



# INTRODUCTION TO GIT



# Repo 101 – Basic Terms

- **Repository** – A place where versions of a project are stored. In GIT you will have a local repository and may have an associated remote repository. If there is a remote repository used as a shared centralized repo, it is typically denoted origin.
- **Branch** – A place that one can work on a repository. A repository typically has a branch called master or main. New branches can be created as needed using the branch command

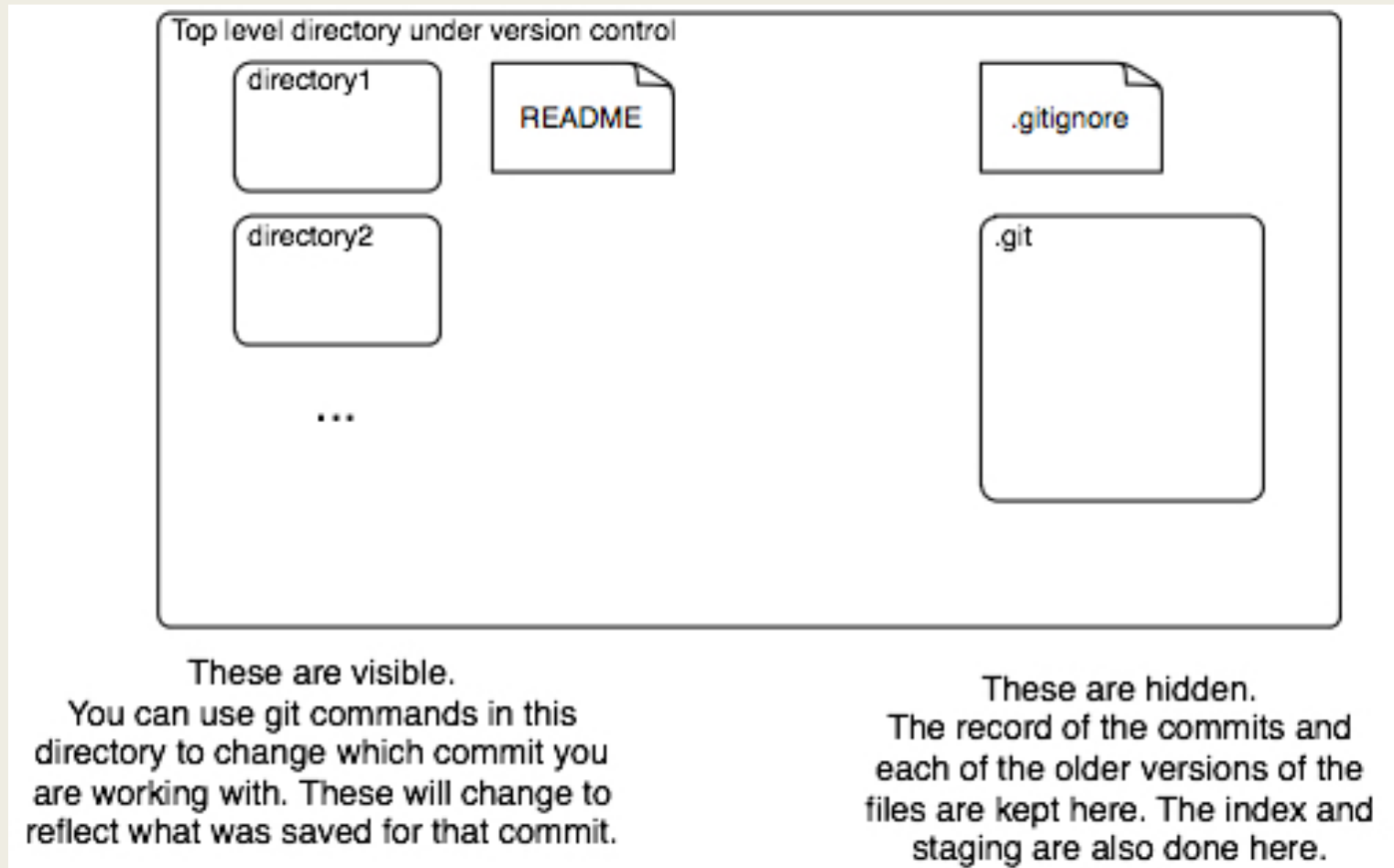
# Repo 101 – Lines of Text

- GIT uses a utility called diff (originally the Unix command) that takes two files and determines in which lines a change has occurred. GIT can efficiently store the changes in its history for those kinds of files
- Text file extensions – txt, java, tex, py, md
- Binary files - jpg, docx, pdf

# Repo 101 – Basic Terms

- **Working directory** – Where you are currently working and making changes.
- **Index** – A structure where changes to the working directory are staged.

# Repo 101 – A local repo



# Repo 101 – CLI

- One of the most basic ways of interacting with a *local* GIT repo is to enter commands into a Command Line Interface which is called a shell in UNIX terminology. Bash is a fairly common shell, but others are popular.
- You need to direct your shell so that your working directory is your repo and then enter git followed by arguments that specify the operation to perform.

# Repo 101 – Unix/Shell

- As a side note, the Unix operating system is the code that manages the resources on your computer.
- As a part of that operating system, we have a standard set of programs that can be run. This includes things like **ls** which allows you to list out all of the files in your current working directory.

# Repo 101 – Unix/Shell

- The operation of one of these programs can be affected by arguments/flags. The following lists all the files in the current working directory that end in .txt (the star is a wildcard).

```
ls -al *.txt
```

- The shell mediates between you and the operating system and maintains an environment. We can also create scripts that can be run in a shell.



# Repo 101 –Graphical Interaction

- There are GUI applications that allow you to do basic operations on your repo without having to use a CLI.
  - *GitHub Desktop is the one that is associated with GitHub.*
  - *SourceTree corresponds with BitBucket.*
  - *Plus others*

# Repo 101 – Which should I use?

- GUIs are often give you a convenient view of the changes in your repo and allow you to perform common operations quickly.
- CLIs allow you to do all operations, not just the most common ones. It's easier to do dangerous operations.
- There will be cases when you will need to use the CLI, so you should be familiar with it. Otherwise, it is a matter of preference.

# Repo 101 – .gitignore

- A place where we tell GIT that certain things don't need to be under version control. Anything that is autogenerated or holds personal settings is a good candidate.
  - *SampleProject/nbproject/private/*
  - *SampleProject /build/*
  - *SampleProject /nbbuild/*
  - *SampleProject /dist/*
  - *SampleProject /nbdist/*
  - *SampleProject /.nb-gradle/*

# Repo 101 – .gitignore

- When you create a remote repo at [www.github.com](https://www.github.com), you can specify the kind of project you are building and get a standard gitignore. There are also places on the web that have thousands of files covering different kinds of projects.

# Repo 101 – Basic Terms

- **Commit** – Save the state of the repo you are working on. Associated with the commit will be a unique identifier and a commit message. The commit message comes in two parts. The first part should be a clear summary of what was done and the second part allows you to go into more details. Good commit messages are critical. You should add your name to the comment if it isn't clear from your user name.
- A snapshot of the files that have been changed is created for the commit.

# Repo 101 – Basic Terms

- **HEAD** – The commit on the branch that is currently being worked on. If you checkout an earlier commit, your working directory will reflect the state of the project at that commit. The HEAD will become detached if you make any changes. To make changes off of an earlier commit, you can create a branch off of that commit.

# Repo 101 – Graph

There are two branches. The origin (remote) master and my master are at the same commit as of the last time we checked. Similarly for the development Branch. The HEAD for the remote is on the master branch. You can see each commit has an id.


