GitHub Workflow Basics

Laying the foundation for a strong, team-based workflow

Published by Andrew McCall

CIG eLearning
Capital Insurance Group

Dependencies

GitHub account

Sign Up at: https://github.com/

Git for Windows

Download at: https://git-scm.com/download/win

SourceTree

Optional, but recommended: https://www.sourcetreeapp.com/

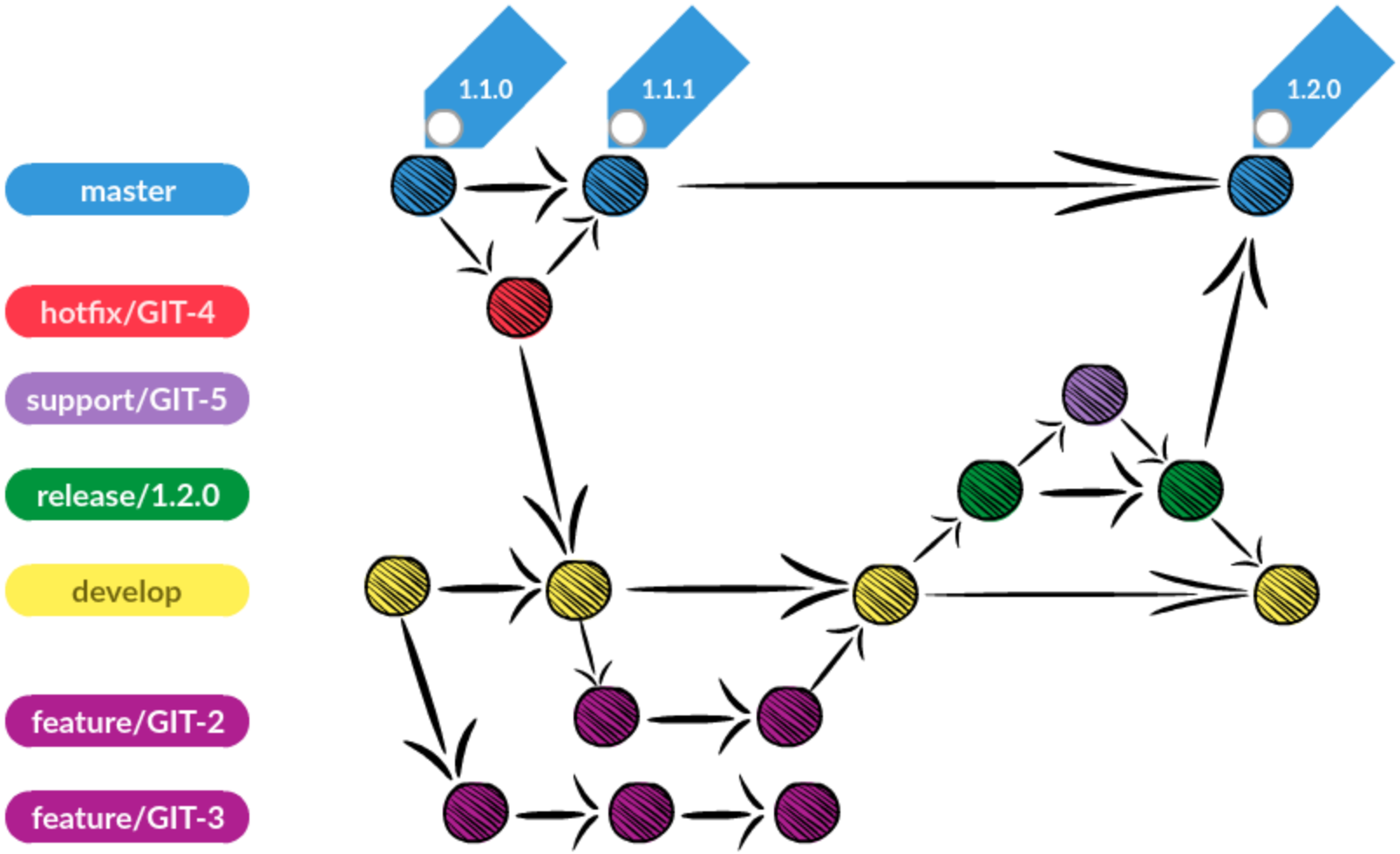
Repository

Our working repository will be here:
https://github.com/CIGInsurance/DemoForGithub

The Overview

- What will we be doing today?
- Why Github and VCS overview
- Workflow introduction
- Workshop
- Application to new projects
- Application to existing projects

What are we going to do?

