The Learning House

427 S. 4th Street #300

Louisville KY 40202

Introduction to Git

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Git

Git is a *distributed revision control* and *source control management (SCM)* system.

Created in 2005 by Linus Torvalds (Linux).

The goal is to handle large projects by multiple developers who may be in the same physical location.

Why Git?

- Very popular as Linux, many open source projects and much more make use of it.
- Relatively easy as it's file system based
- Can work completely offline and then sync with a remote repository when back online

File System Based

- Set a directory as the repository (repo)
- Git watches changes to this repository
- Create new save points locally and then push the to the remote when connected
- Git merge, rebase and branching is easy!

A user commits to their repository initially. This creates their commit ‘1’. After adding two more commits their repository looks like the following

If the user is using a remote repository they can then push to that repository

GitHub

Push to a remote repo like GitHub

Local

With a remote here is a diagram of Git

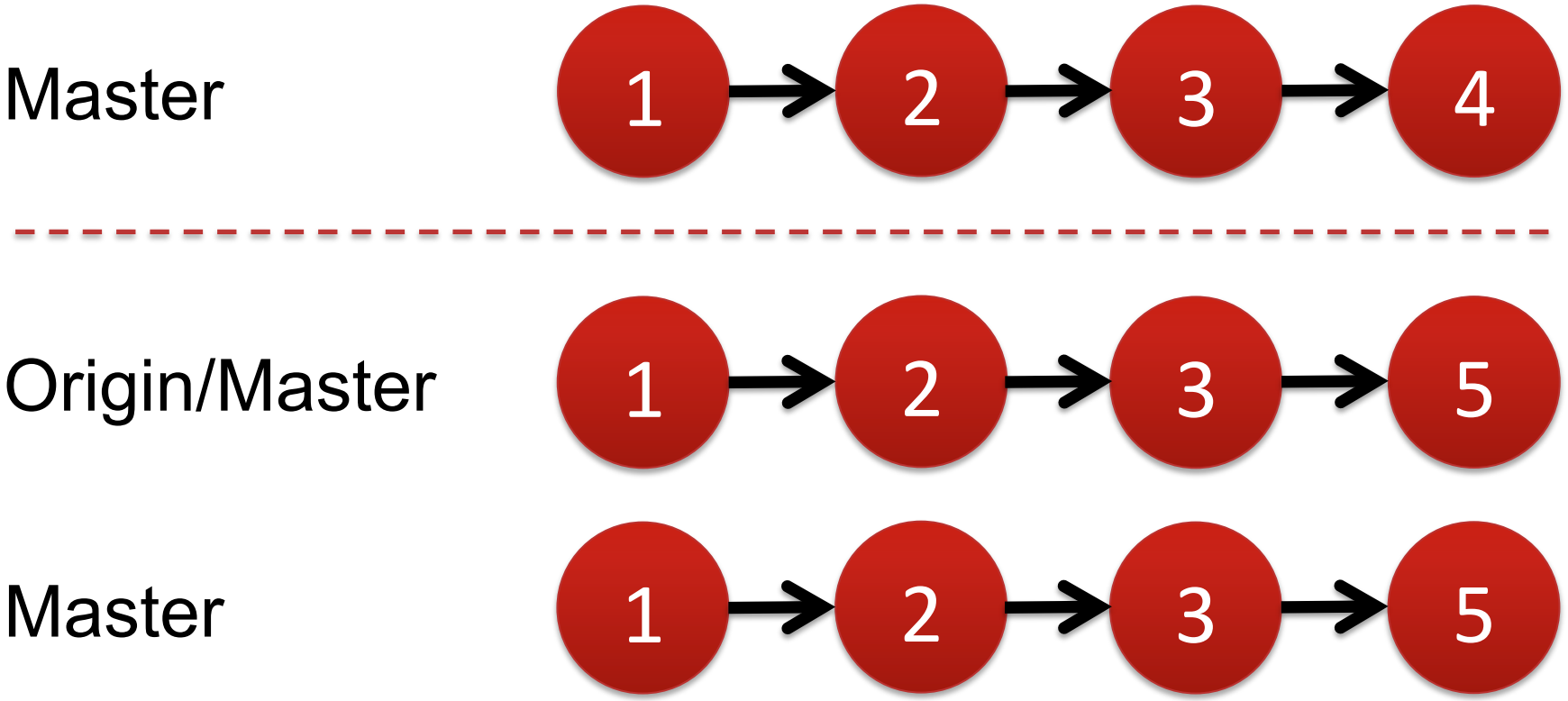
Master

Origin/Master

Master

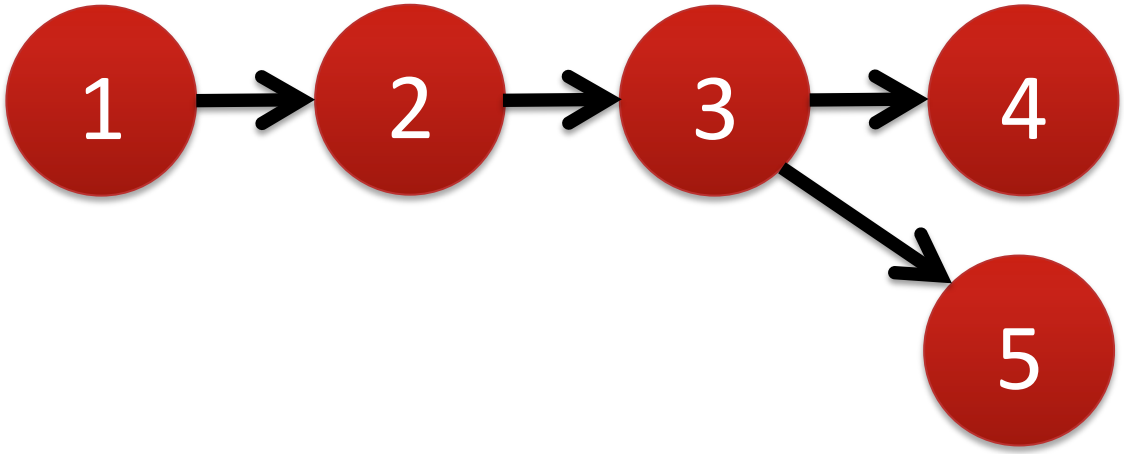