All Branches

Show Remote Branches


Ancestor Order


Jump to:


Graph




Description


 master

 origin/master

 origin/HEAD

Added a print in main in the master branch

 origin/development


 development

Added a new line to main

Merge branch 'development' of https://bitbucket.org/charles\_hoot/mygitdemo into development

It hurts

A hit

 1.0.0

Merge branch 'master' of https://bitbucket.org/charles\_hoot/mygitdemo

Commit

a59ba70

e71706c

68a8f8c

e4ebf33

13ce061

ba5165b

Author

Charles Hoot <h...

Charles Hoot <ho...

unknown <HOOT...

Charles Hoot <ho...

unknown <HOOT...

unknown <HOOT...

Date

Today, 9:36 AM

Today, 9:34 AM

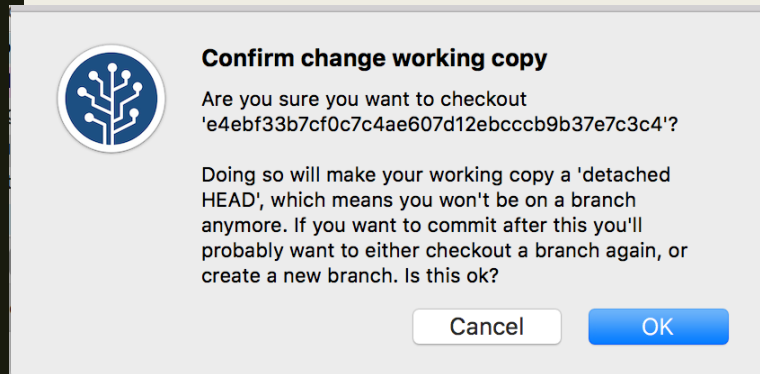
Today, 4:33 AM

Today, 4:31 AM

Today, 4:29 AM

Today, 4:12 AM

# Repo 101 – Going Back











Graph	Description	Commit
	origin/master  origin/HEAD  master Added a print in main in the master branch	a59ba70
	origin/development  development Added a new line to main	e71706c
	Merge branch 'development' of https://bitbucket.org/charles_hoot/mygitdemo into development	68a8f8c
	HEAD It hurts	e4ebf33
	A hit	13ce061
	1.0.0 Merge branch 'master' of https://bitbucket.org/charles_hoot/mygitdemo	ba5165b



# Repo 101 – Going back with the intent of making changes.

A branch (backInTime) was created at a specific commit.  
Now we have a place to work.

Graph	Description	Commit
	 origin/master  origin/HEAD  master Added a print in main in the master branch	a59ba70
	 origin/development  development Added a new line to main	e71706c
	Merge branch 'development' of https://bitbucket.org/charles_hoot/mygitdemo into development	68a8f8c
	 backInTime It hurts	e4ebf33
	A hit	13ce061
	 1.0.0 Merge branch 'master' of https://bitbucket.org/charles_hoot/mygitdemo	ba5165b

# Repo 101 – More Basic Terms

- **Pull** – pull the current state of the branch that you are working on from the remote repository to your local repository.
- **Push** – push the current state of the branch you are working on in the local repository to the remote repository.

# Repo 101 – More Basic Terms

- **Merge conflict** – What happens when two different local users try to push a commit with simultaneous changes to one or more files.
- **Merge conflict resolution** – The process of taking a file with a merge conflict and changing it some appropriate way.
- **Tag** – A command that lets you mark a particular commit as being important. For example, when you have a release version of your product, you can use tag to annotate it.

# Repo 101 – Merge Conflict Example

- We have different sources of the truth and they are all represented in the merge conflict.
- In the example, there are 3 lines that show where a change has occurred and they are marked in red.

# Repo 101 – Merge Conflict Example (diff output)

- *Common line*
- *Common line*
- *+<<<<<<<< HEAD*
- *Block of lines*
- *From version 1*
- *=====*
- *More lines*
- *From version 2*
- *>>>>>>>>>> 4f00f39c...*
- *Common line*
- *Common line*



# A GIT WORKFLOW

# Workflow– Quick In, Quick Out

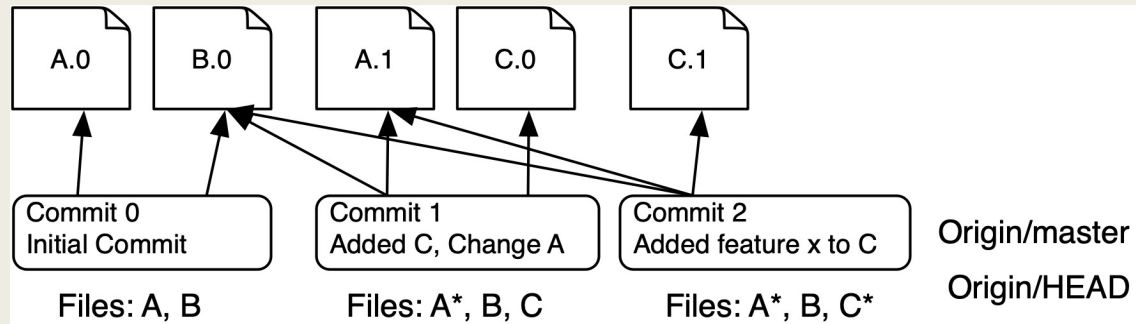
- Pull from the remote repo to your local repo.
- Edit the local.
- Verify you haven't broken anything.
- Commit locally
- Push from local to remote.

# Workflow– Quick In, Quick Out

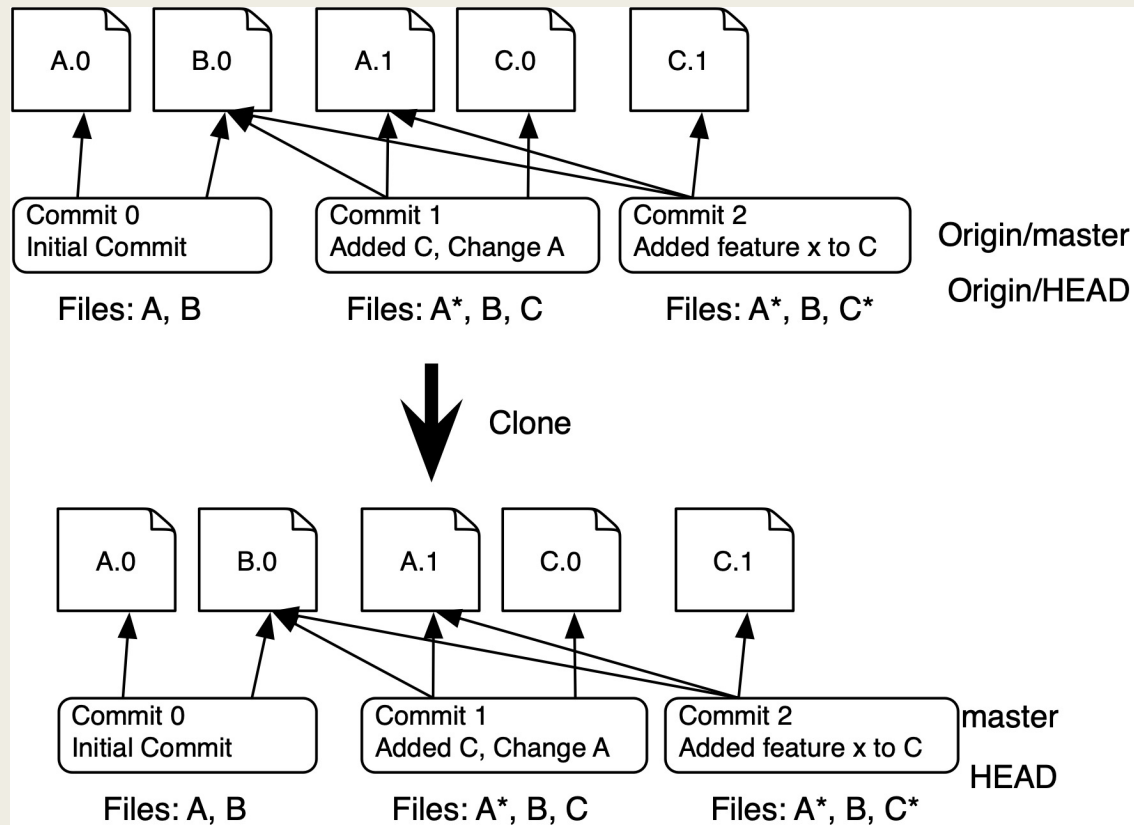
- The goal is to avoid merge conflicts.
- You always want your local version to be as recent as possible. Don't work on a stale repo.
- You don't want to break the remote repo, so check the correctness of your change. It is often helpful to have code that you can run to verify the basic functionality. The faster you complete, the less likely the remote changed.
- Use good coding practices like decomposition to break the project up into smaller pieces. If you have large files, merge conflicts are more likely.