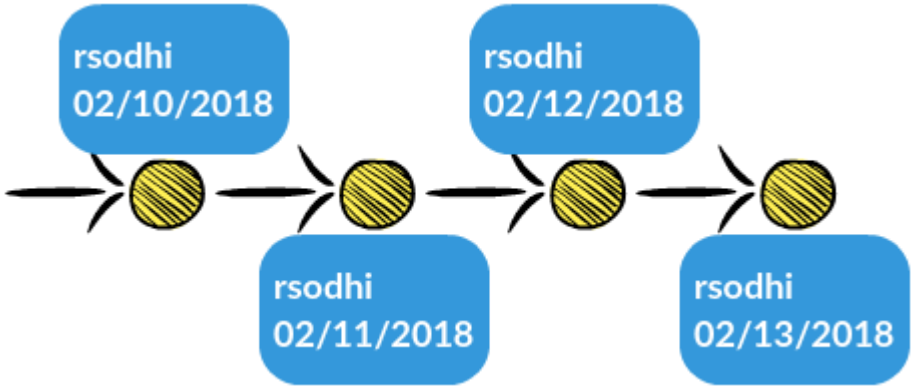


What is a VCS?

- A way to store code and track changes
- Each revision is associated with a timestamp and the person making the change

Why Use a VCS?

Change History

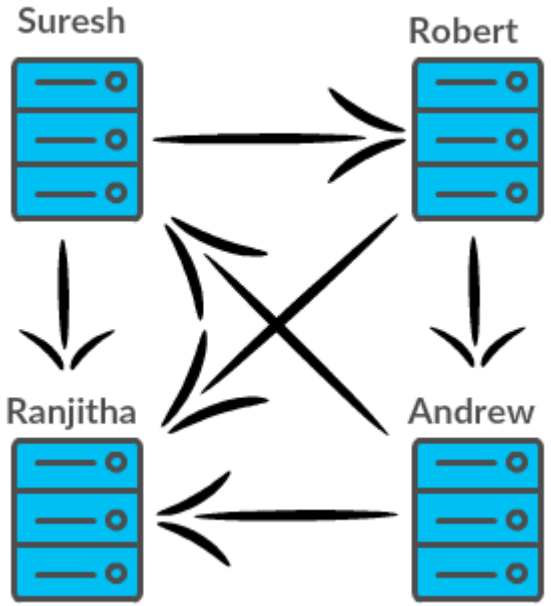


What is Git?

From the Git Website

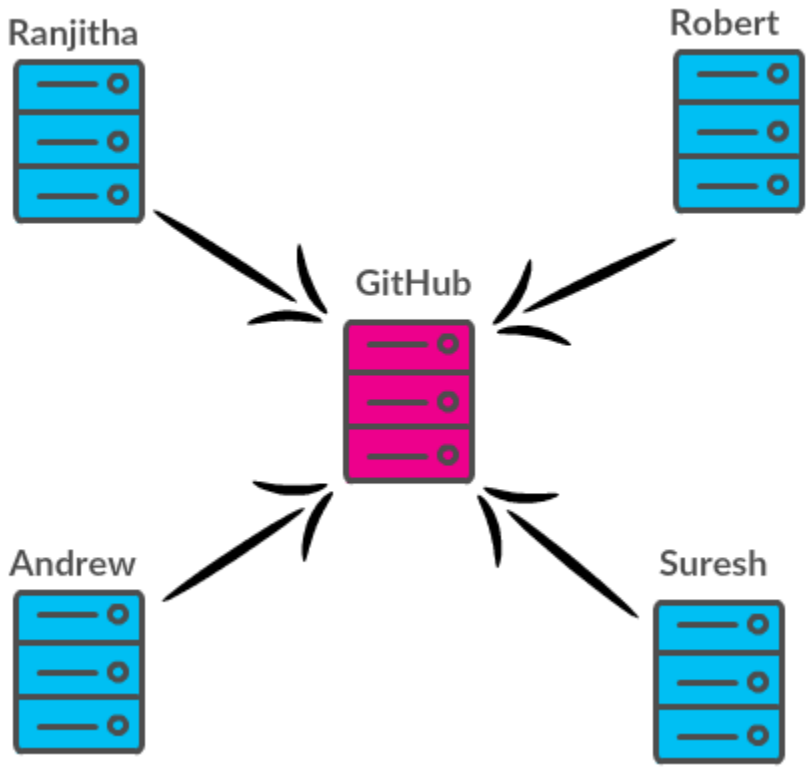
"Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

