GIT

Initialize GIT repository

In your working directory, make an empty directory called ‘project_dir’.

>cd project_dir

>git init

>git status

Add a file into GIT repository

Make a file called hello.html using notepad++ and save it in the project_dir directory. Add this as the content of hello.html

<!DOCTYPE html><html><body>Hello World</body></html>

git status

git add hello.html

git commit –m “my first commit”

git status

git branch

Basic Git Workflow

1) You modify files in your working directory.
2) You stage the files, adding snapshots of them to your staging area.
3) You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

Terminology

- The .git Directory
  - This directory might be hidden. You will need to do ls –a to see it or in Windows Explorer you will need to turn on the option to see hidden files and folders.
  - The .git directory is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you clone a repository from another computer.

Terminology

- The Working Directory
  - The working directory is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.
- The Staging Area
  - The staging area is a file, generally contained in your Git directory, that stores information about what will go into your next commit. It's sometimes referred to as the "index", but it's also common to refer to it as the staging area.

Making changes

Make changes in the hello.html file and save the changes.

```
>git status
```

Make a file called hello_again.html and save it.

```
>git status
>git add hello_again.html
>git commit –m “my second commit”
>git status
```

You will notice that the changes to hello.html are not committed because it wasn’t added to the staging area after changes.

```
>git log
```

4. Staging and Committing

Make a new file called hello3.html & run

```
>git add .
```

```
>git status
```

The . is a shortcut that says add all files in this directory and below. This particular command will add the unstaged hello.html (from earlier) as well as hello3.html to the staging area. Now make a change in the file hello3.html. Add a line and save it for example.

```
>git status
```

4. Staging and Committing continue…

Now there will be the same hello3.html file visible as staged as well as unstaged. If you commit, only the staged version will be committed.

```
>git add .
>git commit
```

Ignoring files (.gitignore)

The purpose of gitignore files is to ensure that certain files are not tracked. A .gitignore file has one pattern per line.

Patterns can have filepaths and wildcards. For example:

```
# Ignore configuration files that may contain sensitive information.
sites/*/settings*.php

# Ignore paths that contain user-generated content.
sites/*/files
sites/*/private
```

History

To see history, you can see use the following command

>git log

The command comes with many options:

>git log --pretty=oneline

>git log --pretty=oneline --max-count=2

>git log --pretty=oneline --since="5 minutes ago"

>git log --pretty=oneline --until='5 minutes ago'

>git log --pretty=oneline --author=<your name>

>git log --pretty=oneline --all

Aliases

- Sometimes its useful to have shortcuts for frequently used commands or long command line options.
- This is done by adding aliases to the .gitconfig file in your home directory. In Windows it is usually C:\users\username

Aliases continue…

Try adding the following in the .gitconfig file. Again this might be a hidden file. Find it and change this . The last line hist is useful. So add it exactly as given

```
[alias]
co = checkout
ci = commit
st = status
br = branch
hist = log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
```

Changing a commit

Once you make a commit and you realize you left out something. You forgot something in the file or you realized that the commit message wasn't correct. Git allows you to edit your previous commit – either change or add files that you left out and change the commit message also.

Steps for changing a commit

Change a file, stage it and commit it.

```
>git hist
```

Make another change and then

```
>git add hello.html
>git commit -–amend –m “oops forgot to add this line in the commit”
>git status
>git hist
```

Undoing changes

There are methods to undo your changes:

- Before staging by using

  >git checkout hello.html

- After staging by using

  >git reset

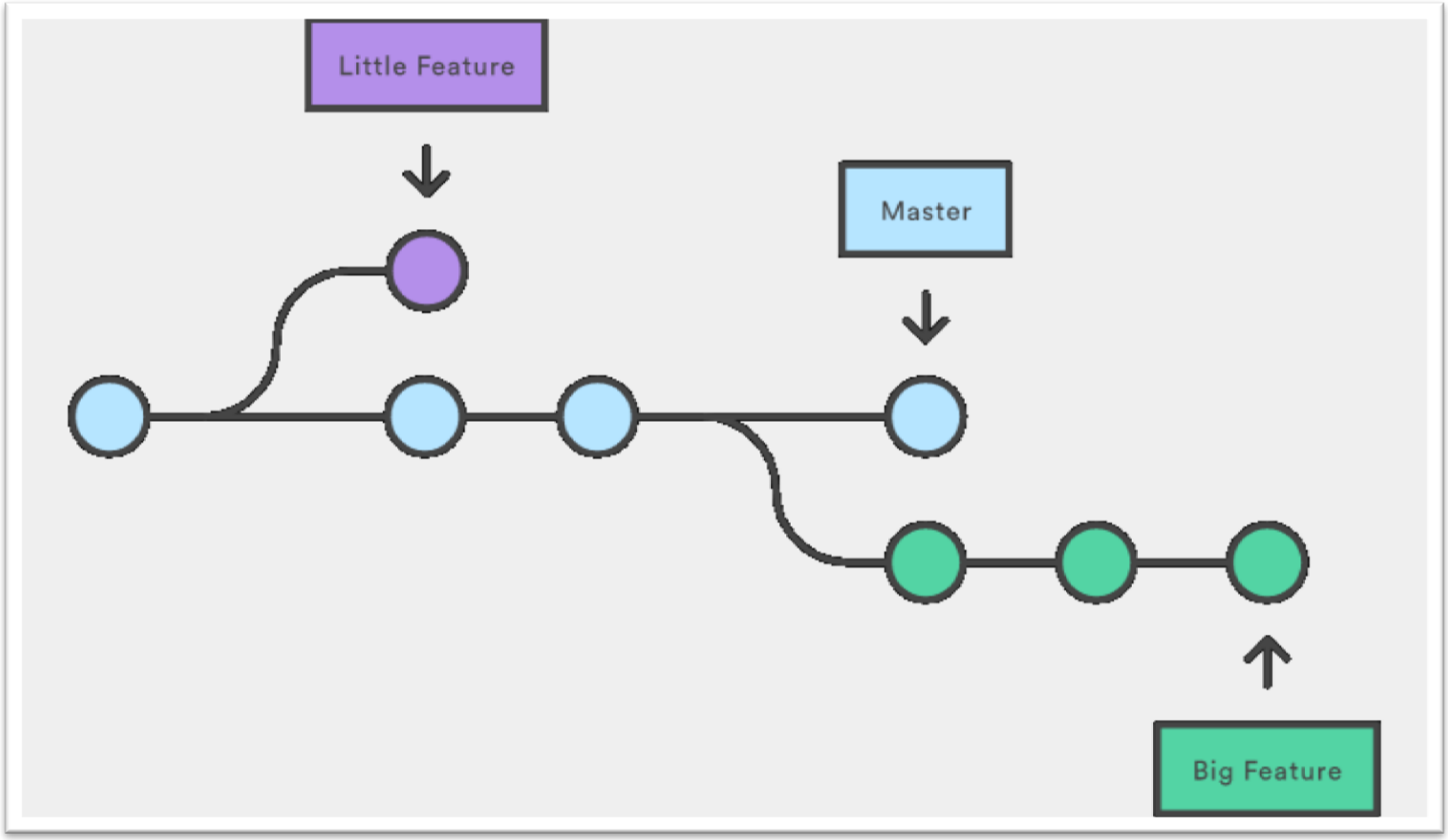
- After committing by using

  >git revert

Branching

- A branch represents an independent line of development.
- Every branch has a brand new working directory, staging area and project history.
- You can make changes, discard changes, make commits, record history, all without losing your original work.
- A typical scenario is keeping your production code on master branch and each feature on a separate, new branch.