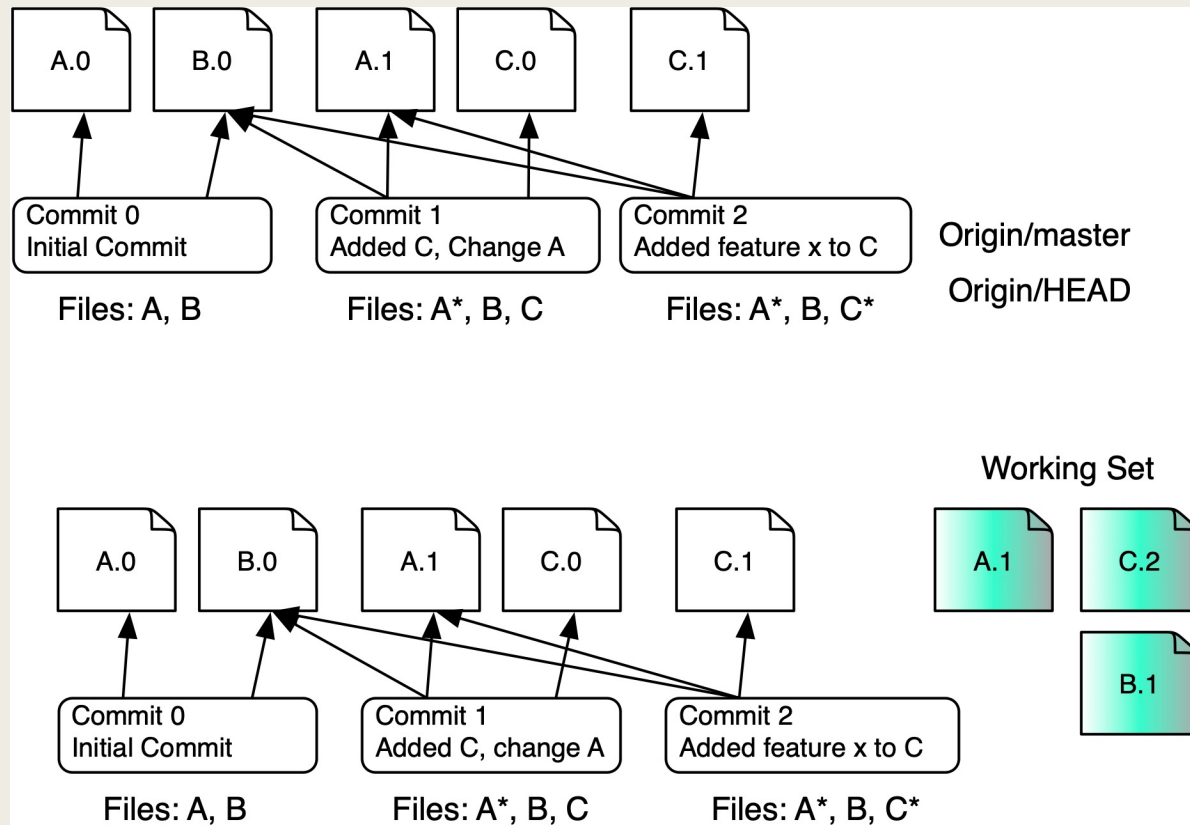
# Workflow- Remote



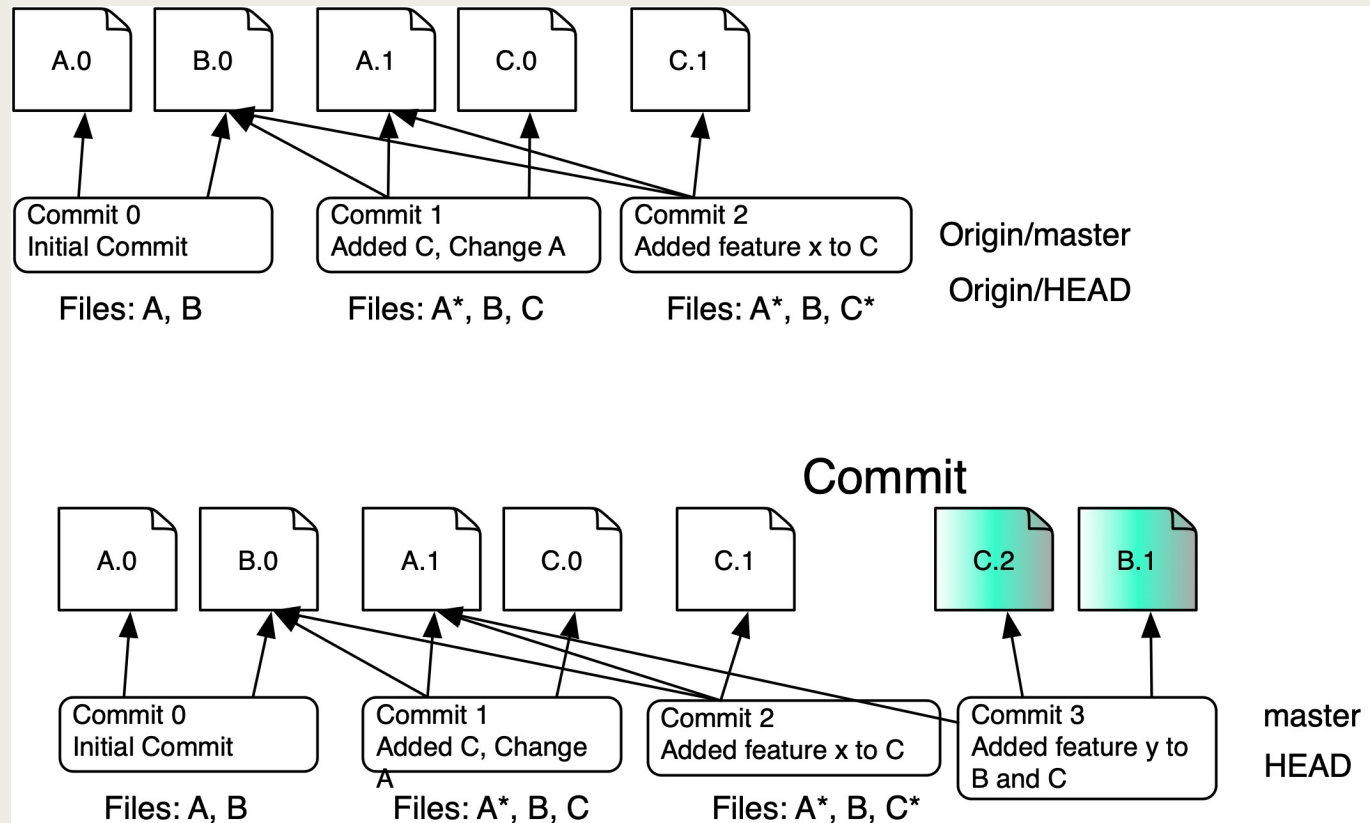
# Workflow – Clone/Pull



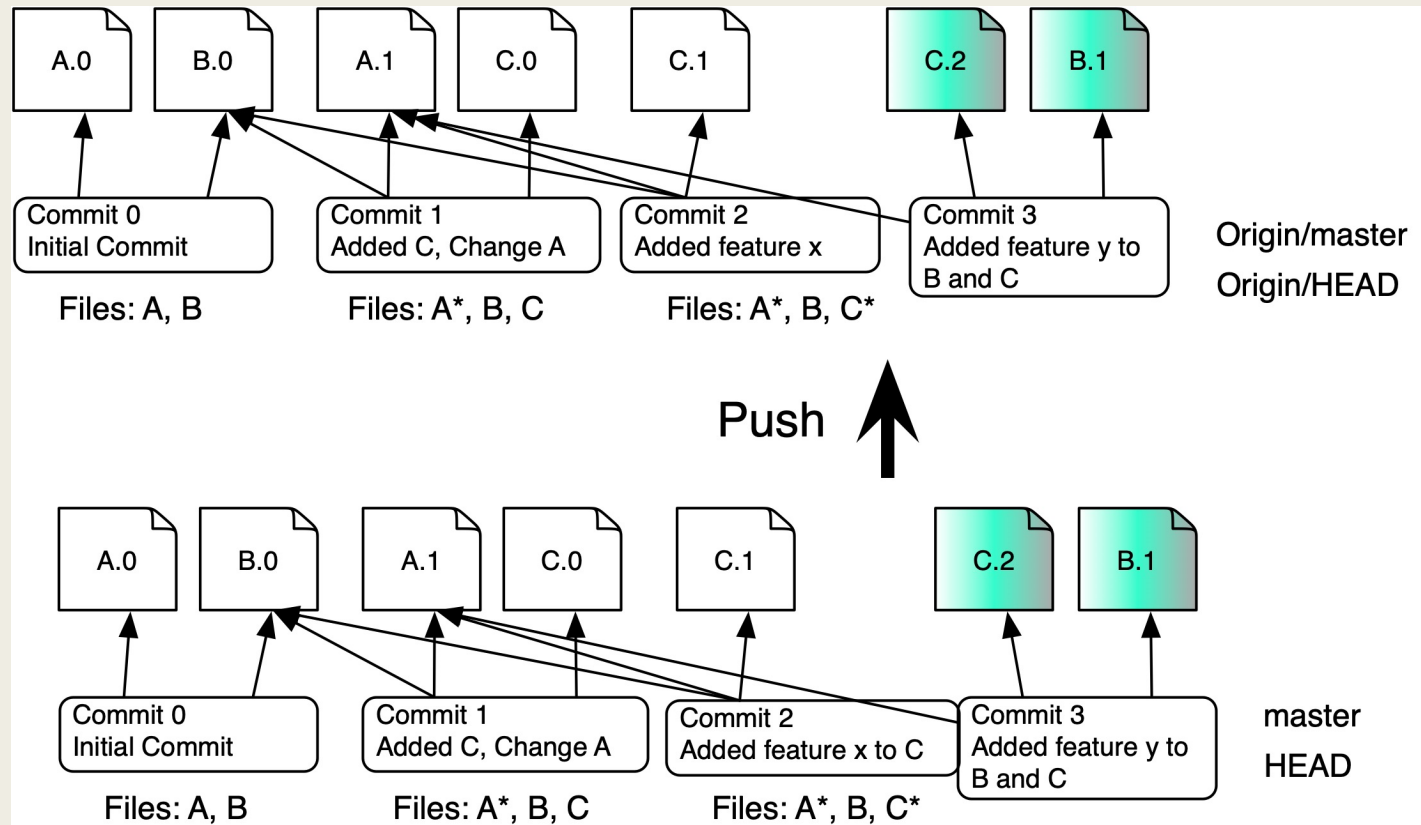
# Workflow- Edit



# Workflow - Commit



# WorkFlow- Push





# MERGE CONFLICT VISUALIZATION



# Merging

- When you push to the remote, it is possible that one of the files you changed, has also changed on the remote.
- GIT does a diff to determine where the changes are. If the changes are on different lines, then GIT makes the assumption that it is safe to merge the versions by making both changes.