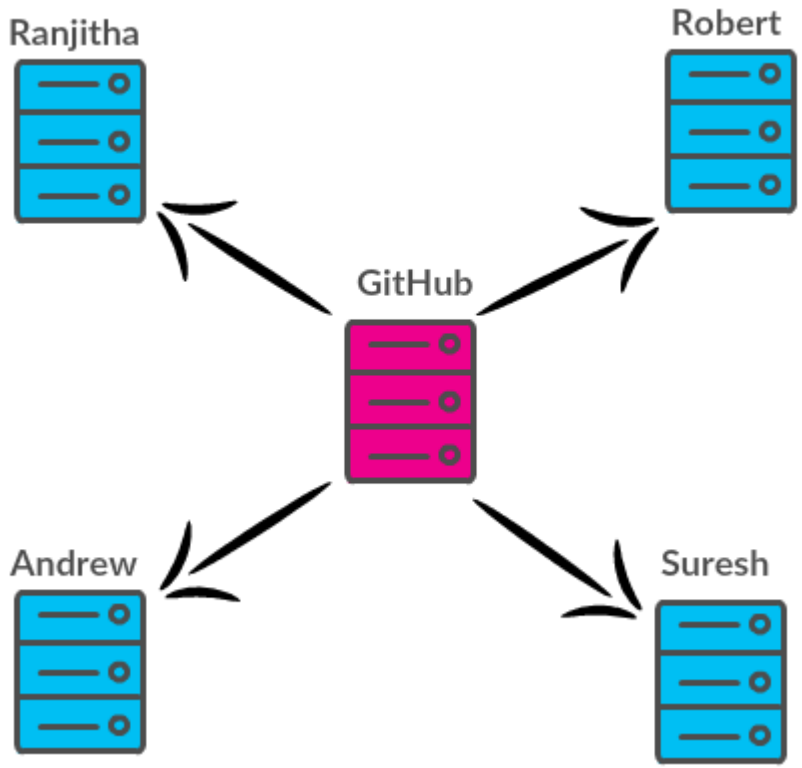
Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows."



What is GitHub?

- A collaborative tools used by teams, organizations and individuals
- A central repository for the code





Why Git?

- Distributed
- Free
- Lightweight
- Enables accountability
- Allows team to self-regulate
- Workflow
- Workflow
- Workflow

Why GitHub?

- Collaborative
- Secure
- Popular
- Extensible
- We are already using it

Git Flow

- A method of branching using Git
- Tools supported
- A standard that teams can follow
- Safe
- Scalable
- Adaptable to Pull Requests

Git Fundamentals - Commit

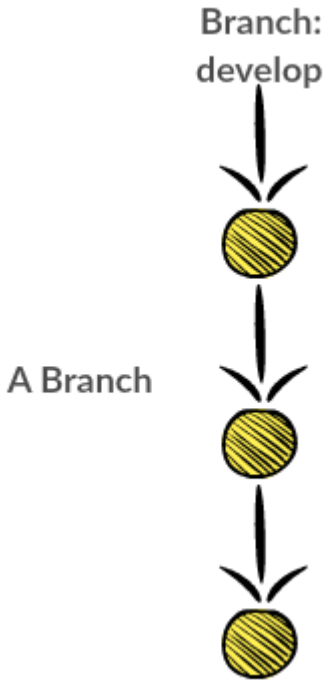
Commit - a small bundle of changes to the code