Master is the default branch, Origin is a copy of the remote

WHAT HAPPENS WHEN THE REMOTE AND LOCAL ARE OUT OF SYNC?



This can happen when a second developer commits their changes before you have pushed your changes up to the remote.

Attempting to pull down the changes will leave the repository looking like this.



Resolving Divergence

- Both commits 4 & 5 are based on commit 3
- We must merge 4 & 5 together so that we have one line of code to move forward.
- In the event that both 4 & 5 make changes to the same file we will have to resolve the conflict first.

Merge Conflicts

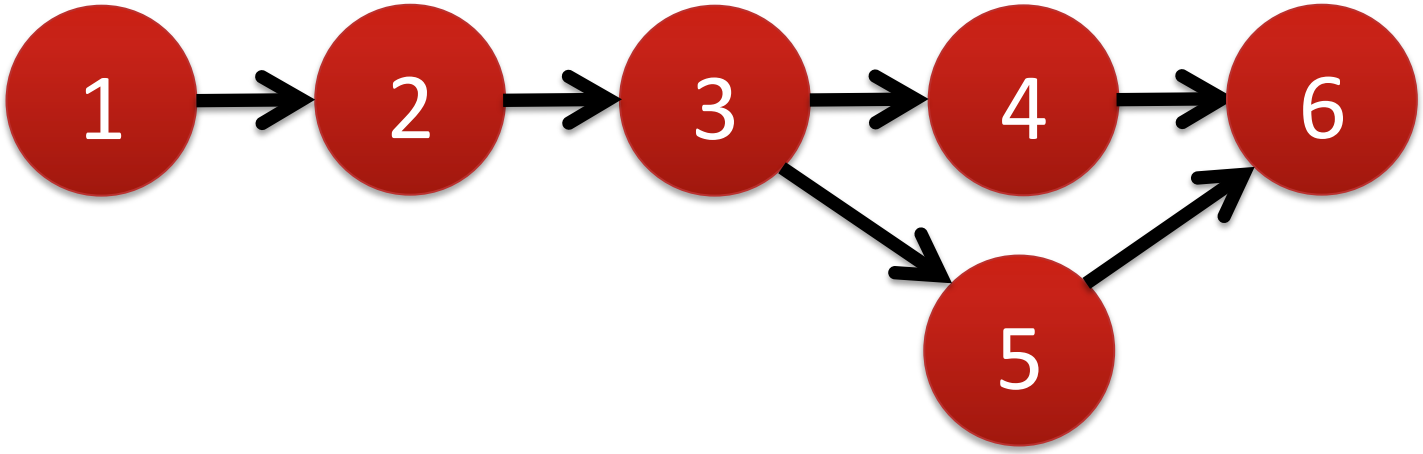
- All files with a conflict will contain the following text.

```
<<<<<<<<<<
OURS
==========
THEIRS
>>>>>>>>>>
```

Remove < = > lines
Fix the code to a working version merging the changes or choosing one or the other to move forward with.

The Result is …

A new commit will be required for the merge



6 is a merge commit

That's Merging!

- That's it, that's all there is to it!

- If you don't have any conflicts the merge will complete successful and create the merge commit.

Merge Hell

- This happens when many changes to the same files have happened. Then you have to resolve all conflicts before continuing.