- If the changes are on the same line, GIT requires intervention. You have to edit the file with the merge conflict. You edit to use some combo or both changes including something completely different.

# Merging

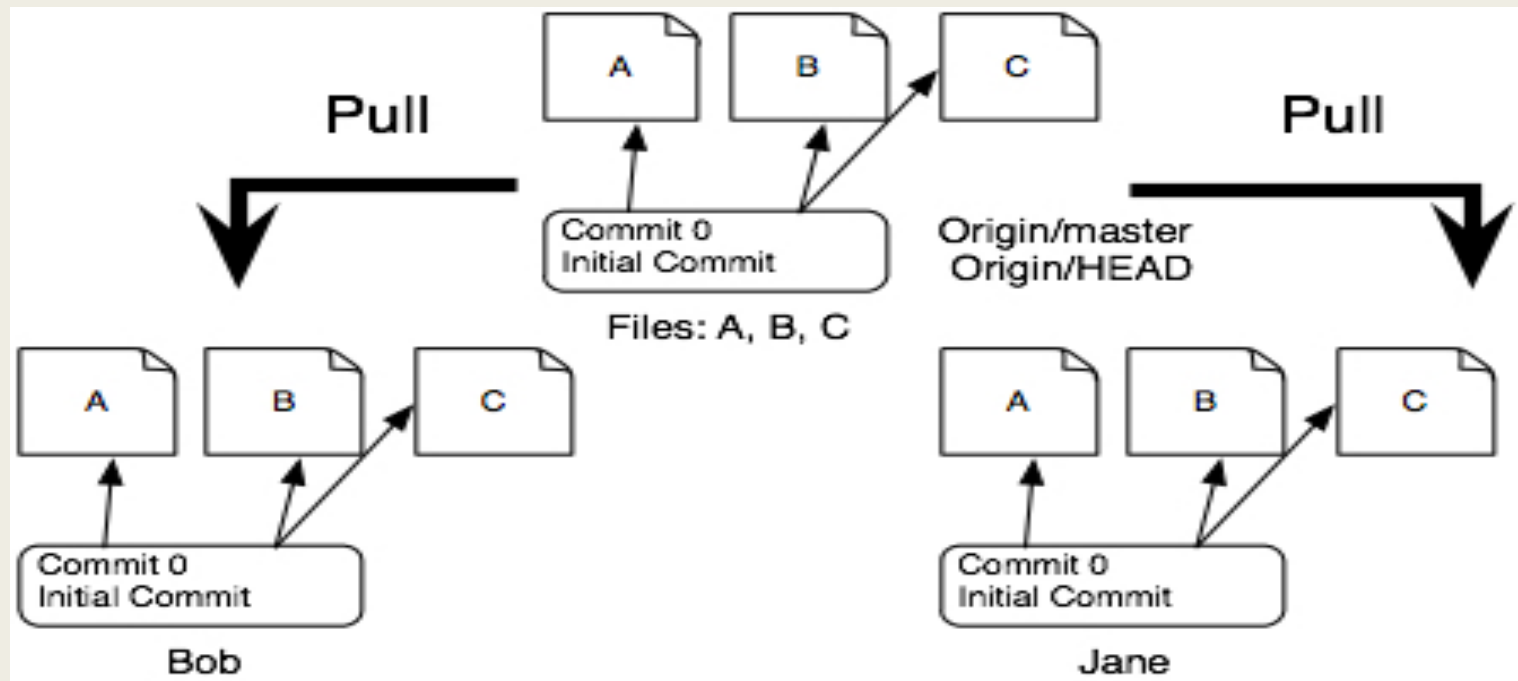
- Merge conflicts are your friend.
- Suppose you and I notice a problem in the code and fix it.
  - *I change line 20*
  - *You change line 25*
- Both changes get made and it is unlikely that the code now works. Furthermore, GIT silently makes the change.