Branching Diagram



Branching commands

- To create a new branch

  git branch my-new-branch

- To list all branches

  git branch

- To move to a branch

  git checkout my-new-branch

- To create a new branch and move to it (short-cut for the two steps above)

  git checkout –b my-new-branch2

Branching Exercise

- Create a new branch called dev-branch

  git checkout –b dev_branch

  git branch

- Open the existing file hello3.html and make a change. Create a new file hello4.html. Add both to the staging area and commit both.

  git status

  git add .

  git commit –m “checking how branches work”

Branching Exercise continue…

Now navigate back to the master branch. What happens to the file? Navigate back to the dev-branch. What happens now?

If you notice, the working tree changes to reflect the branch.

Merging

You have a development branch. This got tested and deployed to production. How do you now bring your master branch up to date?

This can be done with a merge or a rebase. We will discuss merging now and rebasing later.

Remember the steps for merging

1) Move to the main branch
2) Merge the side branch into the main branch.
3) After the side branch has been merged, you can delete it.

Merging (contd.)

git checkout master

git merge dev-branch

git branch –d dev-branch

If you try to delete an un-merged branch, Git won't let you… unless you force it with a -D

Types of Merging

- Fast forward merging

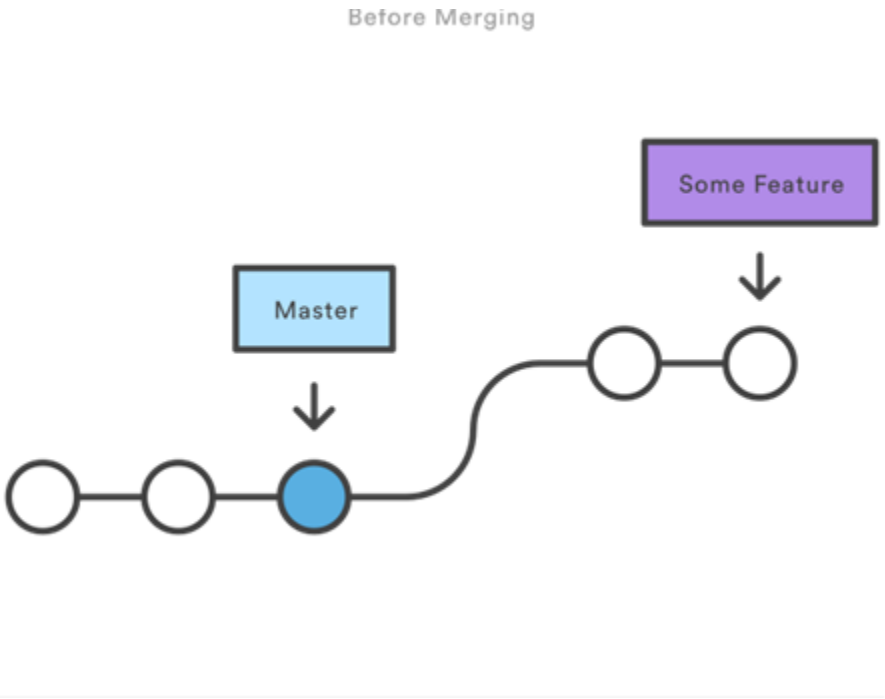
  In fast forward merging, parent does not move ahead while branch merging.

- True merging or 3-way merging

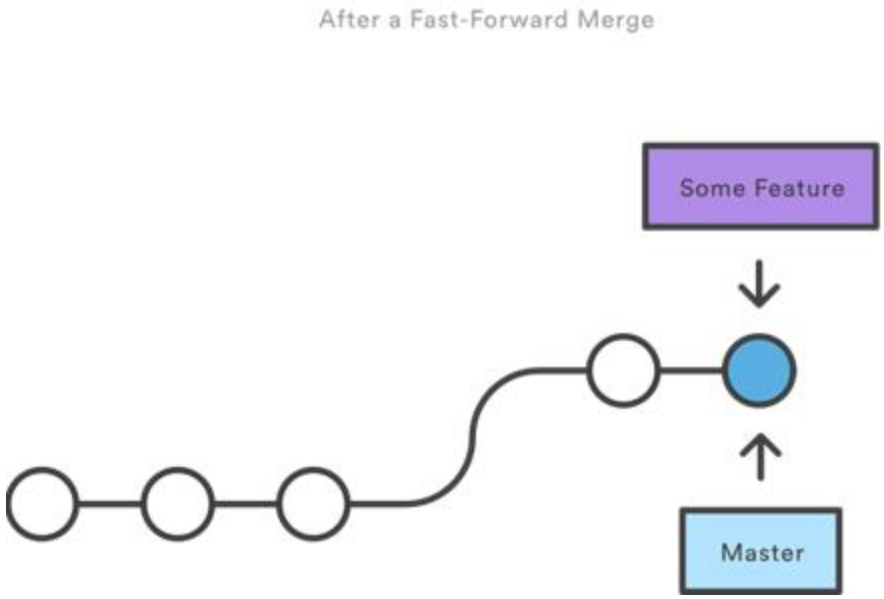
  In true merging or 3-way merging, parent moves ahead while branch merging.

Fast Forward Merging

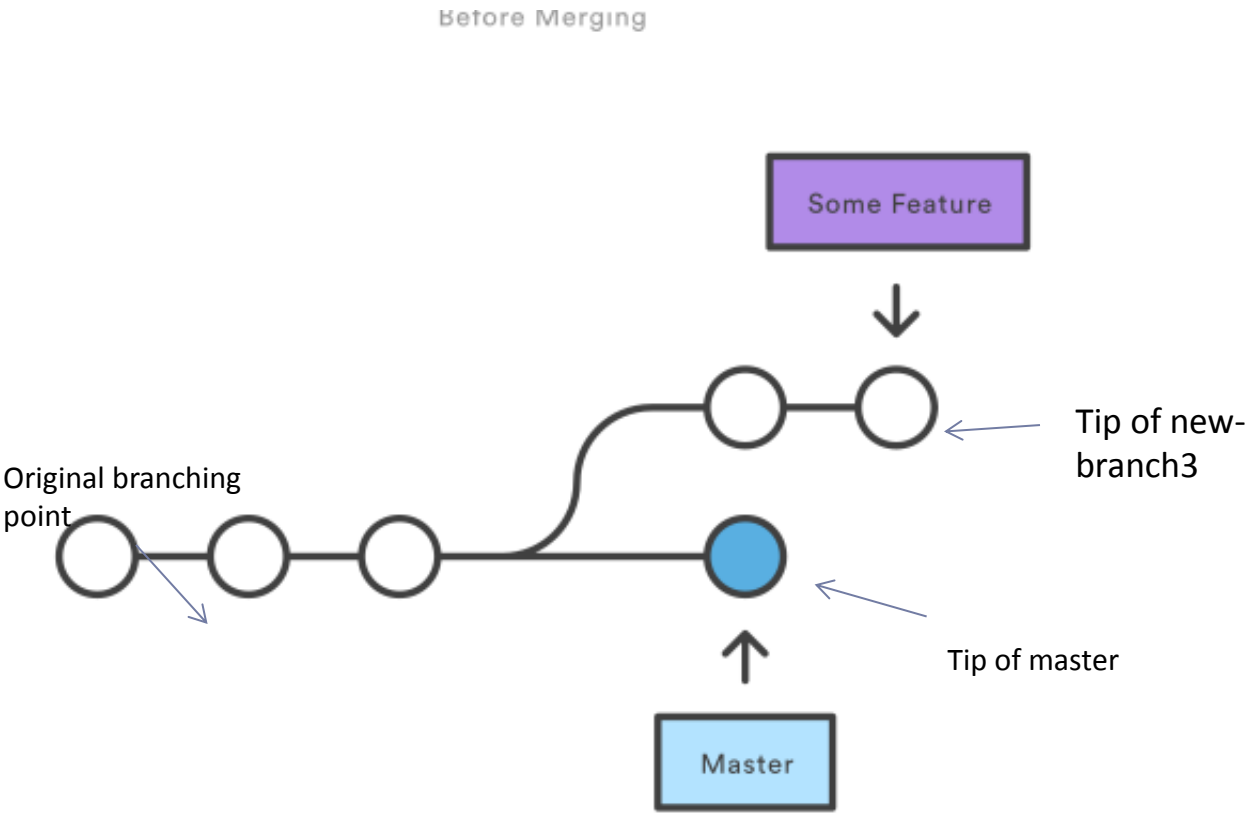
When the parent branch hasn't moved ahead from when you branched off.