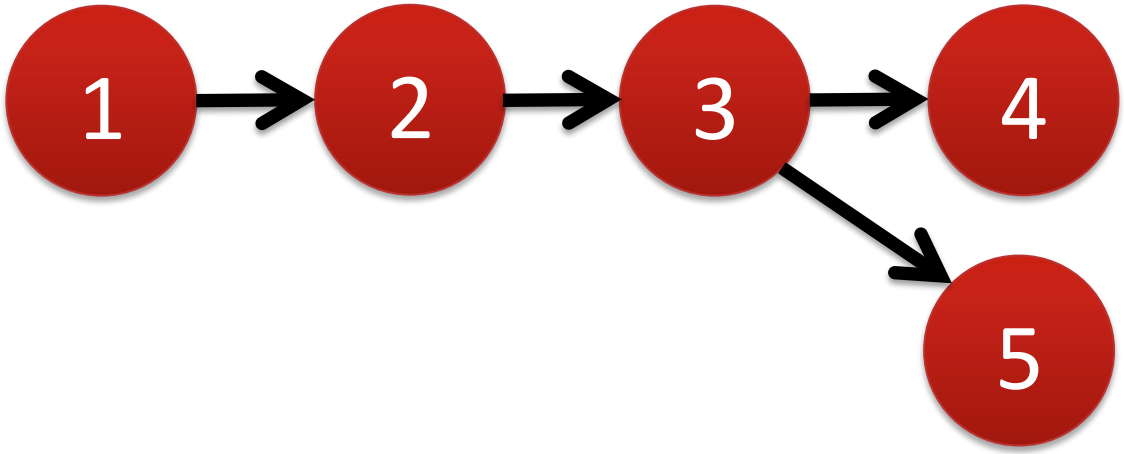
- This is also a possibility when a branch exists for an extended period of time without merging into a main branch of development

WHAT'S REBASE?

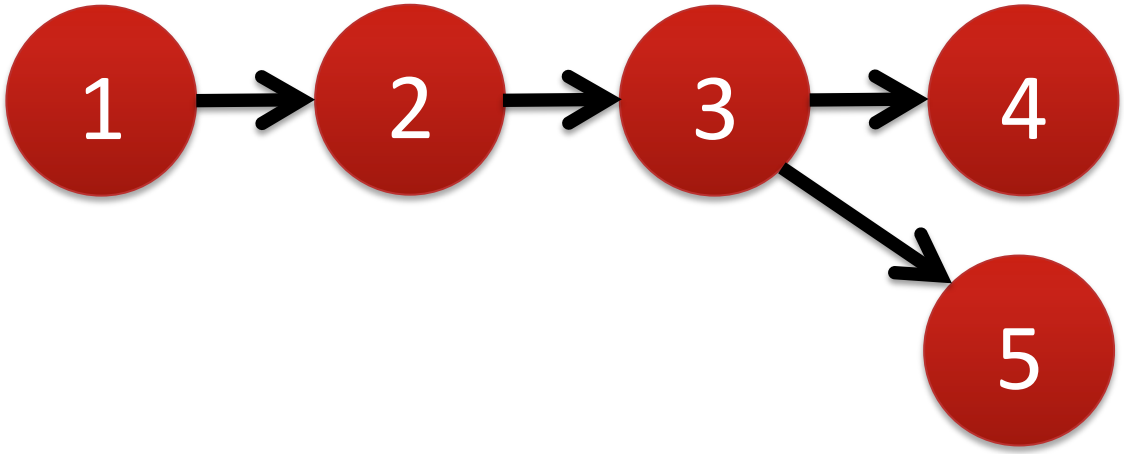
Rebase

- Rebase is alternative to merge
- Here we will use Merge
- Still important that you understand rebase

At the state where you are pulling changes you can use a rebase instead of merge to modify changes.



A rebase will replay the changes in commit 5 on top of commit 4 instead of being based on 3



What's happening?