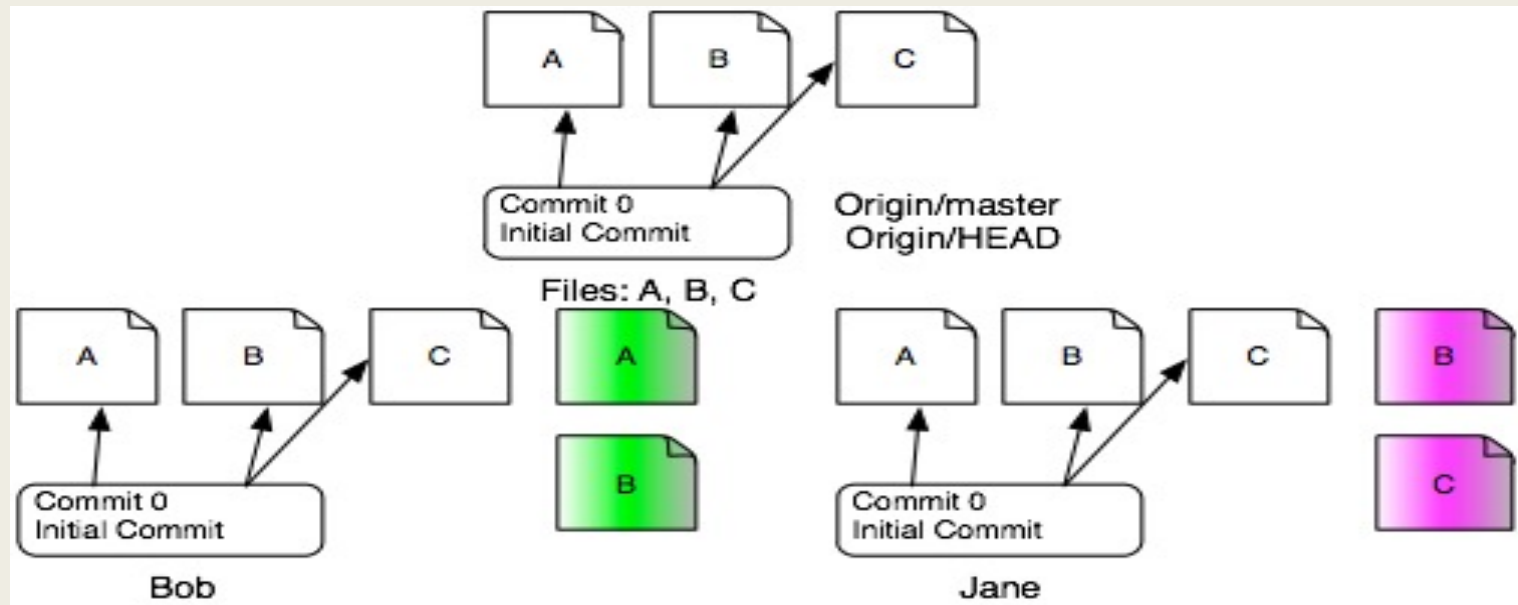
# Merging

- Merge conflicts are most common when multiple people work on the same part of the code.
- You as a solo developer are less likely to have a merge conflict, but it can still occur.
  - *You work on two branches and then merge the branches*
  - *You work on different local copies of your repo (presumably on different computers)*