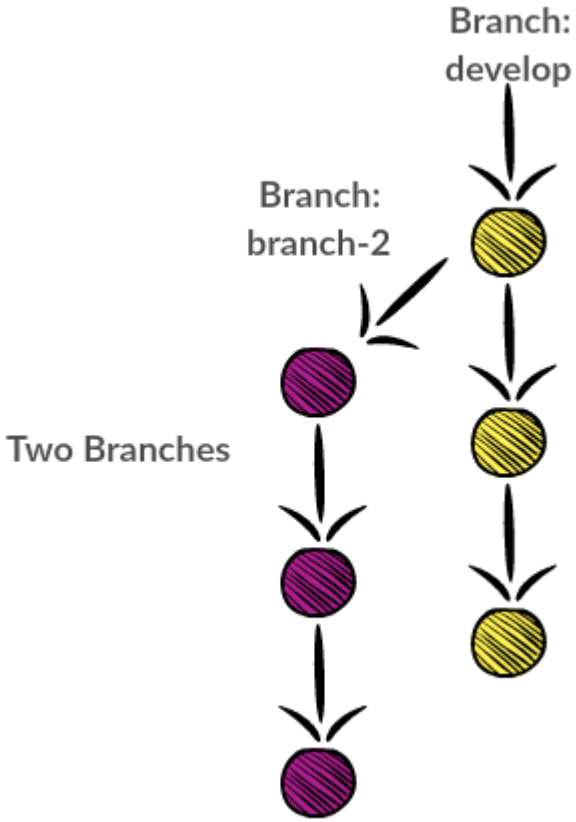
A Commit

Commit Message :	"I fixed a thing"
Commiter :	Andrew McCall < amccall@ciginsurance.com>
Commit Date :	Tue Feb 13 12:03:10 2018 +0800
Author :	Andrew McCall < amccall@ciginsurance.com>
Author Date :	Tue Feb 13 12:03:10 2018 +0800
SHA1 Hash :	9c435a86e664be00db0d973e981425e4a3ef3f8d
Parents :	[0d973e9c4353ef3f8ddb98a86e664be001425e4a]

