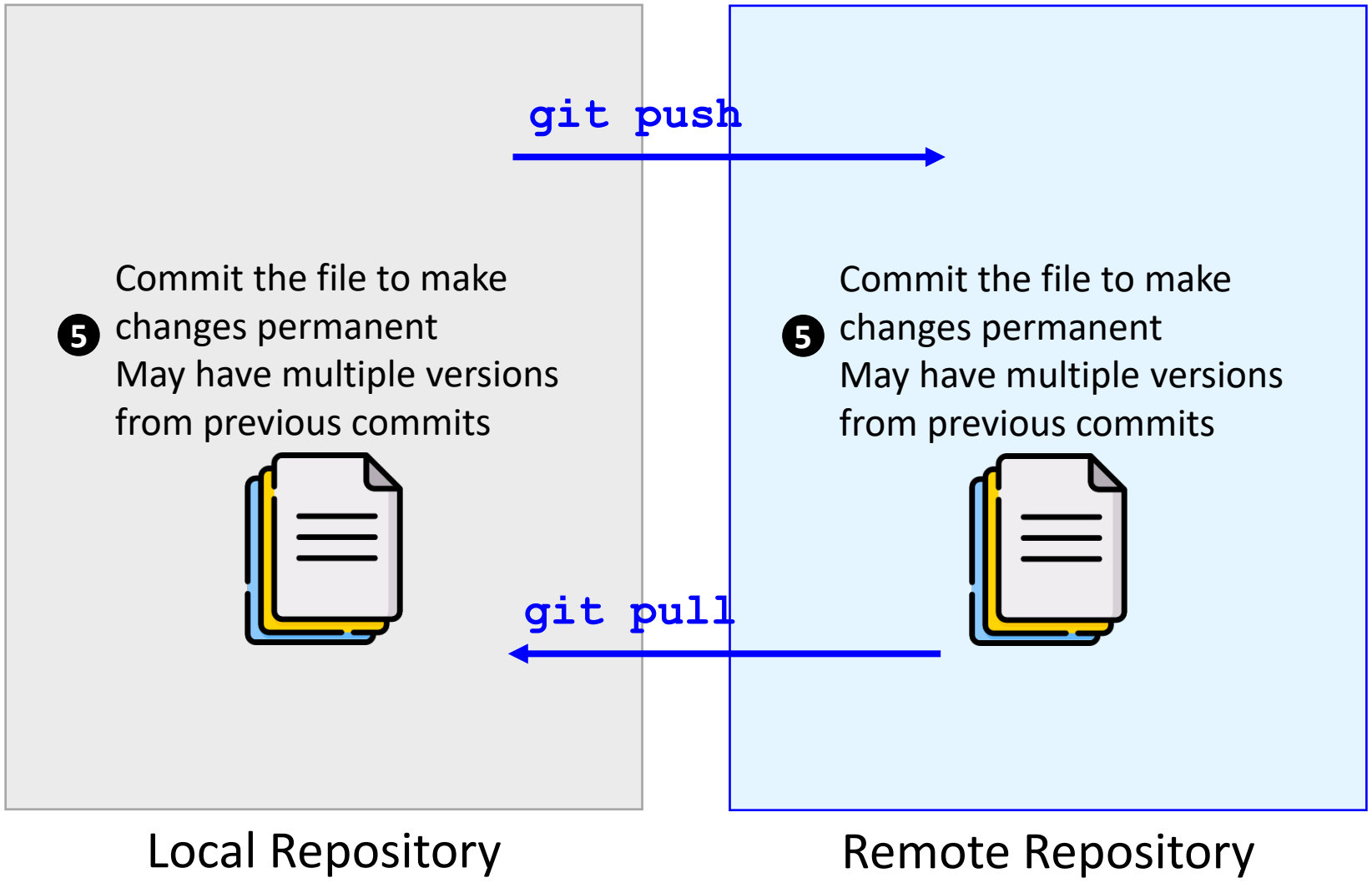
Common Git Commands

- Create a repository – `git init`
- Add a file to the staging area – `git add <file name>`
  - Either a new file or a tracked modified file
- Commit all files in staging area to repository – `git commit –m <commit message>`
- Look at the status of the files in the repository – `git status`
- List the commit messages – `git log`

What is Github, Gitlab, Bitbucket, etc?

- Git hosting services
- Other services besides hosting git repositories
  - Issue tracking
  - Automated CI/CD
  - Host static web site
  - Collaboration

Working with Remote Repository



Working with Remote Repositories

- Add a remote repository to your local repo

```
git remote add origin git@github.com:fred/myrepo
```

  - Github no longer uses username/password, have to generate SSH key pair
- Push your work to the remote repository

```
git push origin master
```

- Pull (refresh) your local repo with the code from remote

```
git pull origin master
```