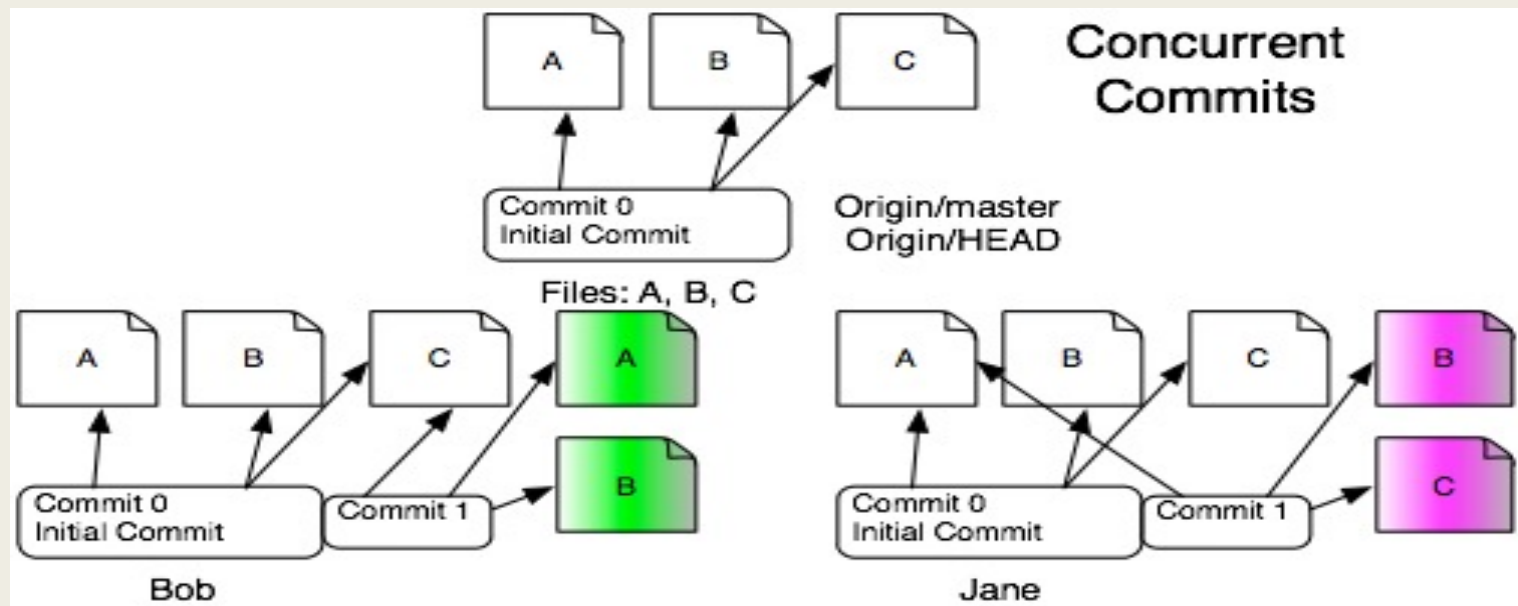
# Merge – Shared Remote



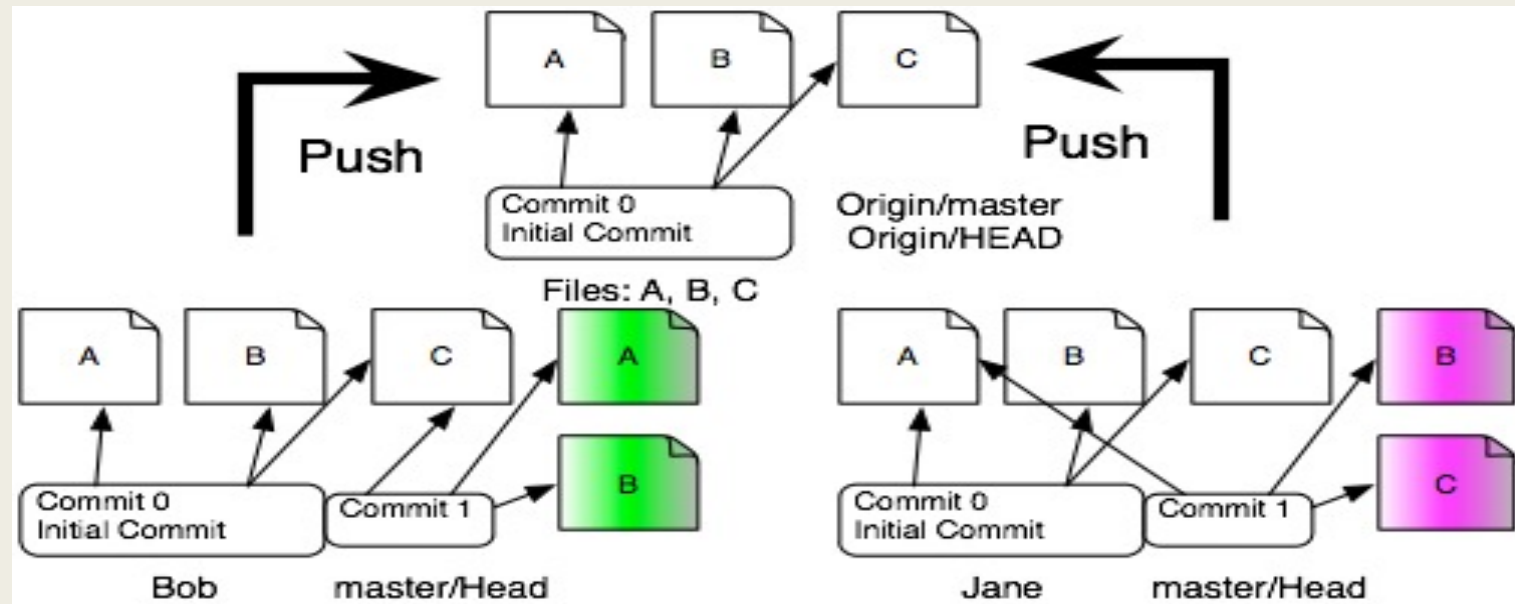
# Merge – Each works on their local repo



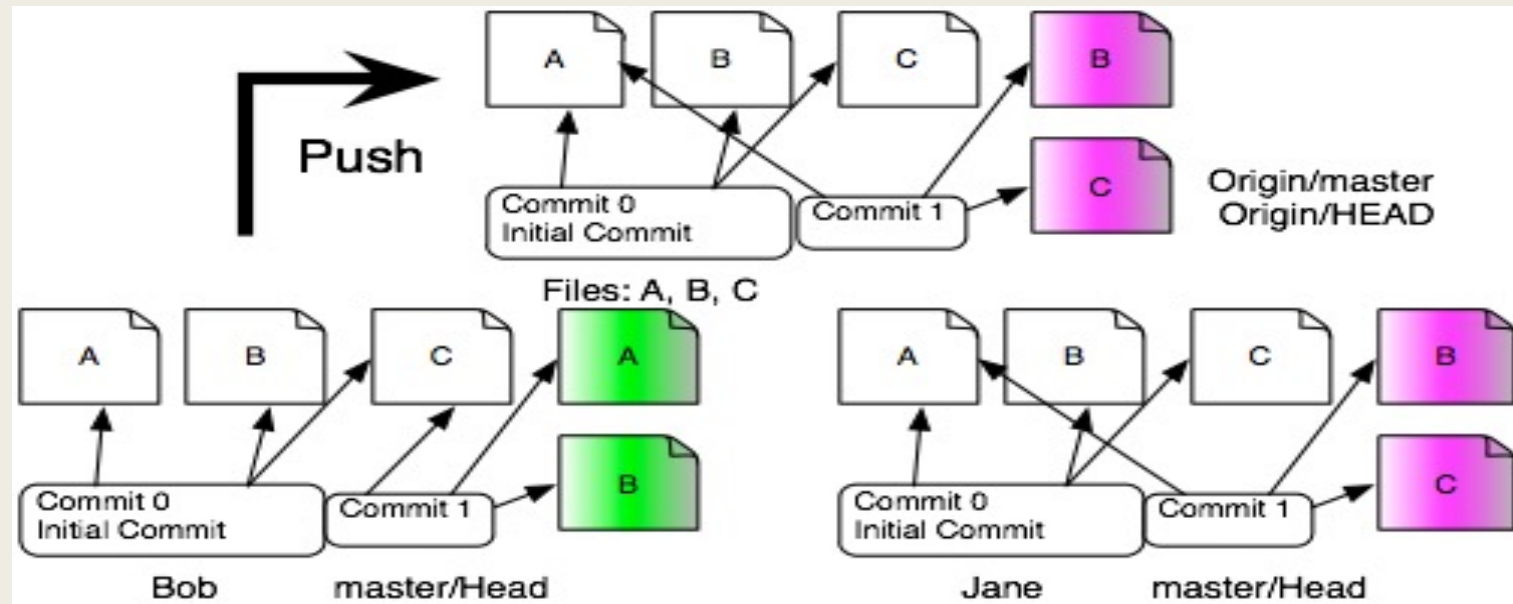
# Merge – Each commits to their local repo



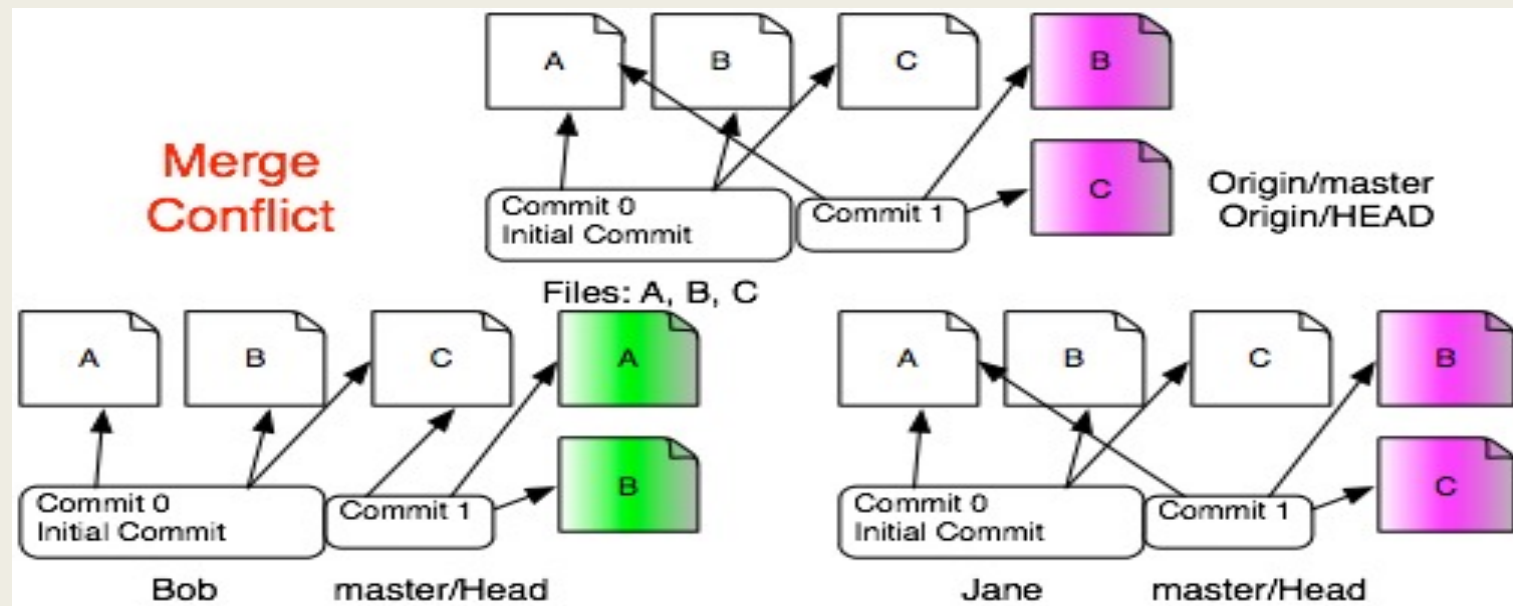
# Merge – Each pushes to the shared repo