Fast Forward Merging



True Merging or 3-way margin



True Merging or 3-way merging

True Merging – When the parent branch has moved ahead. We can divide true merging under 3 different cases:

- Different files changed.
- Same file has been changed but different line.
- Same file is changed on same line.

Case 1: Different files changed

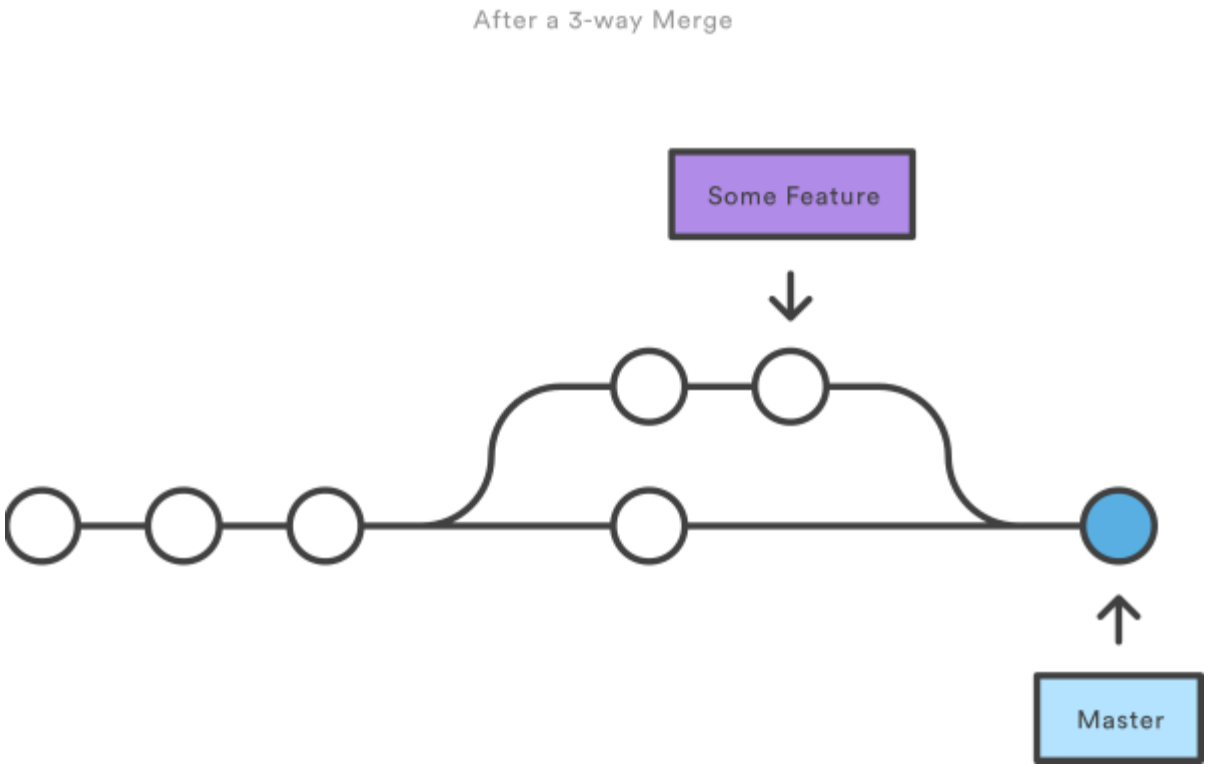
- Make a new branch called new-branch3 and navigate to it.
- Add a new file hello-newbranch.html in the new branch. Stage and commit it.
- In the meanwhile there needs to be an urgent commit on the production branch. So go back to the master branch and make a new file called urgent_fix.html and add it to the staging and commit it.
- Now navigate back to the new-branch3 make another change in hello-newbranch.html. Stage and commit.

Case 1: Different files changed

Now that you have completed development on your new feature on new-branch3, you want to merge it back to the master branch. How will you do this?

1) Move to the main branch (git checkout master)
2) Merge the side branch into the main branch. (git merge new-branch3)
3) After the side branch has been merged, you may delete it. (git branch –d new-branch3)

True Merging or 3-way merging



Case 2: Same File has changed but in different lines

- Make a new branch called new-branch4 and navigate to it.

  git checkout –b new-branch4

- Change our original hello.html in the new branch in a part in the beginning of the file. Stage and commit it.

  git add hello.html

  git commit –m “added a new feature”

- Now checkout to the master branch

  git checkout master

Case 2: Same File has changed but in different lines

- Change hello.html again but at the end of the file.

  git add hello.html

  git commit –m “made another urgent fix”.

- Finally merge the new-branch4 into master.

  1) Move to the main branch
  2) Merge the side branch into the main branch.

  git checkout master

  git merge new-branch4 –m “merging both changes in the same file”

- Confirm both branch changes are automatically merged.

Case 3. Same File has changed but in the same line

- Make a new branch called new-branch5 and navigate to it.

  git checkout –b new-branch5

- Change our hello.html on the 10th line. Stage and commit it.

  git add hello.html.

  git commit –m “added a new feature on line 10”.

- Now checkout to the master branch (git checkout master).

- Change hello.html again but again on the 10th line.

  git add hello.html.

  git commit –m “refactored on line 10”.

Case 3. Same File has changed but in the same line

- Now do a merge the same old way.

  git checkout master

  git merge new-branch5

- Git doesn't know which version to keep. So it will throw up a conflict like this.

  Auto-merging hello.html

  CONFLICT (content): Merge conflict in hello.html

  Automatic merge failed; fix conflicts and then commit the result.

Case 3. Same File has changed but in the same line

- Open the file in your editor. You will see conflict markers:

```
<<<<<<< HEAD
Lines added in master
=======
Lines added in branch
>>>>>>> new-branch5
```

- One line marks what the line in the HEAD (or tip of the master is) and the other line shows you what is the version on the branch.
- You edit the file manually. Keep the line that should be there. Remove the conflict markers from the file.