Git Workflow

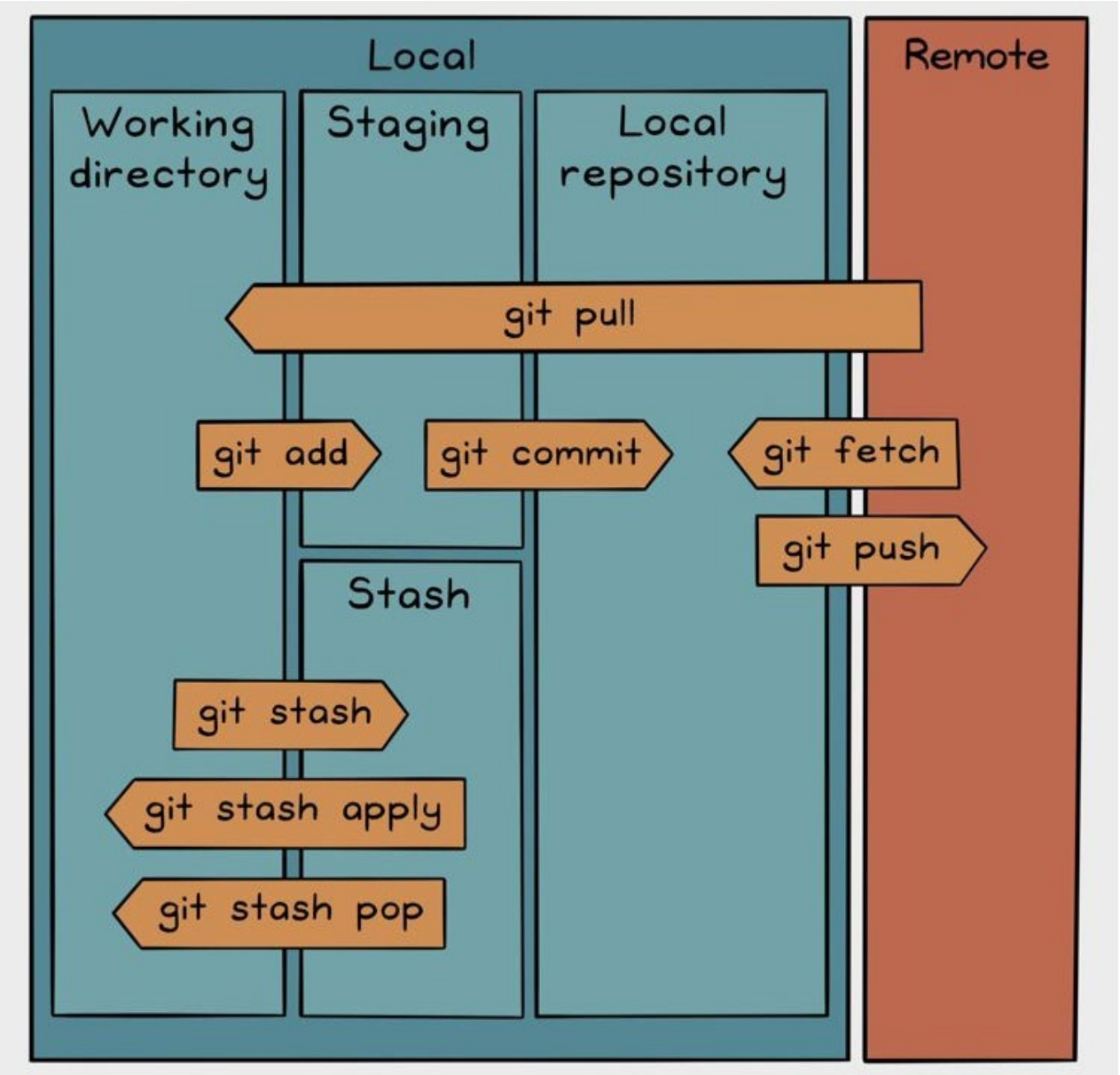
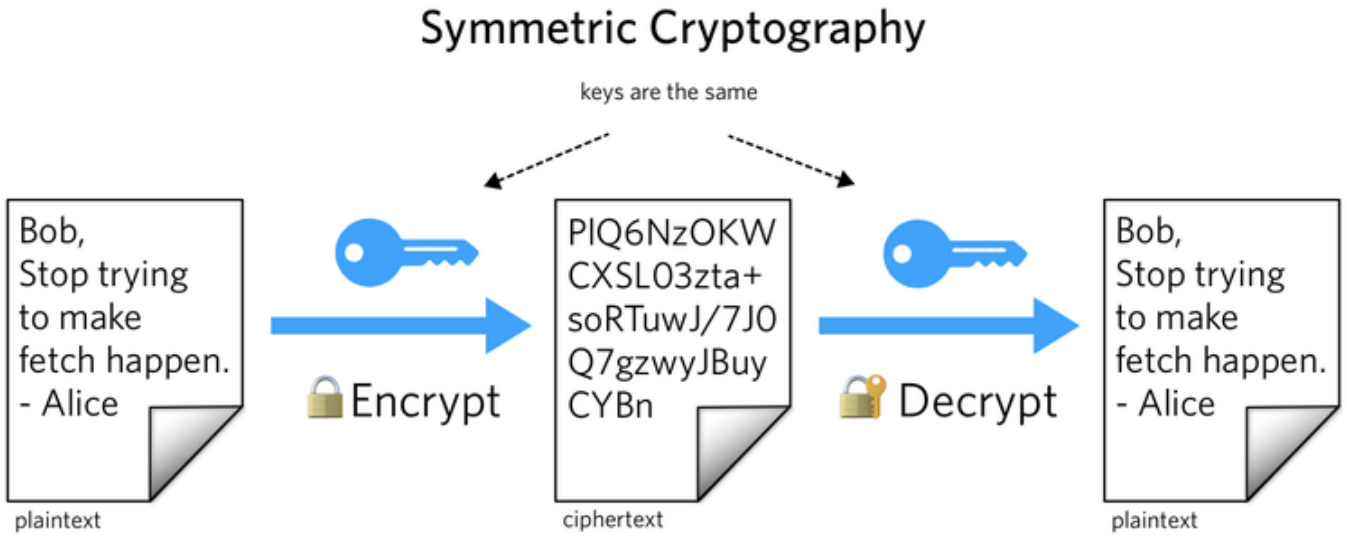


Image from https://twitter.com/nikkisiapno/status/1593882400983072769

Authentication

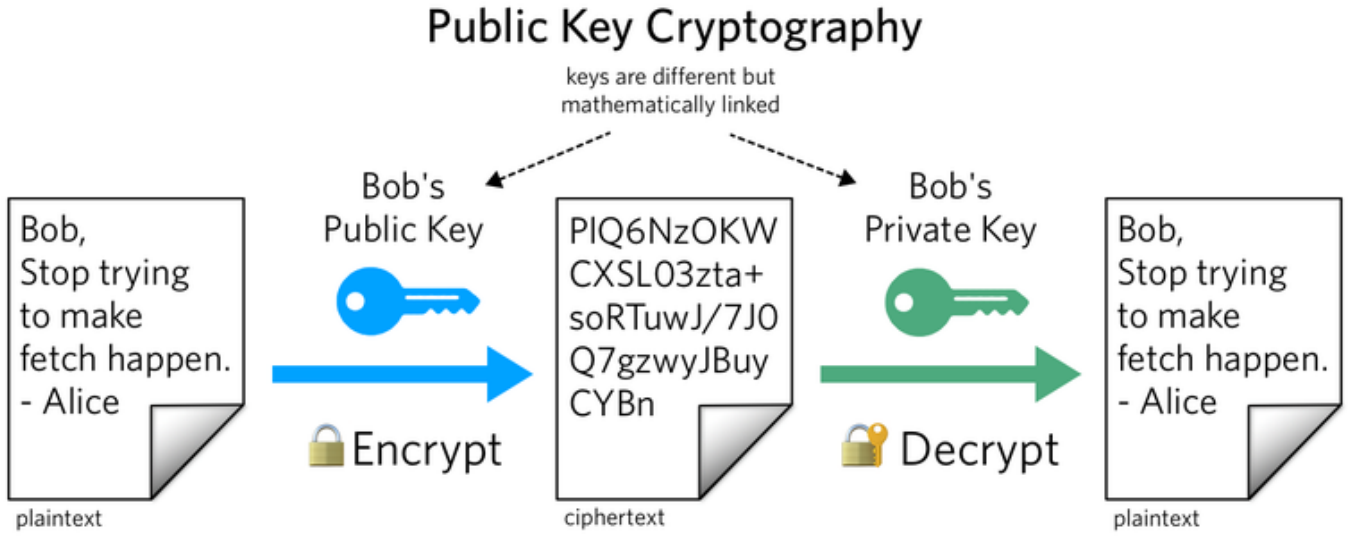
- Github has deprecated using username / password for pushes
  - Insecure
- Require you to setup SSH key pair
- SSH keys are used to communicate securely with remote servers
  - By creating an encrypted tunnel between the client and the server
- SSH keys are asymmetric keys
  - 2 different cryptographic keys that are related