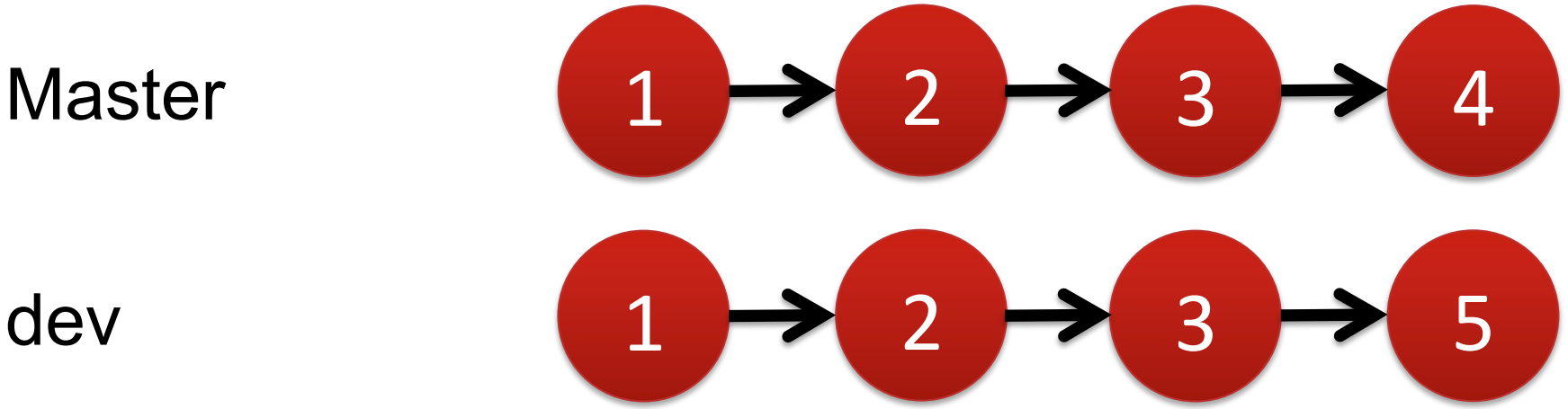
- Rebase changes the base of the commit
- The history of the commit is lost
- A linear history is re-established by rebase where merge preserves the entire history.

BRANCHING

What is a branch?

- A branch is a separate line of development from the main branch.
- A branch does not have to be added to the remote repositories
- You can create as many branches as you need for your workflow

Branching may look familiar if you think of it like this..



In this case we have a commit on Master that we didn't have on our private branch ('dev')

Why Branch

- Isolate your development. Fix issues, Work on new releases and experiment all in isolation without affecting production code.
- If a branch fails and you decide you no longer need the branch, simply delete it
- If you need the branch merge it back to master just as you would with a remote and master and origin/master

Branching Best Practices

- Follow company specified workflow
- Don’t delete branches you pushed to a remote repo
- Name branches with descripting names
  - iss123 – a branch for issue 123
  - jsdev – a branch for John Smith’s development

MANAGING A REPOSITORY

Git is file-based

- Use a folder to hold all repositories you have on your machine. The parent folder shouldn't be a repo but each sub folder will be.

Ex:

_repos

DevelopmentProject1

DevelopmentProject2

Each Development Project is a separate repo

A few notes:

- All files in the repo directory will be tracked for changes by default
- You can use a .gitignore files to ignore files or directories from a repo.
- Files committed will be tracked unless removed from the repository even if .gitignore should allow file to be ignore.

.gitignore

- Files that goes at the root folder of a repo. i.e. in the repo folder
- Include files and folders
- Best to create before first commit
- Will be added to repo like any other file

Git Commands

Basic Commands

- git add
- git commit
- git init
- git clone
- git push
- git pull

Others to Know

- git status
- git log
- git fetch
- git merge

Initial Setup Commands

git init will create a repository from scratch

git clone will copy a repository from a remote location and create a local copy

File Focused Commands

- ***git add*** can stage any modifications, deletions or new files. git add is the only way to stage new files…

- ***git commit*** will create a new save point in the repo. You can combine git add and git commit using the '-a' option but only for modifications and deletions.

Helpful Commands

- ***git status*** will provide the current status of the repository as it relates to the current branch and any remotes.

- ***git log*** can be used to view the history of the repository. Command line arguments can be used to show it in a graph with all branch positions present.

```
git log –oneline –graph --decorate
```

Remote Interaction Commands

- ***git push*** will attempt to send your changes to the remote repository

- ***git pull*** will run a ***git fetch*** and then either a ***git merge*** or ***git rebase***.

Remote Commands continued

- ***git fetch*** retrieves the changes from the remote repository updating the local copy of the remote without changing files in the working directory

- ***git merge*** will create a merge commit converging two varying lines of development

- ***git rebase*** will replay changes of one branch on the HEAD of another branch

Conclusion

- Git is easy, file-based source control
- Don’t be afraid of merging
- Remember to run commands in order
  - $ git add
  - $ git commit
  - $ git pull
  - $ git push
- Run git status frequently!!!

A Few Things to Note

- Here we will stick to master. This is the case with smaller projects with few developers
- Branches are for those projects where multiple developers constantly developing are required
- Here we will perform merging. It is more important to keep history.
- Rebase may be used on production code branches to keep history linear

Remote Repositories

- GitHub – used in the prework. Hosted git that works well for open source/public repositories. Not ideal for private repos.

- BitBucket – like GitHub is hosted but licensing model allows for private repositories for up to 5 developers free.

CREATE A BITBUCKET ACCOUNT

DEMO – GIT WALKTHROUGH

EXERCISE – PAIRED MERGE