Case 3. Same File has changed but in the same line

To resolve the conflict:

- Manually edit the file and keep the line that should be there. Remove the conflict markers

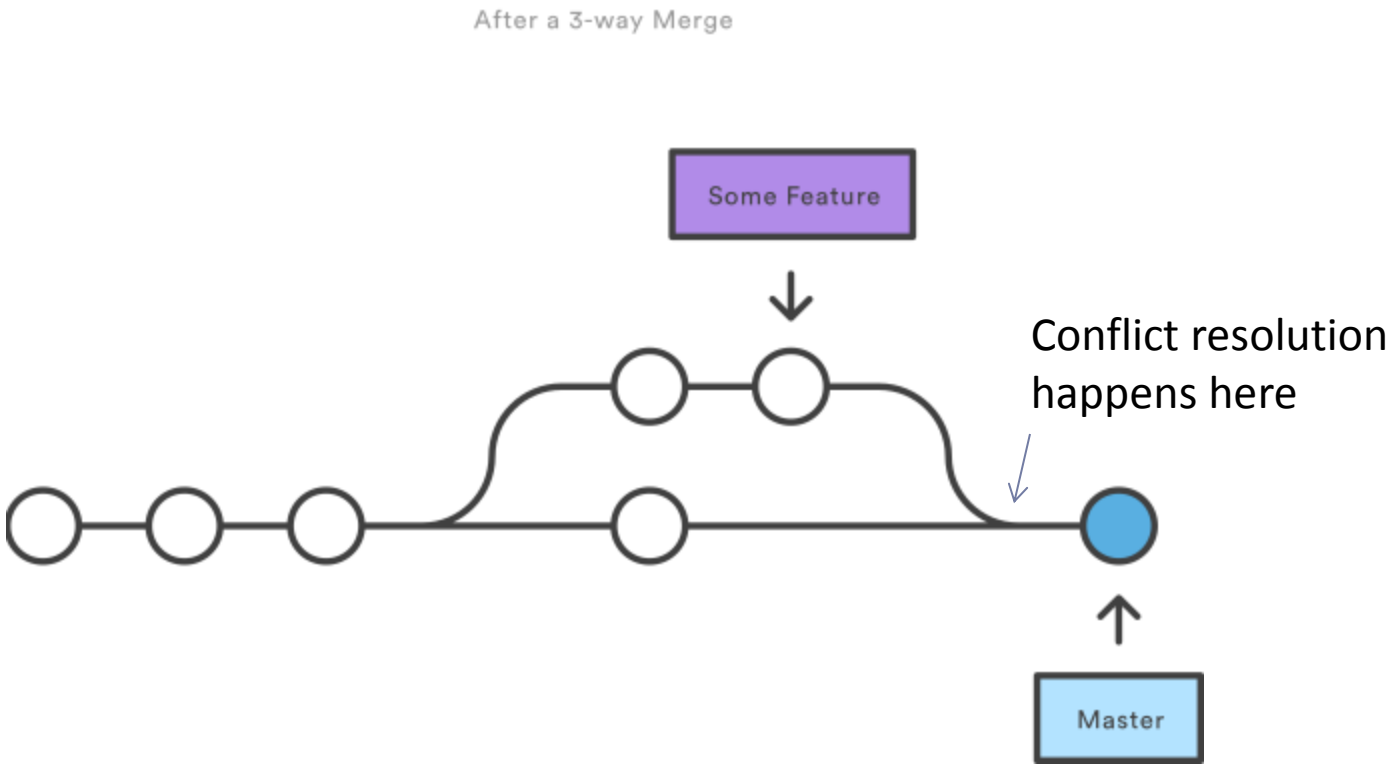
  ```
  <<<<HEAD
  ======
  >>>>new-branch5
  ```

- Do this for every conflicting part marked by the conflict markers. Then stage & commit the file again.

  ```
  git add hello.html

  git commit –m “new feature successfully merged”
  ```

Case 3. Same File has changed but in the same line



Rolling back a merge

- Sometimes things can go wrong. You attempt a merge and Git throws up a conflict. You see that there is confusion as to what part of the file to keep and what to discard. Maybe you need to discuss this with the developer whose branch you are trying to merge. In such a scenario, you can abort the merge with the following command.

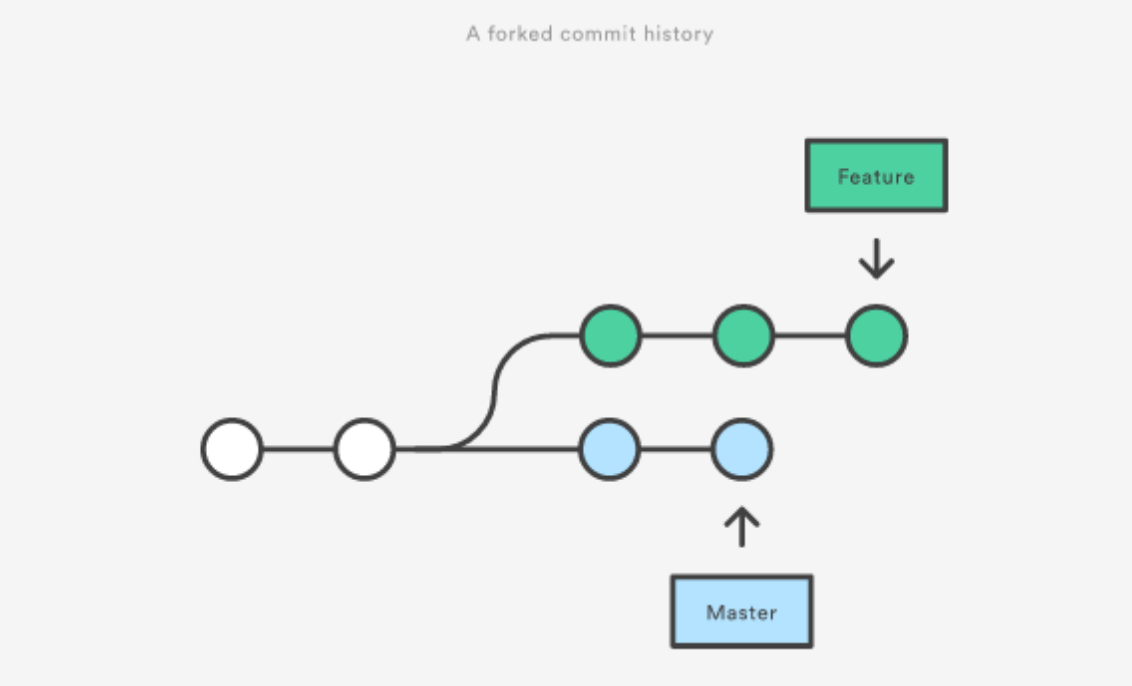
  ```
  git merge --abort
  ```

- This will roll back the attempted merge and you can retry after resolving whatever issue you had.