Asymmetric vs Symmetric Cryptographic Keys



Same key is used to encrypt and decrypt

Different keys are used to encrypt and decrypt

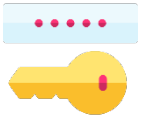


Images from https://www.twilio.com/blog/what-is-public-key-cryptography

How Does SSH Work?

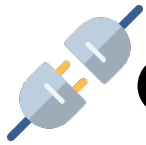
1 Generate a key pair

Public key



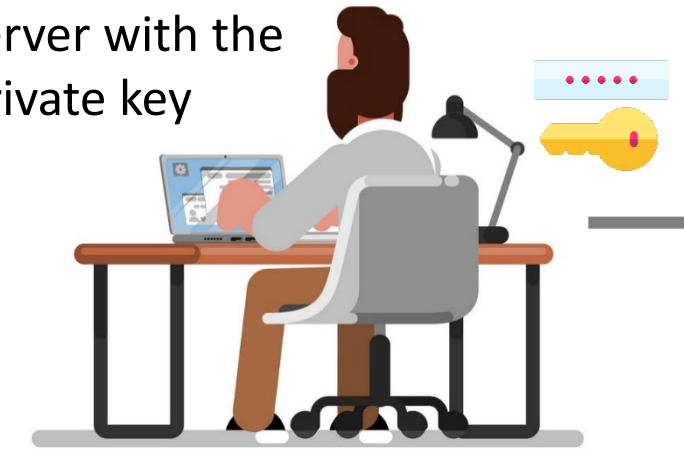
Private key

2 Distribute public key to the target server

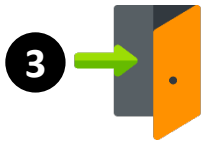


5 Establish connection after successful handshake
Agree on session key

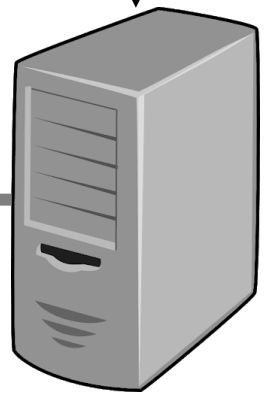
4 Connects to the server with the private key



User



3 Only allow access from the corresponding private key



Target

6 Encrypt/decrypt data with session key

SSH Keys Setup

- Generate a public/private key for SSH

```
ssh-keygen -t rsa -b 4096 -c "fred@gmail.com" -f fred_rsa4096
```

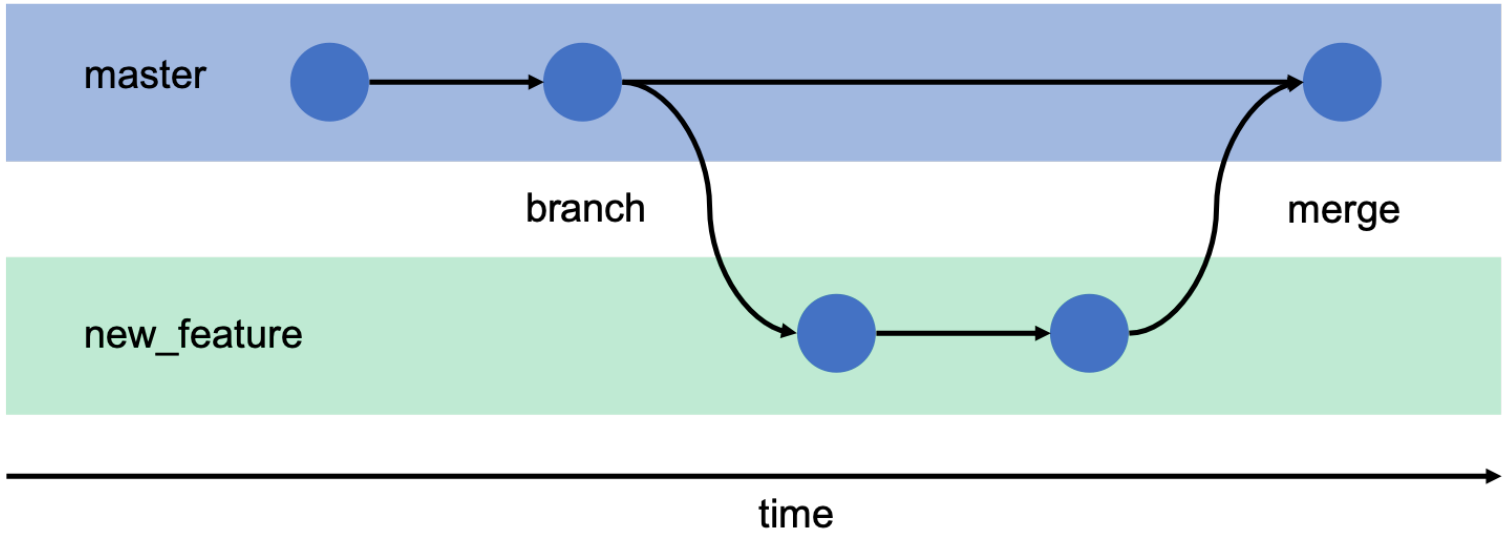
- Add the public key to your Git account
  - https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account
- Add Git configuration on your notebook

`$HOME/.ssh/config`

```
Host github.com
   Hostname ssh.github.com
   Port 443
   User git
   IdentityFile /path/to/public/key
   IdentitiesOnly yes
```