Git Fundamentals - Branch

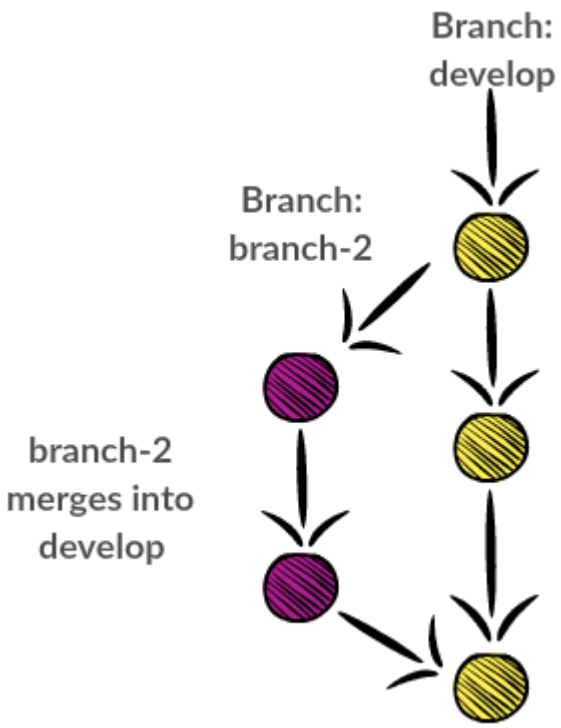
Branch - a sequence of commits





Git Fundamentals - Merge

Merging - the process of incorporating one branch into another



Branching Strategy – Git Flow

Two perpetual branches – Always exist for the life of the project

- **develop** – bleeding edge features
- **master** – Current production code

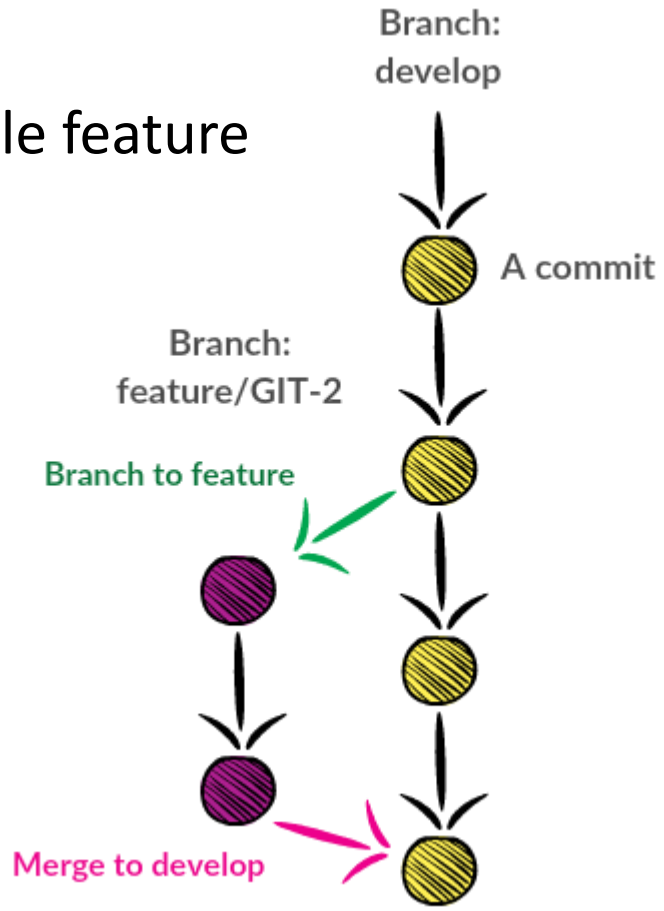
Four temporary branches – Used and deleted

- **Feature** – Used to create new features
- **Release** – Used to stage a release
- **Support** – Used to support a release
- **Hotfix** – Used to fix high-priority production issues

Branching Strategy - Feature

Feature branches

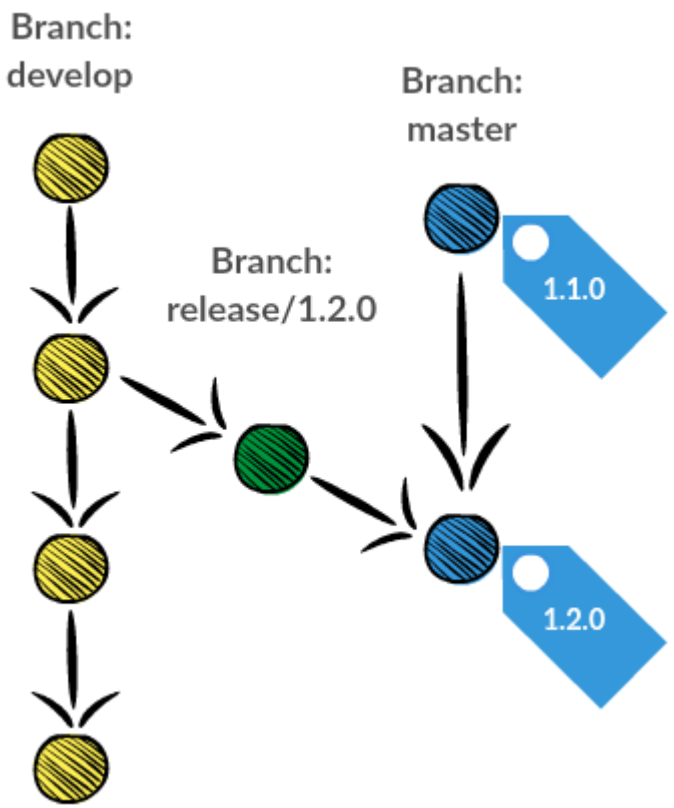
- Branch from develop, merge into develop
- Are small, ephemeral branches that contain a single feature
- “feature/[jira number] [brief description]”
  - i.e. “feature/GIT-2 a new feature here”