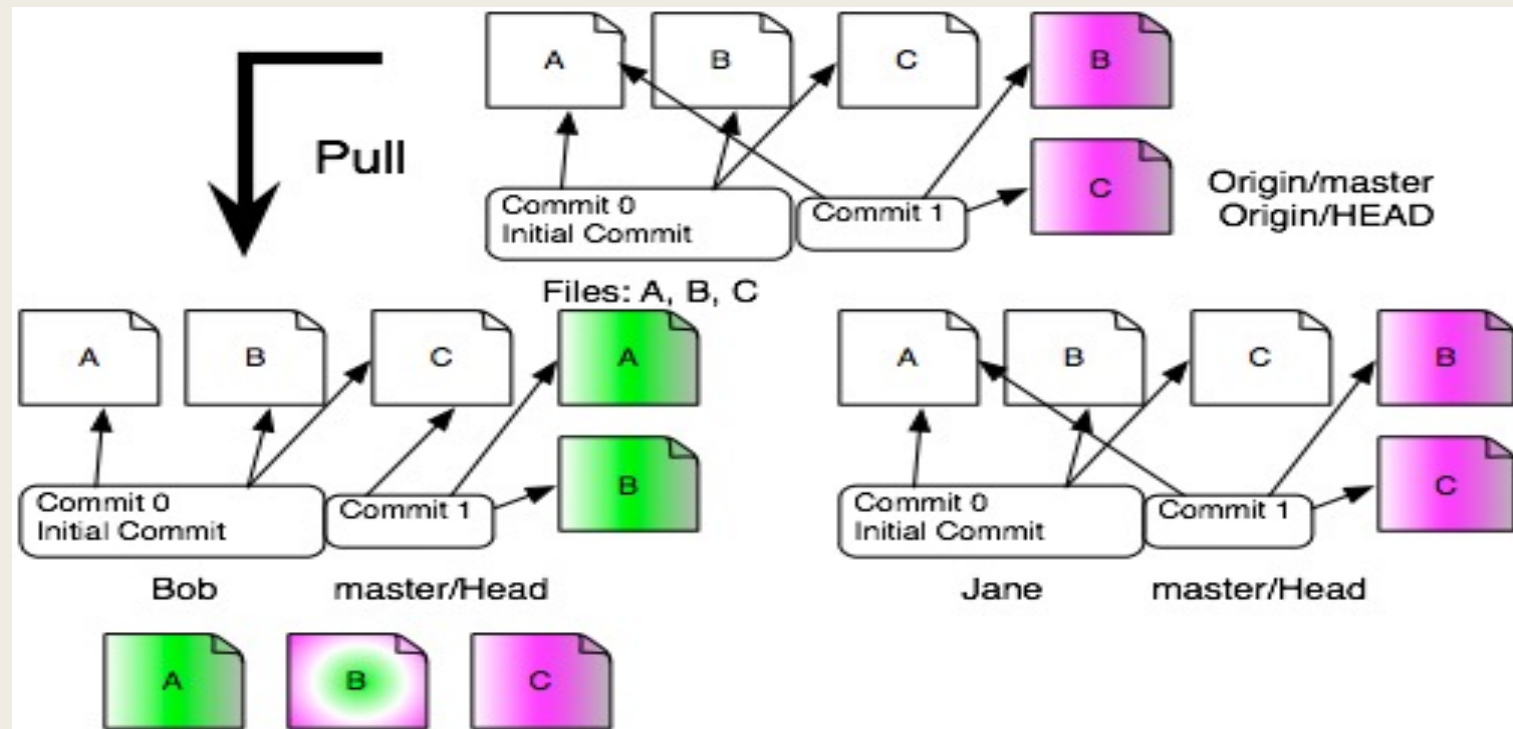
# Merge – Someone wins and someone does not.