Branching

- An independent line of development, a branch of the main source code
- Git repo has a main branch, called master (or main)
  - A branch would be a branch off the main
- Branches can be merged back into the master or deleted entirely



Creating a Branch

- Create a branch

```
git branch new_feature
```

- Change branch

```
git checkout new_feature
```

- Display all the branches

```
git branch
```

- Branches are local, push a local branch to remote repository

```
git push origin new_feature
```

Merging

- Merging process merges a branch into another branch
  - Typically merge a feature branch into the master branch after prototyping a new feature or fix a bug
- Merging process
  - Commit all changes in feature branch (source branch)
  - Switch to the branch (receiving branch) you want feature branch to merge into
  - Merge feature branch into current branch
  - Delete feature branch

Merging Process

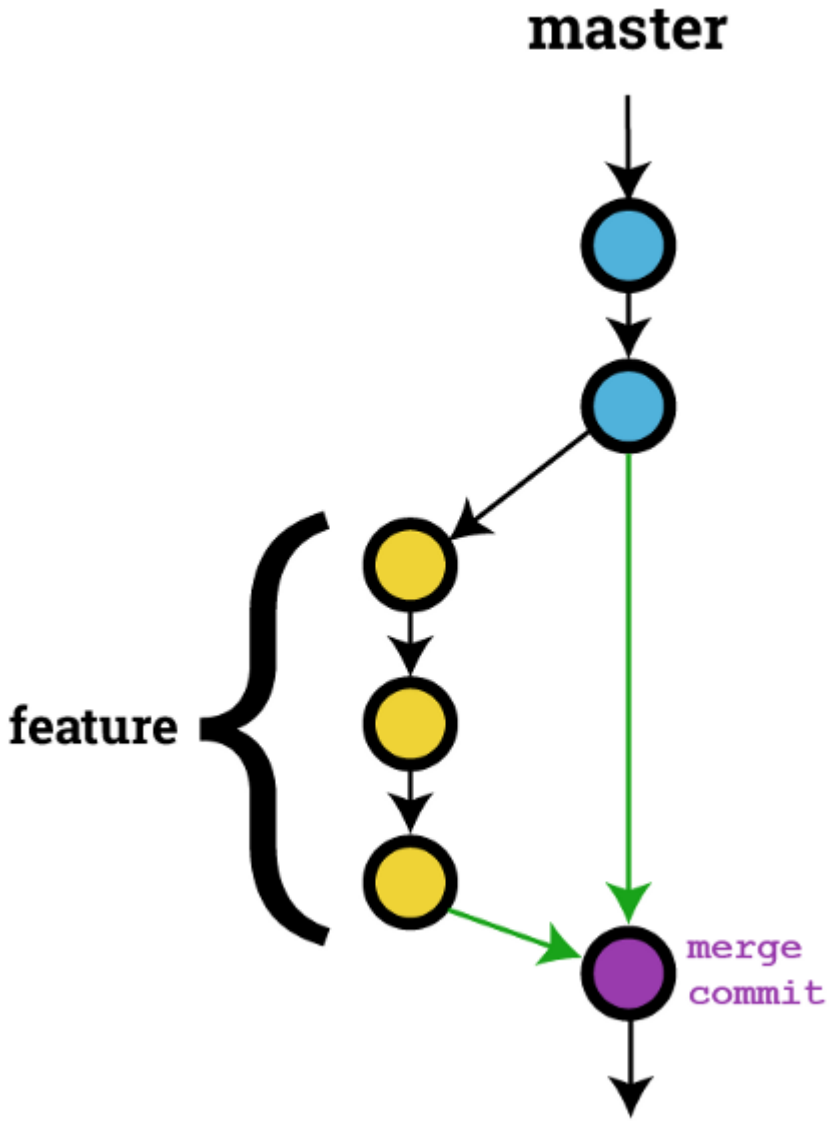
```
git commit -m "V1"
```

```
git checkout -b new_feature
```

Implement new features in `new_feature` branch

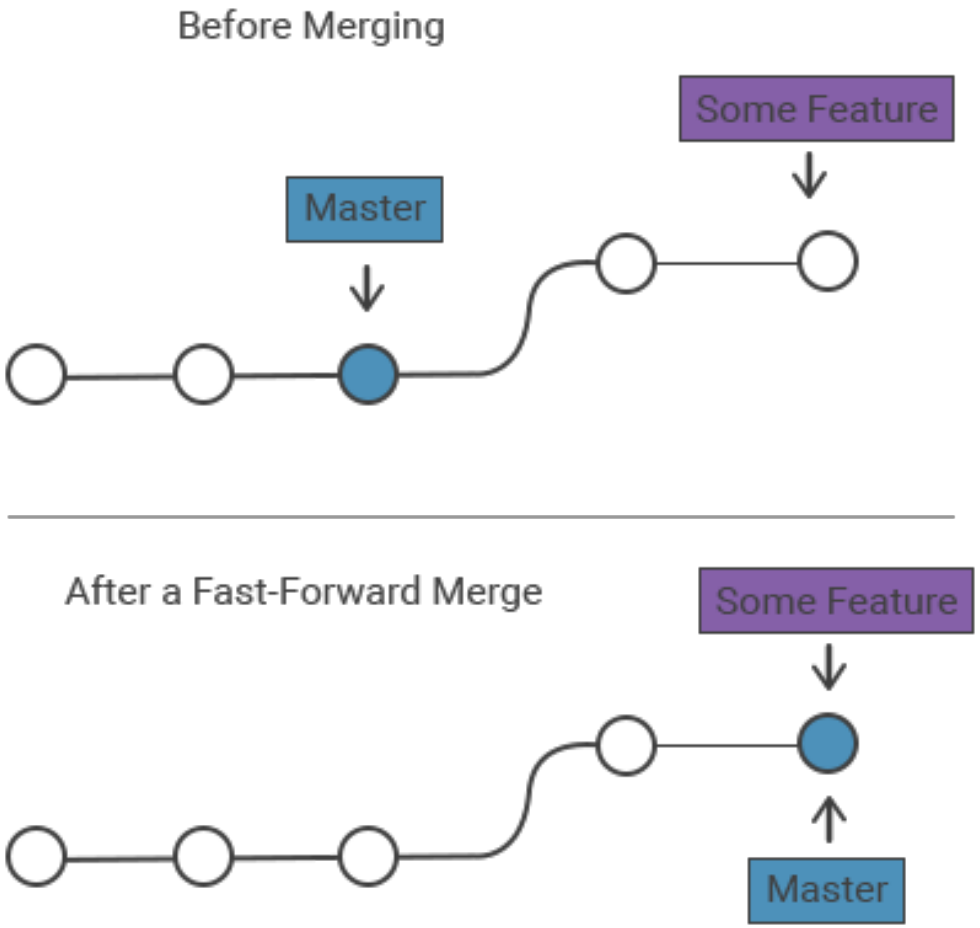
```
git add .
git commit -m "New feature completed"
```

```
git checkout master
git pull origin master
git merge new_feature
git branch -d new_feature
```