Branching Strategy - Release

Release branches

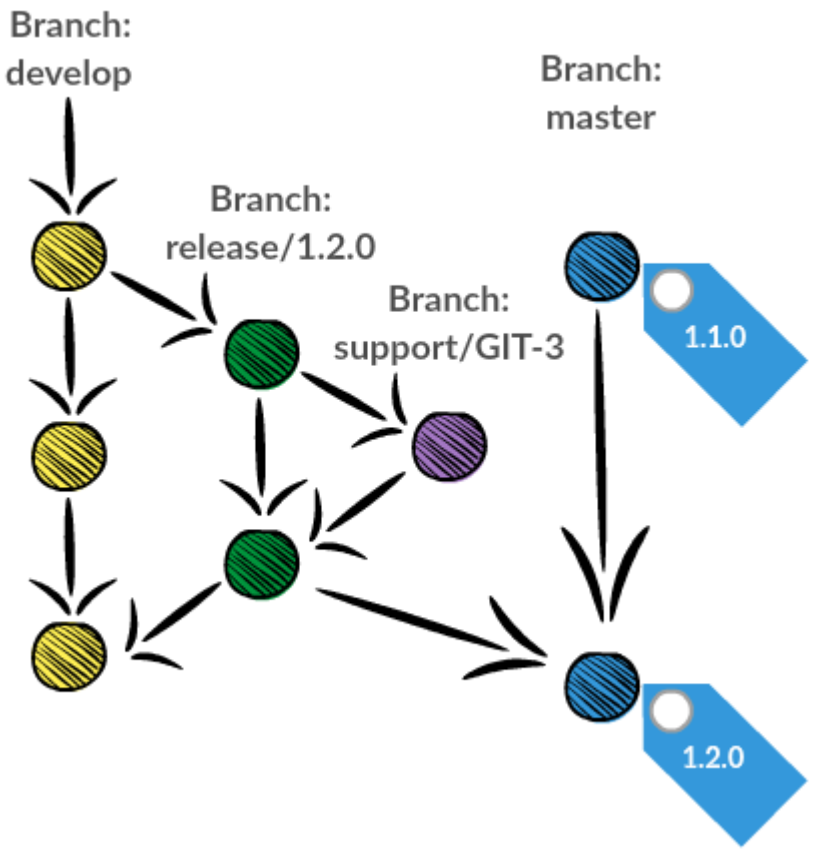
- Branch from develop, merge into master and develop
- Are created when a release to production is ready.
- Allows work to continue on develop branch
- “release/[release version]” i.e. “release/1.2.0”



Branching Strategy - Support

Support branches

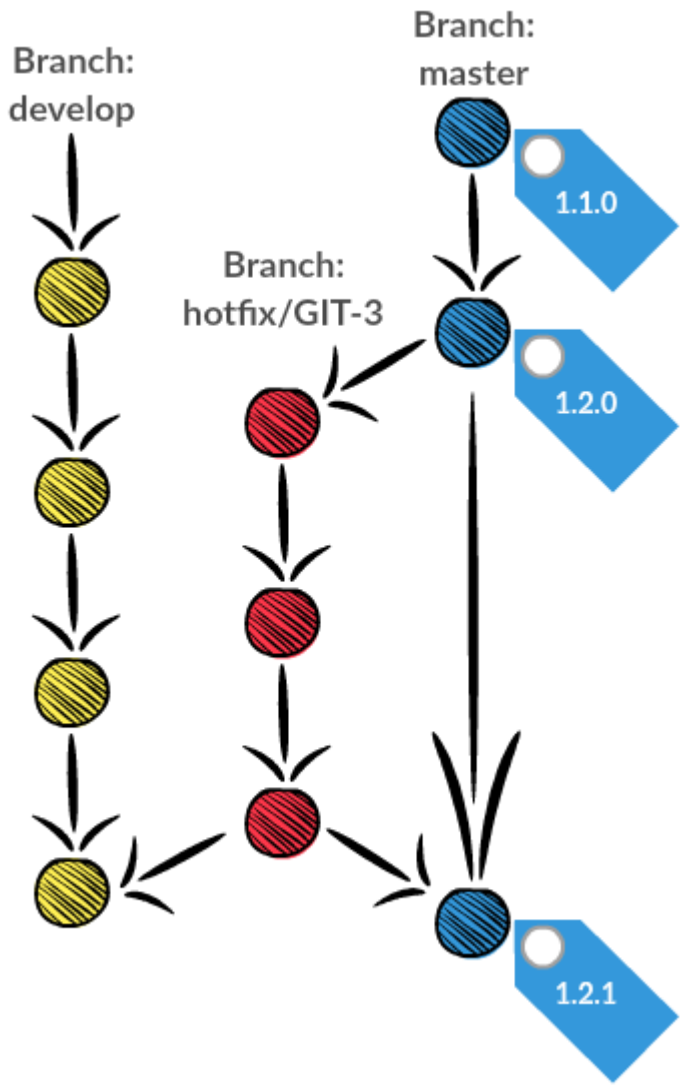
- Branch from release, merge into release
- Are created when release code requires bug fixes
- Allows work to continue on develop without delay
- “support/[Jira Number]” i.e. “support/GIT-3”