# Merge – Bob is unlucky and has a merge conflict.

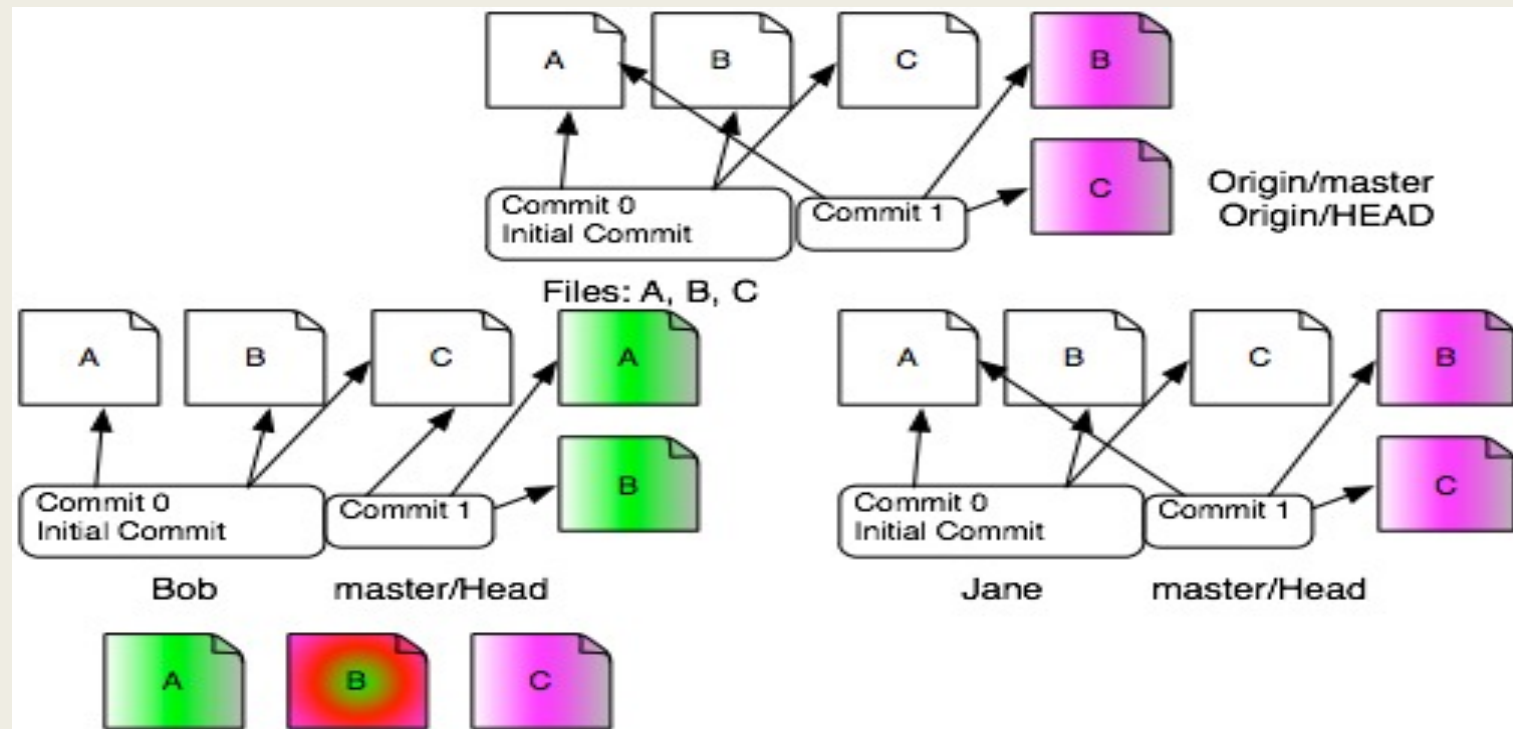


# Merge – Do a Pull so all the information is available

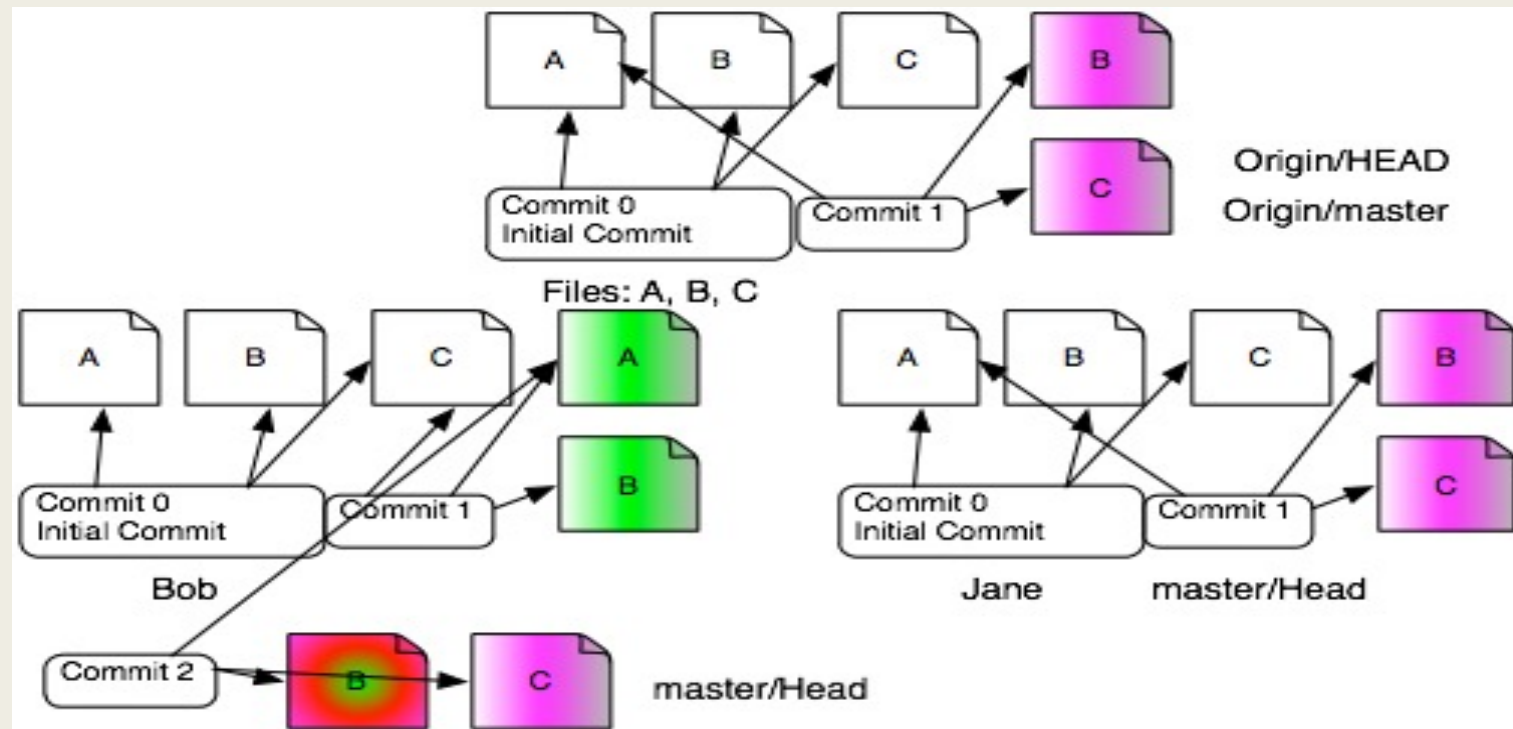




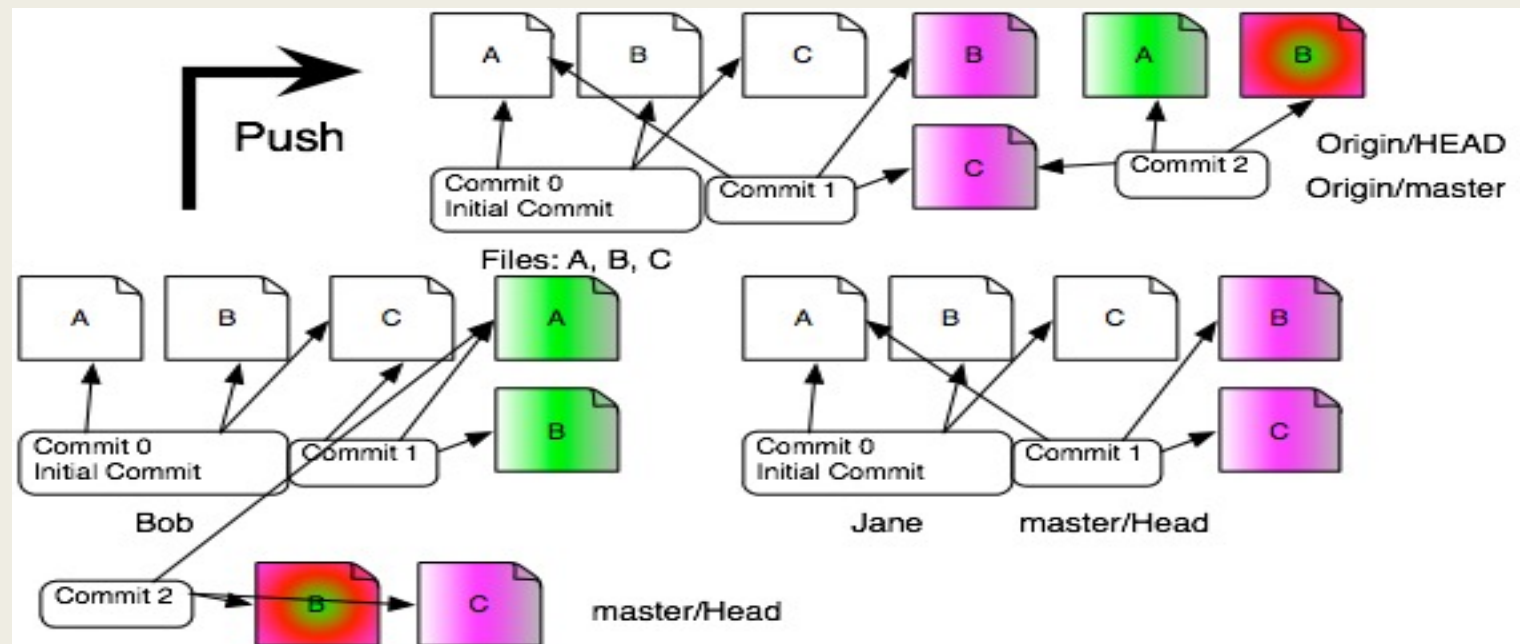
# Merge – Fix the conflict



# Merge – Commit



# Merge – Push the resolved





# ANOTHER GIT WORKFLOW

# Workflow– Stable Master

- Pull from the remote repo to your local repo.
- Branch from master (lets call that the development branch)
- Work normally on the development branch  
(pull/commit/push)
- When you have completed creating/testing the development,  
merge the master into the development branch
- Resolve merge conflicts
- Merge the development branch into master.

# Workflow– Stable Master

- The goal is to keep a working version of the project on the master branch.
- The chaos of development is sequestered.
- We embrace the possibilities of merge conflicts, but, again keep those off of the master.
- Small bug fixes can still be done on the master, but any major development is not.

# References

- [GitHub documentation](#)
- [Using GIT](#)
- [GitHub Desktop documentation](#)
- [gitignore templates](#)
- [Basic shell commands](#)