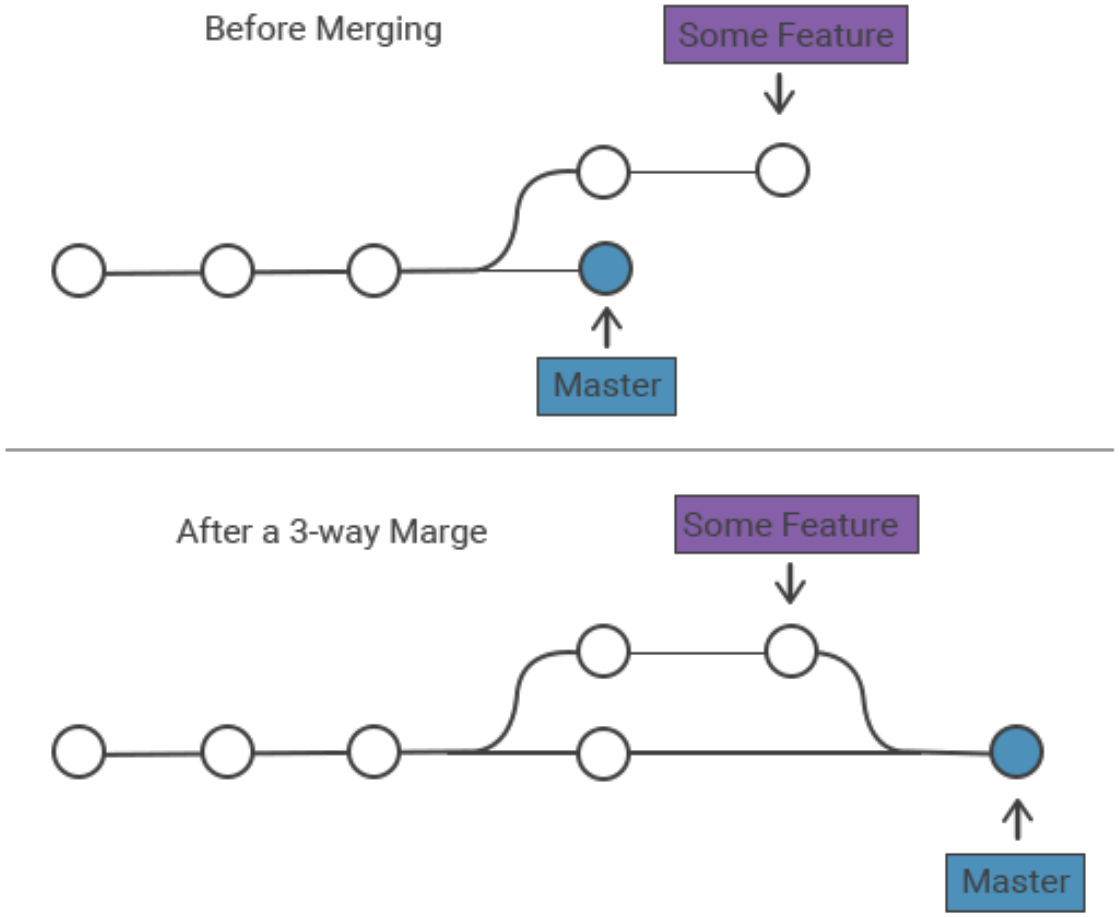


Fast Forward Merge

- No commits made to the master branch
- Move the pointer (HEAD) from master to the feature
- The commit from the feature branch becomes the latest commit in the master
- Note: using master branch as the receiver and feature as the source



Three Way Merge



- Occurs when the target branch has diverged from the source branch
- Git creates a new commit in the master that merges commits from both master and feature branch
- Need to resolve conflict before merge can be successful
  - Eg. same file is modified in both the branches
- Note: using master branch as the receiver and feature as the source

Resolving Conflicts

- Git conflict occurs when one or more files has been changed by 2 or more developers
- Git will stop the merge
  - Conflict files will be unstaged
  - Non conflict files will be staged
- Need to manually resolve the conflict
  - Stage and commit the conflict files

Delete

`<<<<<<<, =======, >>>>>>>`

after resolving the conflict

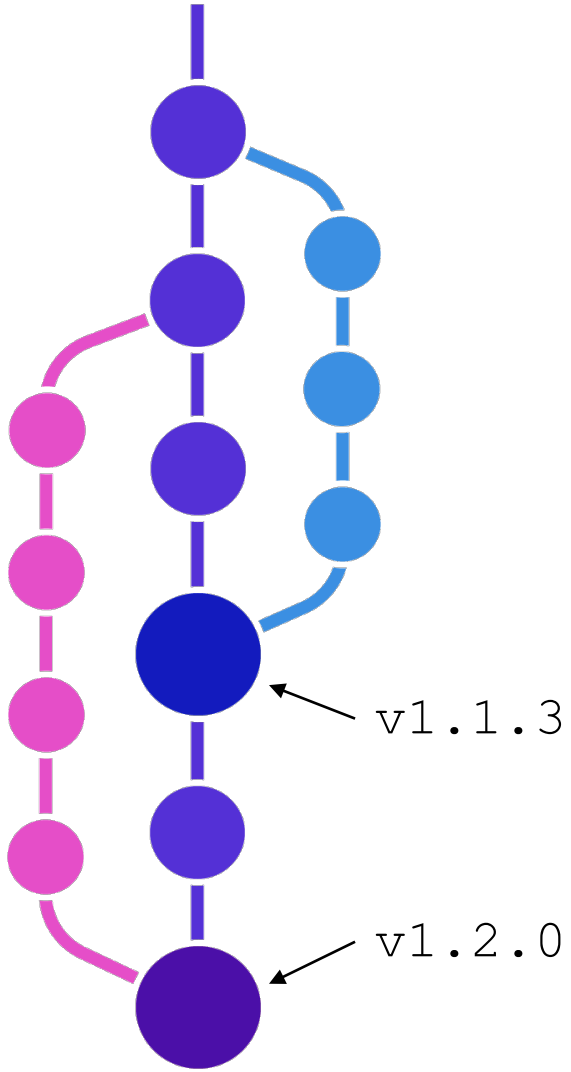
```
<<<<<<< HEAD
   console.log("position: ", ...)
=======
   console.info("position: ", ...)
>>>>>>> new_feature
```