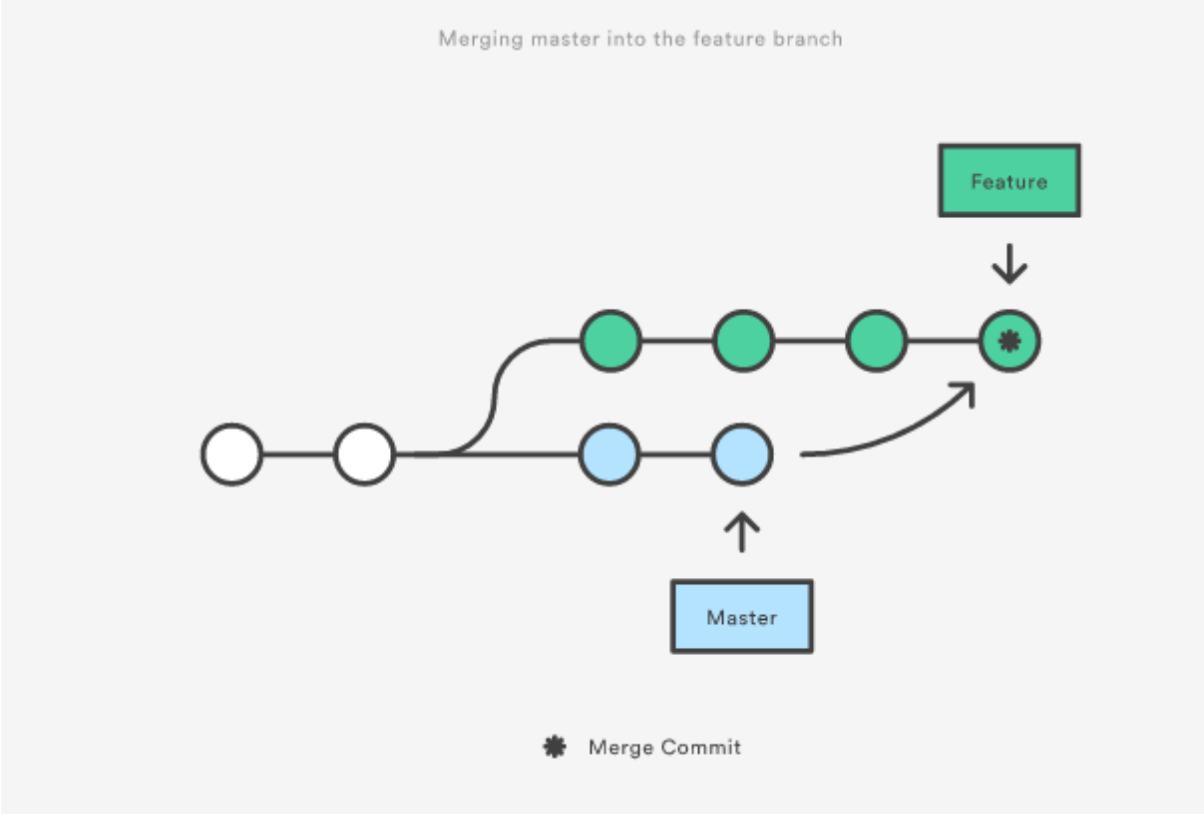
13. Rebasing

Rebasing is another way to combine your branch with the master branch.

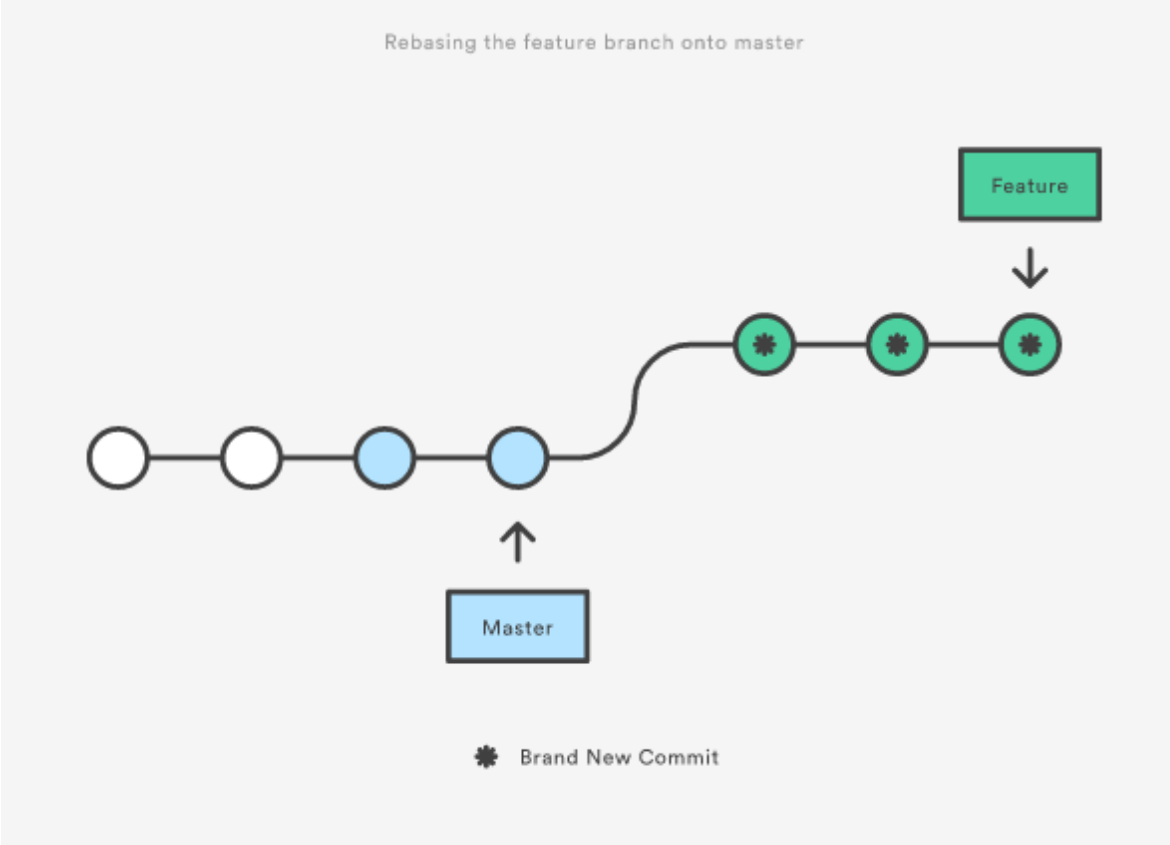
Forked Commit History



Merging


Merging master into the feature branch
Feature
Master
Merge Commit

Rebasing


Rebasing the feature branch onto master
Feature
Master
Brand New Commit

What happens in a rebase?

- Your current branch is rewound till the last branching point from the master and all your commits are stored in temp storage.
- All the commits on master since you branched off are added in the right order.
- All your old commits are added after this in the right order.

How to rebase?

This is how you rebase your feature branch onto the master branch. The order of this is important. You are rebasing YOUR branch onto the MASTER branch. This is done in the reverse way as merge.

- Checkout your feature branch (git checkout feature)
- Rebase your branch on master (git rebase master)

  At this point feature is like a branch from the latest commit on master. In this situation we can do a fast-forward merge.
- Checkout master (git checkout master)
- Do a merge which will be a fast forward merge. (git merge feature)

16. Conflicts during rebase

- In case a conflict occurs at any step, you can

  Abort the rebase git rebase --abort

  Skip the commit (very rare and dangerous to use)
  git rebase --skip

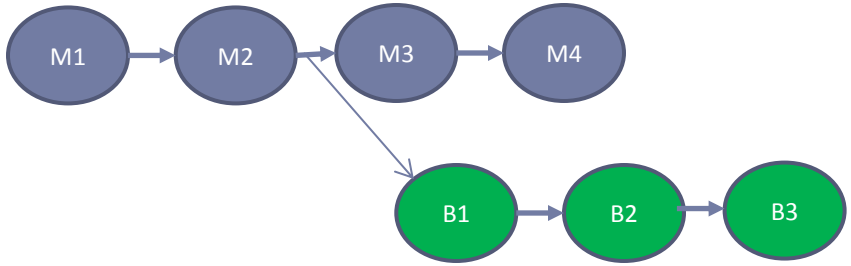
  Fix the conflict like you did during a merge conflict and then continue the rebase.
  git rebase --continue

17. Rebase Example

Make a tree like this.

Commit your master branch with commit messages M1, M2.

Then make a branch called my-branch. Make commits with messages B1, B2, B3. Checkout the master branch and make two more commits M3, M4.