Branching Strategy - Hotfix

Hotfix branches

- Branch from master, merge into master and develop
- Are created when production code requires bug fixes
- Allows work to continue on develop without delay
- “hotfix/[Jira Number]” i.e. “hotfix/CIGTS-20154”

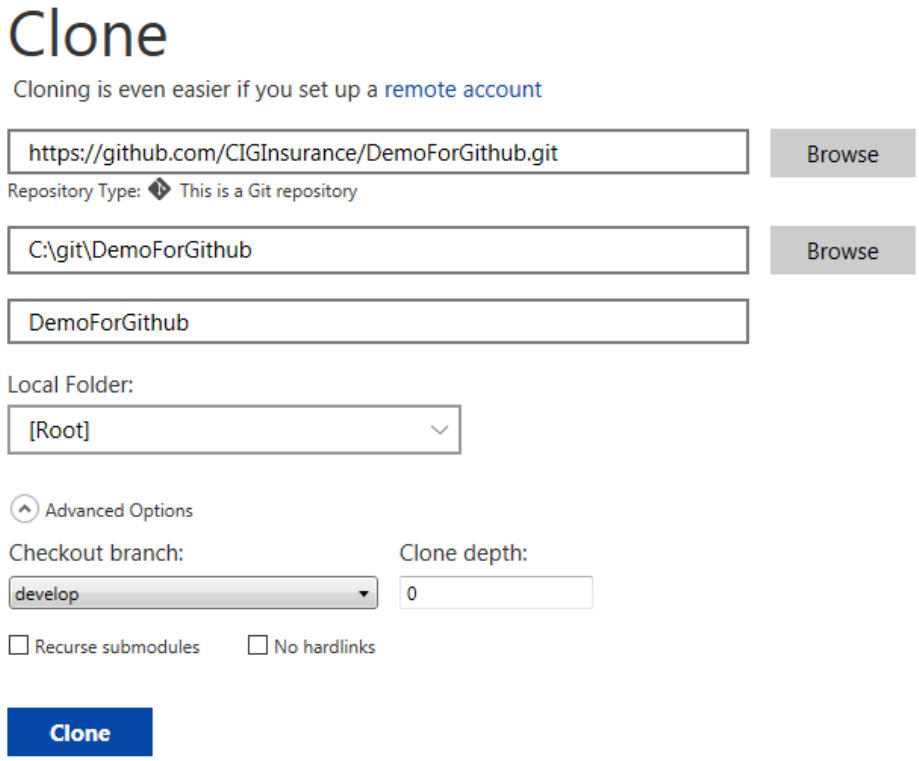


Workshop Overview

- Setup your environment
- Grab an issue from Jira
- Create a feature branch for the Jira issue
- Resolve the Jira
- Create a Pull Request, get approval, and merge the feature branch
- Create a release
- Fix a release bug
- Deploy a release
- Fix a production bug
- Deploy the hotfix

Install and Setup

- Download and install the dependencies if you have not done so
- Clone the Demo for the Github Project
  - Make some folder that will house Git repos
    - Personally, I use C:\git