Resolving Conflict Process

- Merge will stop when there are conflicts
  - Will list the conflicted files
- Conflict files with have HEAD and the branch name
  - Manually look thru the files and decide what you want to keep, or even rewrite the conflicted parts
- Once all the conflict have been resolved perform a commit

Tags

- Git commits provides a continuous history of commits
  - Commit message gives insight into what is done in each of the commits
- Certain commits may be milestones
  - Eg. Commit id abcd123 is version 1 of the software
- Tags creates a pointer with a symbolic name to a specific commit
  - Once a tag has been created, it will not change unless deleted



Tags

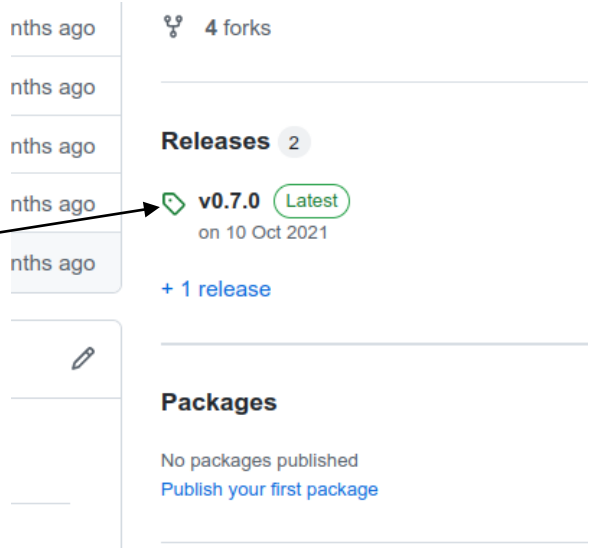
- Git annotated tags associate a symbolic name and a message to a commit

```
git tag -a v0.7.0 -m 'Implemented policy enforcement'
```

- When annotated tags are pushed to GitHub, GitHub will create a release from the annotated tag

```
git push origin v0.7.0
```

Latest/newest tag is displayed under Releases



Create Releases from Tags

- Release is a deployable iteration of the application
  - Includes release notes, changelog, the buildable source and/or prebuild binaries
- Releases are created from tags
  - When a release is created, it can be published