Now checkout the new-branch and rebase your branch onto master.

```
git checkout my-branch
git rebase master
git hist
git checkout master
git merge my-branch
git hist
```

Difference Between merge and rebase

- Merging introduces a commit every time there is a merge while rebase doesn't.
- Rebasing allows you to have a perfectly linear history.
- Rebasing allows you to rewrite the history.
- With rebasing you can lose history while merge is safe.
- Rebasing should NEVER be done on a public branch.

When do you use a rebase usually?

- To avoid many merge commits. If the master branch is very active, and you have to keep merging, you will get a new commit for every merge. In such a case rebase would make sense.
- Rebase is also used to do a rewrite of history. If you had a lot of frequent and small commits on your branch and you want to combine them together before merging with master then rebase allows you to do that.

Other useful git commands to know

- git rebase –i master

  Interactive Rebase . Allows you to interactively change commit messages and squash commits together

- The .gitignore file

  Tells Git which files not to track.

- git stash

  Allows you to temporarily store unstaged files before doing a checkout.

Other useful git commands to know

- git reflog

  Allows you to look at how the HEAD has changed over time and various commits.

Remotes

A remote git server is useful for collaborating between multiple developers or even syncing your own work between machines.

Github and Bitbucket are two popular Git server providers.

WARNING :- The code you share on a Server may be public.

Please be very mindful of this when using github or bitbucket.

You can create private repositories but these are usually paid accounts.

Remote (Exercise)

- Make a directory called work2 (cd work2)

  git clone https://github.com/mockingbirdz/gitcourse

  cd gitcourse

  git branch

- The remote branches show as remotes/origin/<branch-name>. 'origin' is the remote alias set for the repository that you have cloned from. You can also add remotes with other aliases e.g. prod, staging, uat, test.

  git remote add prod https://github.com/mockingbirdz/prod

Remote (Pushing Changes)

- Make changes in your working directory, stage and commit.

  git add .

  git commit –m “my commit”

- One person push their changes.

  git push (Please note that you need valid credentials to push)

  This will push the current branch to origin.

- To push a specific branch to a specific remote

  git push <remote-name> <branch-name>

Remote (Pushing Changes after the remote has moved ahead)

Everyone else try pushing their changes. You will get an error

```
! [rejected]        master -> master (fetch first)
error: failed to push some refs to
'https://github.com/mockingbirdz/gitcourse.t'
hint: Updates were rejected because the remote
contains work that you do
hint: not have locally. This is usually caused by
another repository pushing
hint: to the same ref. You may want to first
integrate the remote changes
```

Syncing with the remote (rebasing)

- To get what changes have happened since your last commit

  git fetch (It fetches remote data)

  git branch –-all (Shows all the branches including the remote branches)

  git diff origin/master (Will tell you the difference between current and remote master)

- Now how do we sync it to our master? The preferred way is to rebase.

  git rebase origin/master

Syncing with the remote(merging)

- You can also merge after fetching. That will cause an additional merge commit.

  git merge origin/master

- A pull is a combination of a fetch and a merge.

  git pull

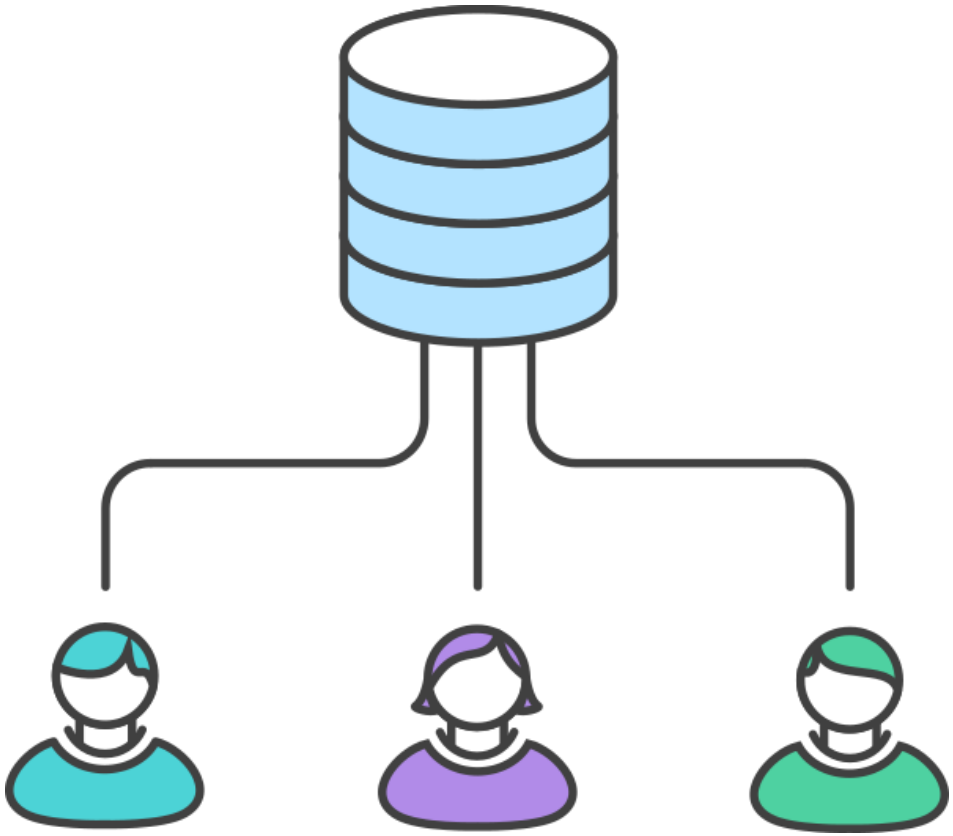
- To pull a specific branch

  git pull <remote-name> <branch-name>

Workflows

- Git allows developer teams to collaborate with each other using remote servers.
- Git also provides the flexibility to set certain workflows for collaboration.
- This next section will explore the various possible workflows.

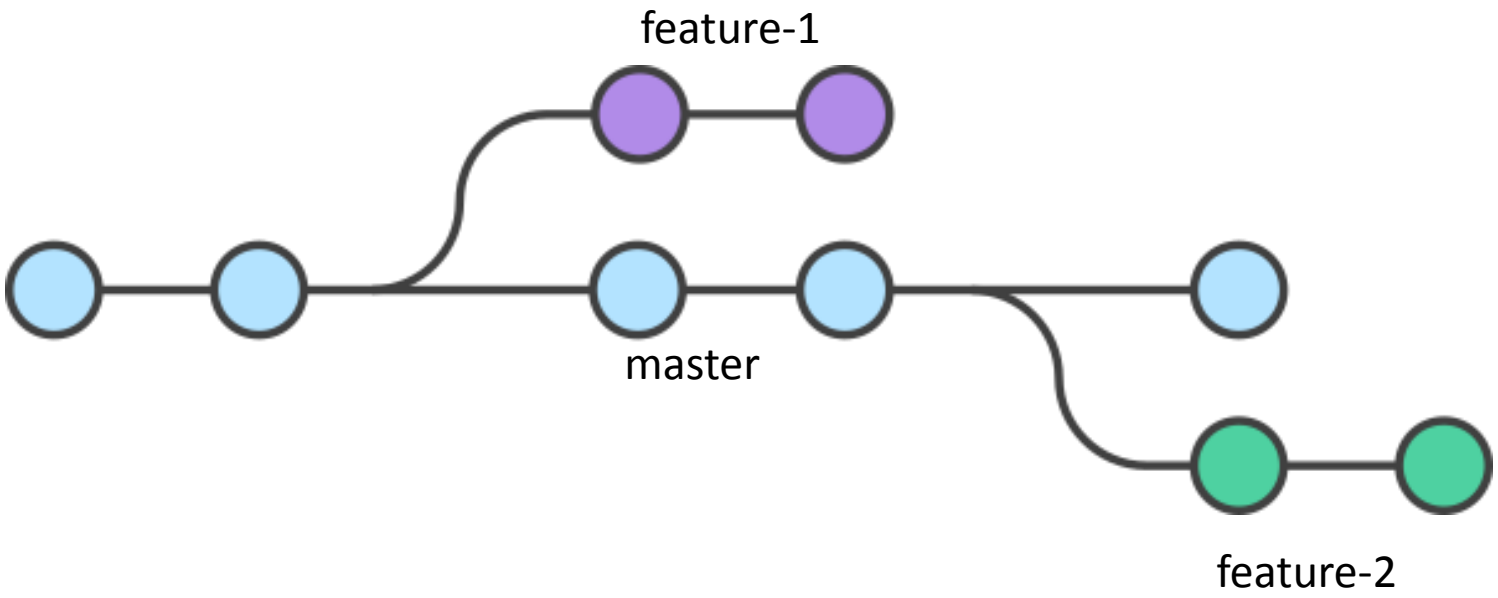
Centralized Workflow (diagram)