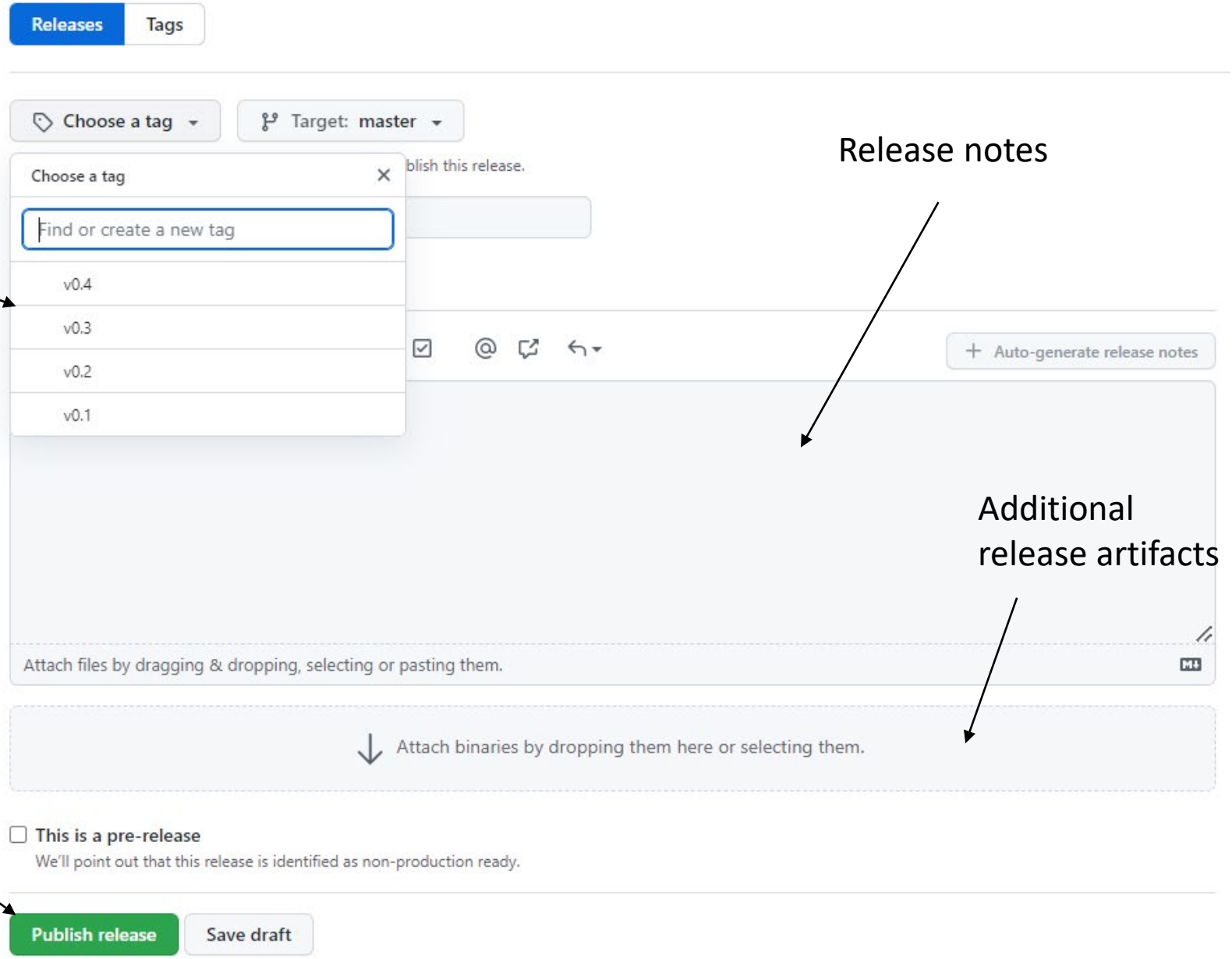
Release

Select tag to use for this release

Release notes

Additional release artifacts

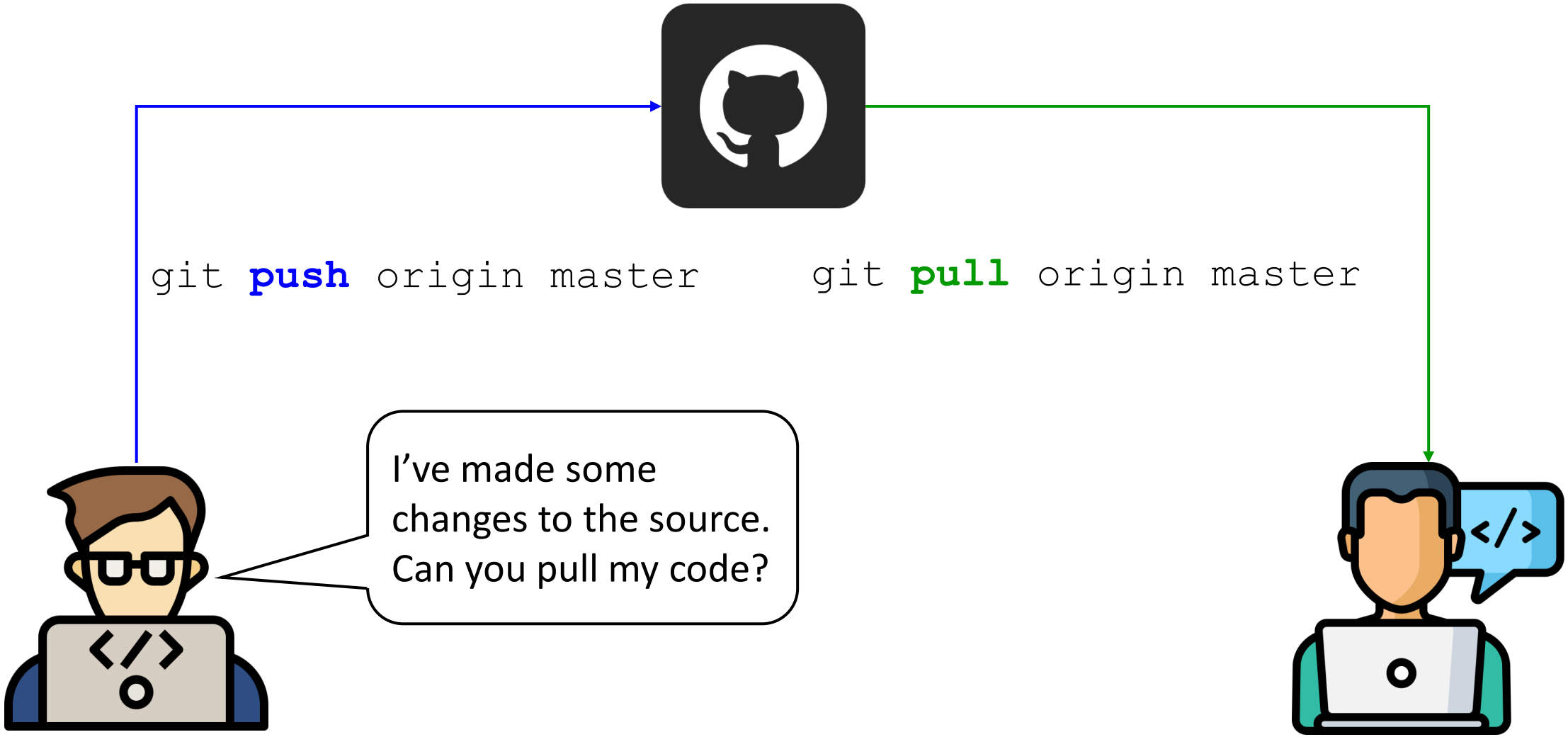
Publish the release



Collaborating with Others

- Collaborators are developers who are working with you on the same repository
  - Add them into the repo as collaborators under Setting
- Collaborators can pull and push changes to the repository
- Can cause issues if we allow collaborates (or even ourselves) to merge changes without any code review

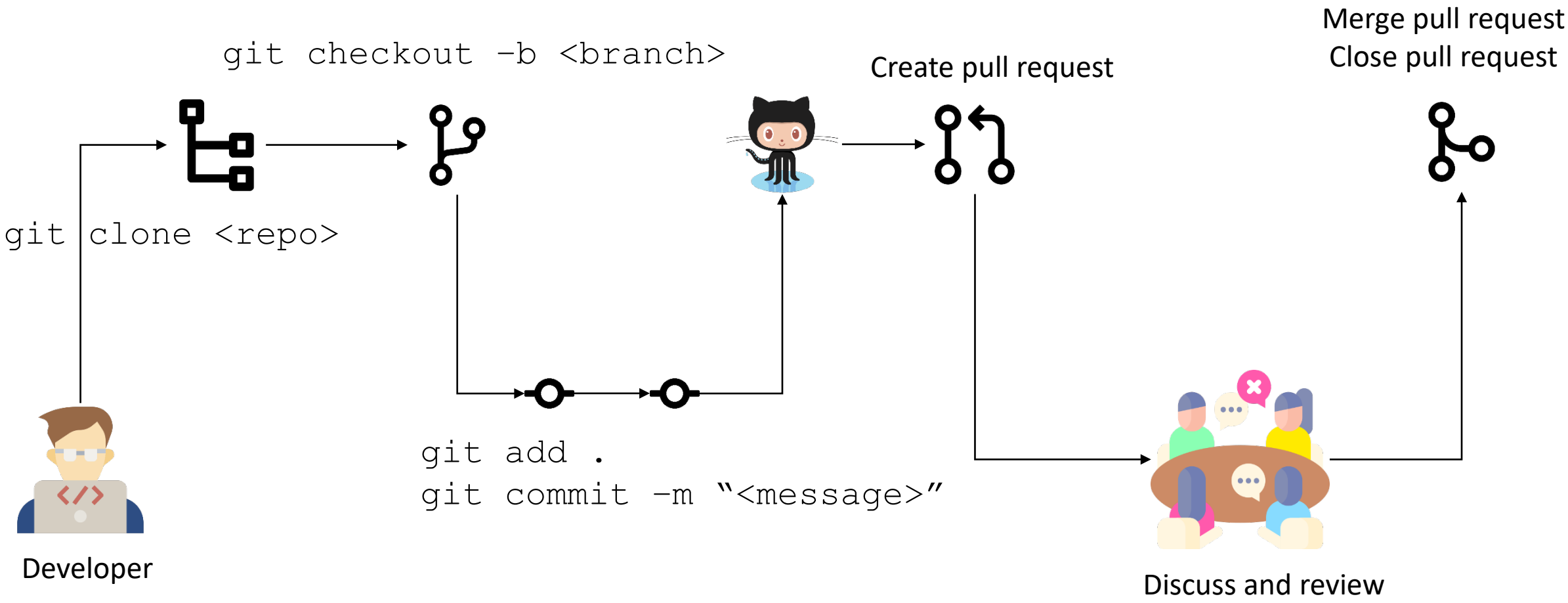
Collaborating



Pull Request (PR)