Centralized Workflow

- Each developer clones from the central repository.
- Each developer develops and commits locally.
- Each developer pushes to server.
- In case the central repository has moved ahead, the developer will pull or rebase and then push.
- Similar to SVN.
- Everyone has equal rights on the master repository.

Feature Branch Workflow



Feature Branch Workflow

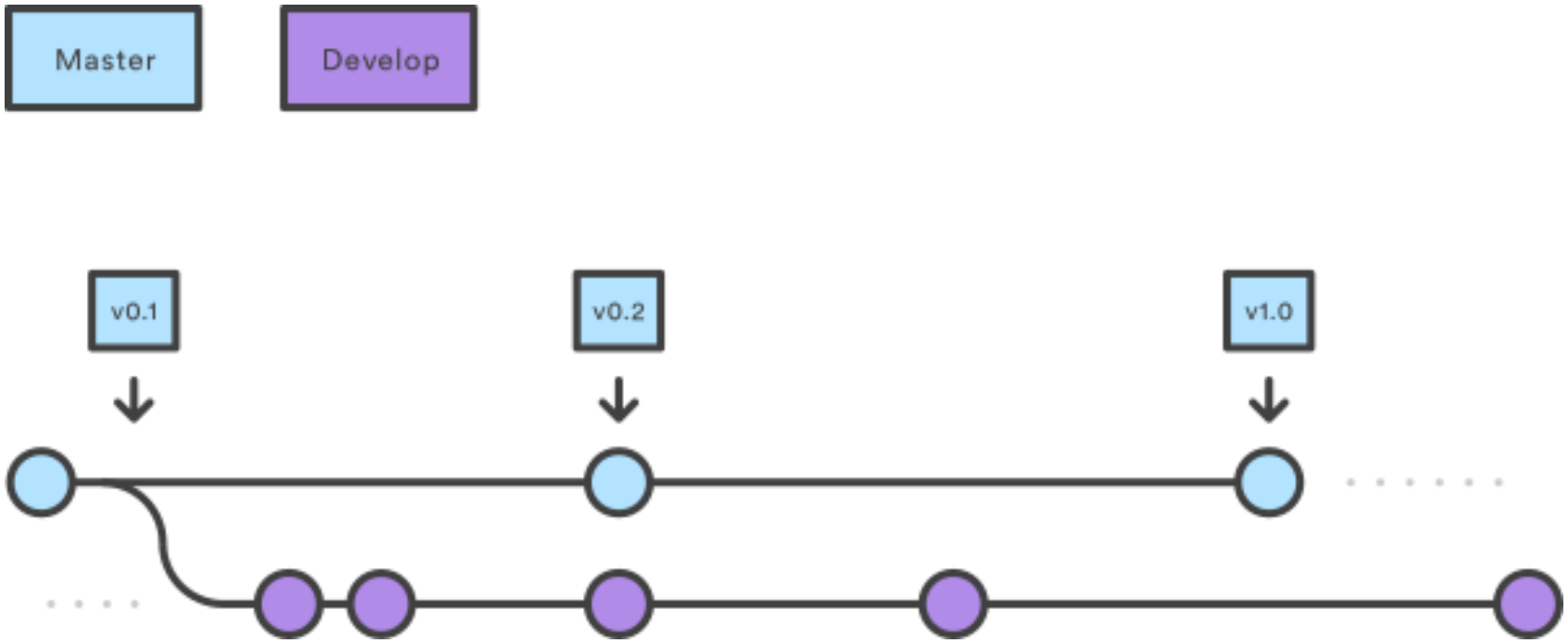
1) The master branch is the official branch
2) Every feature is developed on its own separate branch
3) Once a branch development is complete and the code is stable, it becomes merged into master
4) Merging is usually done with a pull request.

Try out a pull request.

Make a branch in your code. Make changes, stage and commit on the branch and push the branch.

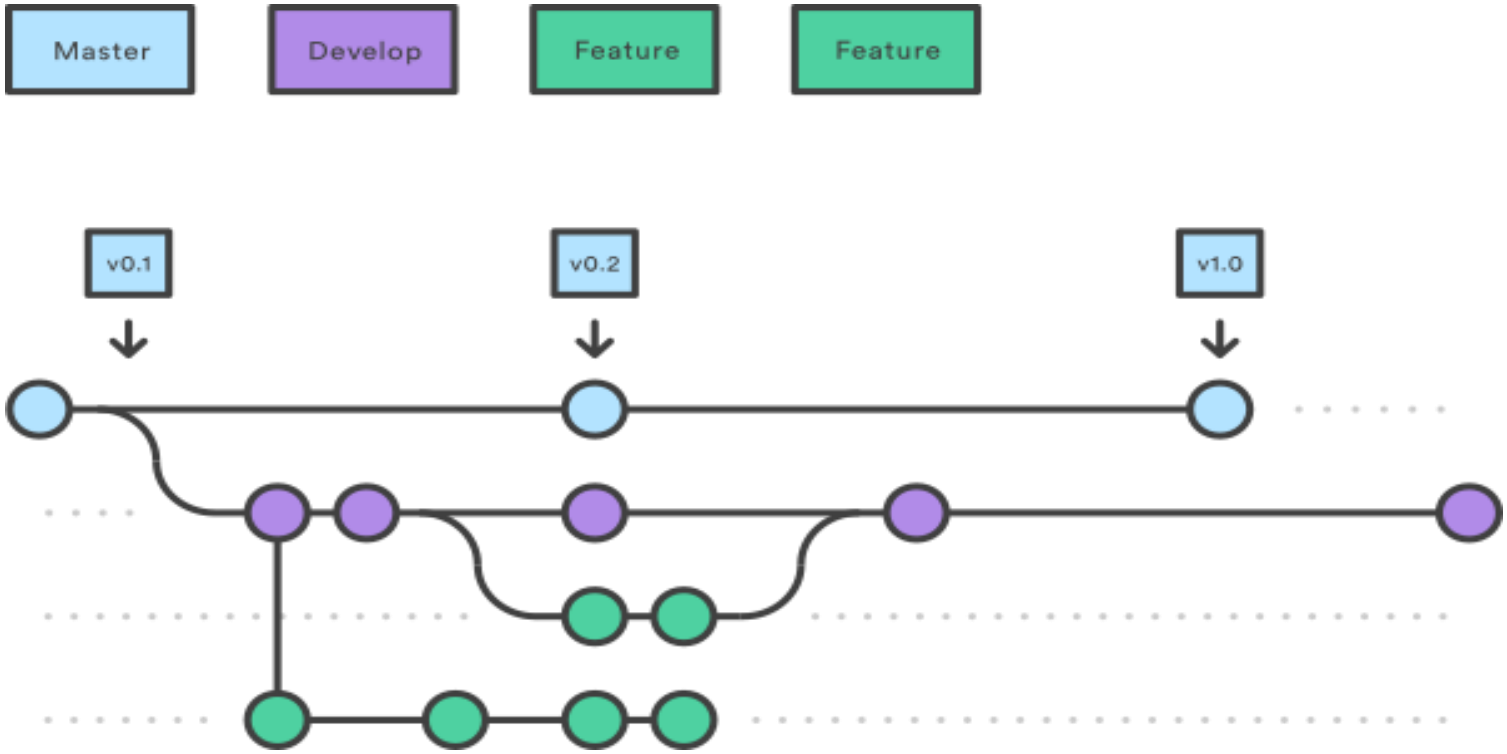
Then make a pull request on Github.

Gitflow Workflow



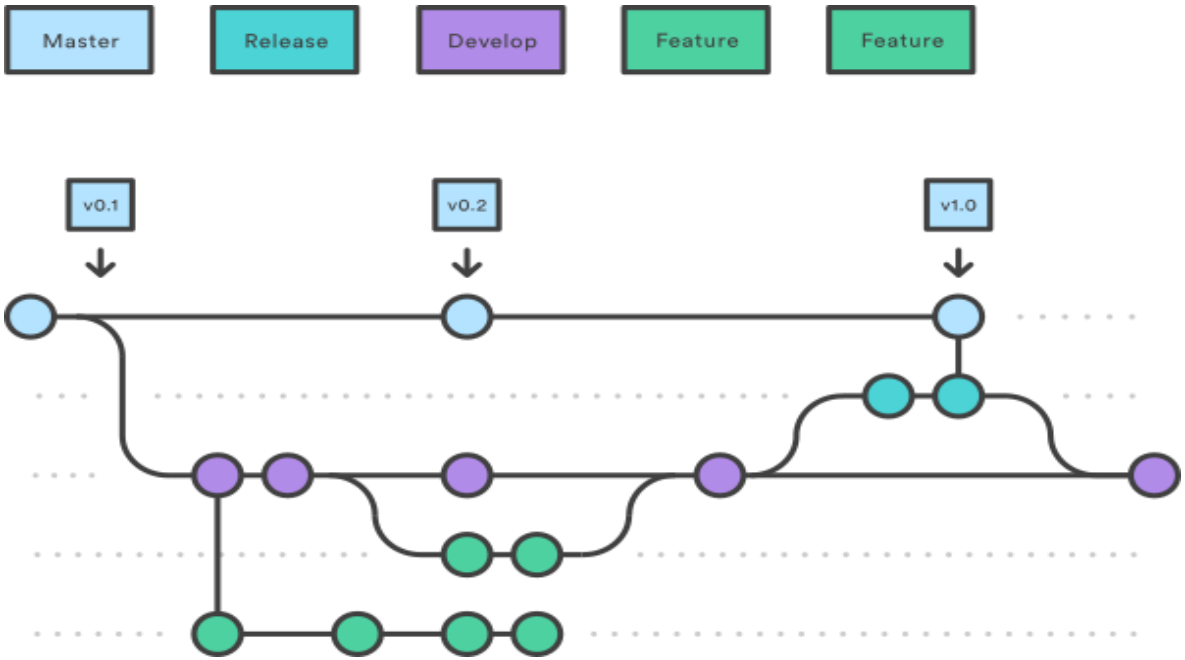
The prod branch is the master. Development happens on a release branch.

Gitflow Workflow



All features are developed by branching off the feature branch and merging back once development is complete.

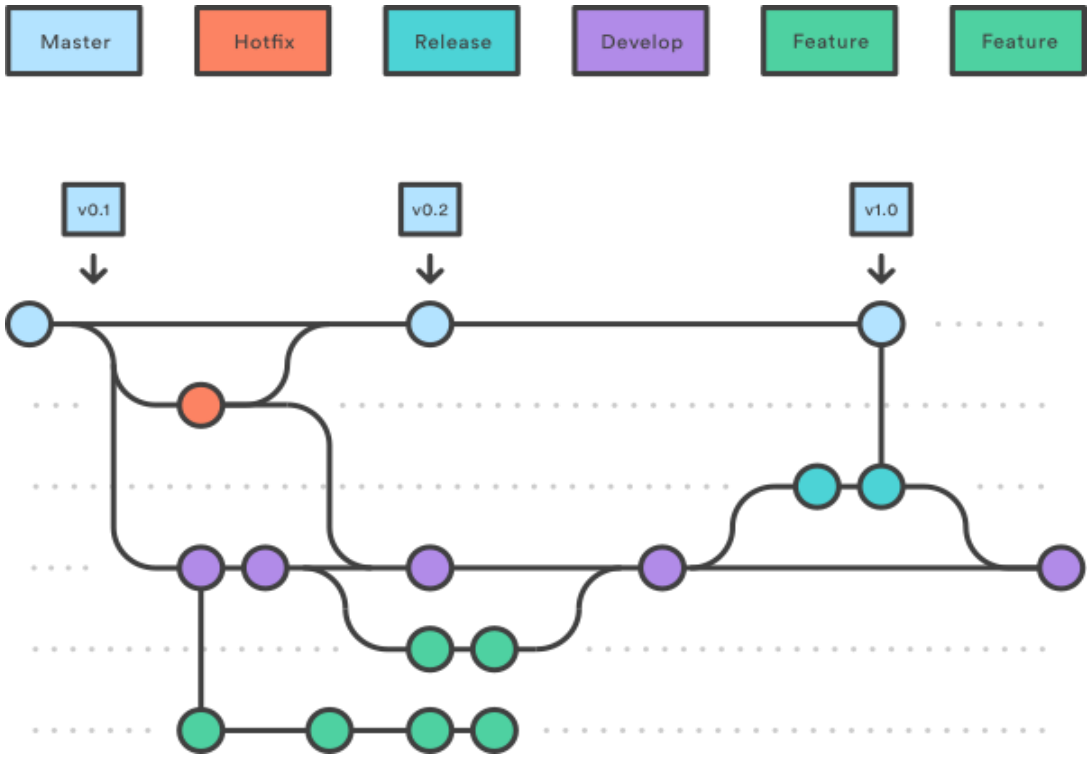
Gitflow Workflow



Once the development for the release is complete (either development is complete or date is reached), then release branch is merged back into master.

For the next release, a new branch is again created from master.

Gitflow Workflow



Any hotfixes (urgent patches) are developed by branching off master and then merging back. These are also merged into the release branch.

35. Useful Resources

Pro-Git Book
https://git-scm.com/book/en/v2
Atlassian's Git tutorial
https://www.atlassian.com/git/tutorials/
Github's Interactive Tutorial
https://try.github.io/
A guide for the perplexed
http://think-like-a-git.net/
Git Immersion
http://gitimmersion.com/
Git from the bottom up (a look at exactly how Git works)
http://ftp.newartisans.com/pub/git.from.bottom.up.pdf
Linus Torvald's TechTalk at Google on Git (purely for entertainment)
https://www.youtube.com/watch?v=4XpnKHJAok8