- Not a git pull
- Pull request is a proposal for a potential change in the code
  - Changes can be reviewed and discussed by collaborators in the project
  - Not a good idea to merge changes from branch to a branch (eg. master) without some review
- Pull request is a process
  - Combines multiple git command
  - Most Git hosting platform supports the process with discussion forums

Pull Request Process



Protected Branches

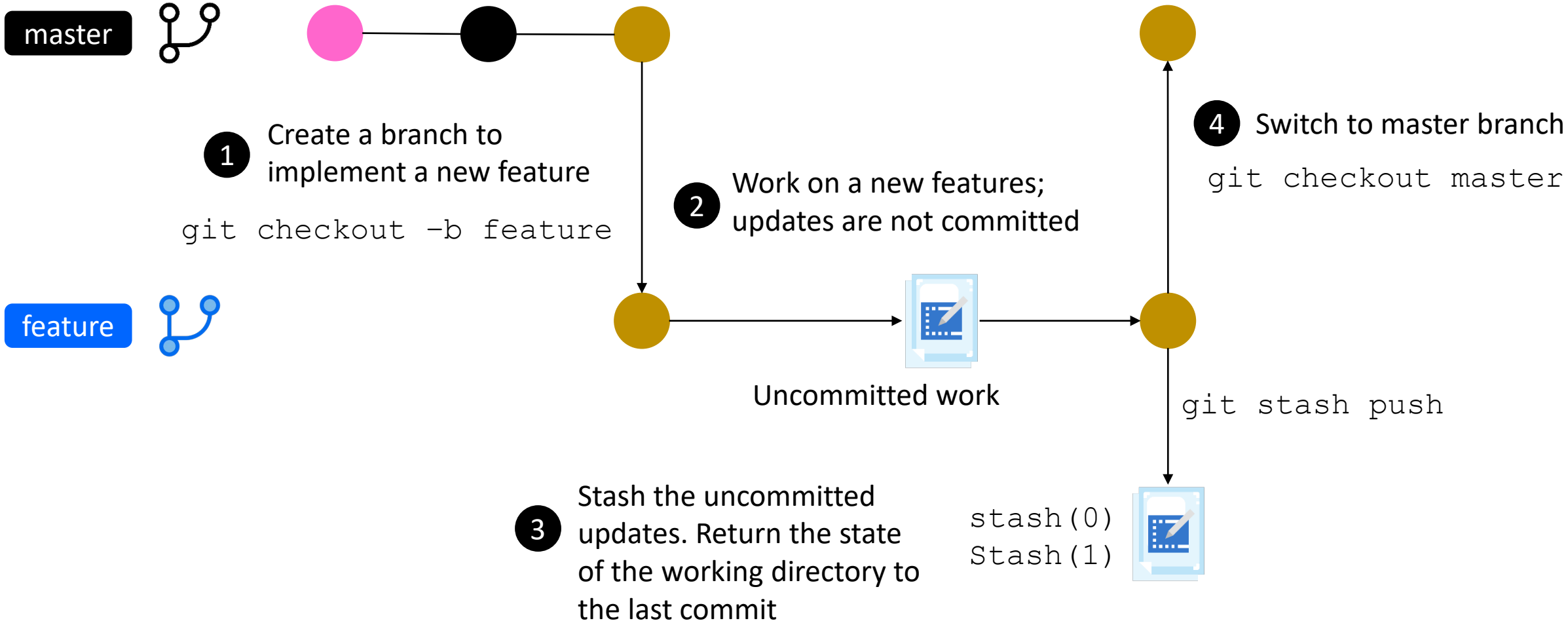
- Protected branch are branches that have protection rules
  - Prevents branches from being deleted, requires PR before merging
  - Eg cannot merge to master without creating a PR
- Works with collaborators
  - These are developers who are working on the same project
  - Add them into the repo as collaborators under Setting
  - Collaborators can pull and push changes to the repository

Appendix

Stash

- Change to another master while your are working a feature branch
  - Eg. require to look at an issue
- Cannot switch if you have uncommitted work in your in your current branch
  - May not want to commit your work because you have not completed it
- Stash allows you to temporarily shelve your uncommitted work
  - Shelve all uncommitted work from the last commit
  - You branch reverts back to the last commit
  - Checkout an other branch
  - Reapply the shelved work when you check back to your working branch

Git Stash Illustrated



Git Stash

- Stash all uncommitted work

```
git stash save “message”
```

- Apply a stash to the current working directory

```
git stash apply stash{n}
```

- Apply the top stash to the current working directory and delete the stash

```
git stash pop
```

- Delete a stash without applying it

```
git stash drop stash{n}
```

- List all stash

```
git stash list
```

- Delete all stashes

```
git stash clear
```

Git Stash Use Cases

- Interrupted by some urgent work in another branch
  - Need to shelve uncommitted work before switching to another branch
  - Do now what to commit unfinished work, stashing is a good way
  - Eg. Need to work on a urgent bug fix
- Work on a wrong branch
  - Make modification on the master branch when the modification is meant for feature branch
  - Stash all the work in master branch, switch over to feature branch and apply the stash to the branch

Git Stash Use Cases

- Updating the repository with the latest from remote
  - Started work on a branch and you remember that you forgot to perform a pull at the start of the day
  - Performing a pull will fail because you have modified one or more sources
  - Stash your work, which returns to the last commit, perform a pull and the apply the stash on top of the pull

```
<update sources>
git stash save “forgot to do a pull”
git pull
git stash pop
```

  - You may have to resolve conflict