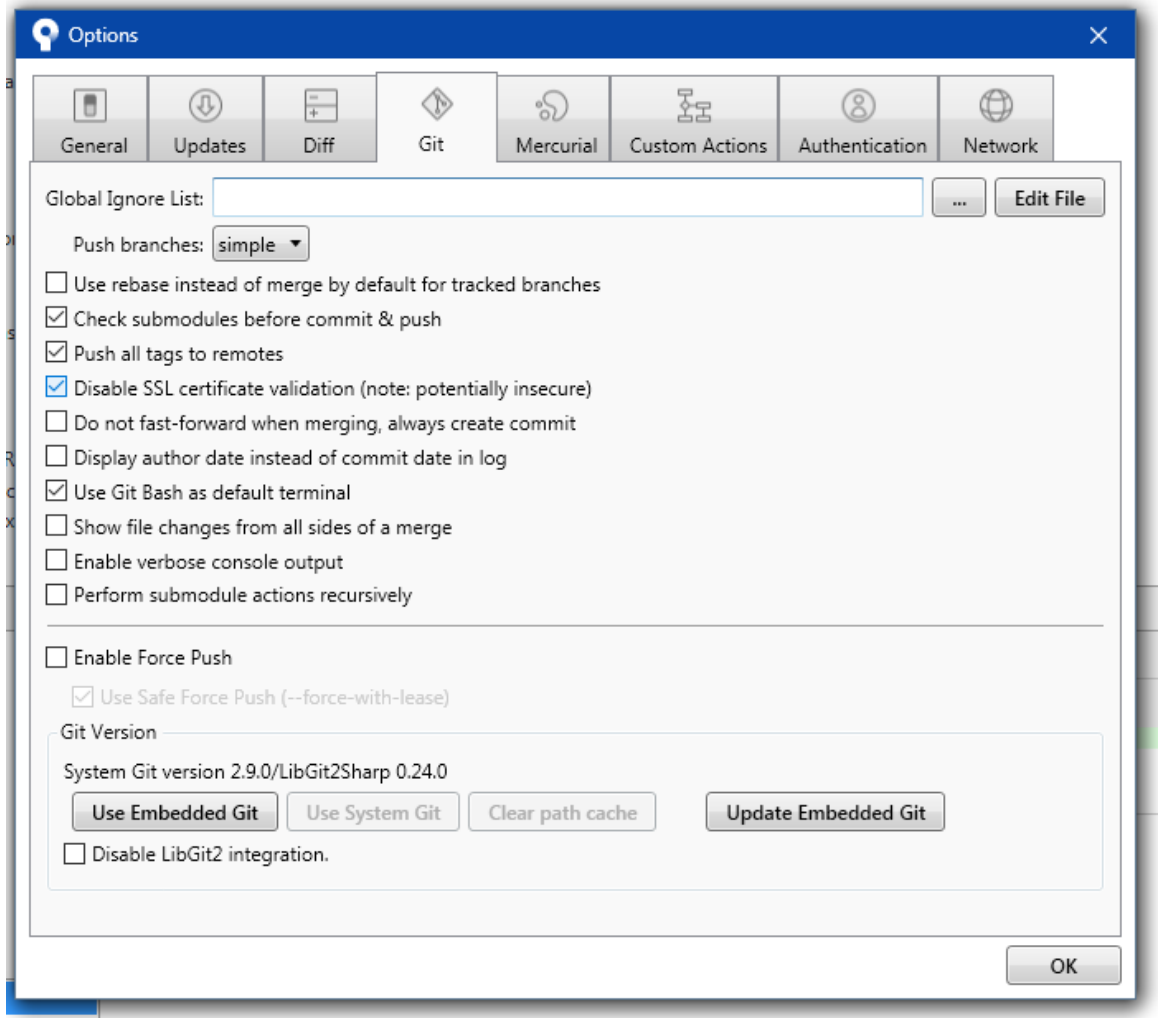
Install and Setup – SSL Issues

The Error

Anything SSL, CA Related

The Solution

Tools -> Options



Grab a Jira

- We have a project!
  - https://ciginsurance.atlassian.net/secure/RapidBoard.jspa?projectKey=GIT&rapidView=306&view=planning
- Assign a Jira from the backlog, that was previously unassigned, to yourself
- Put the Jira into “In Progress”

Create a feature branch

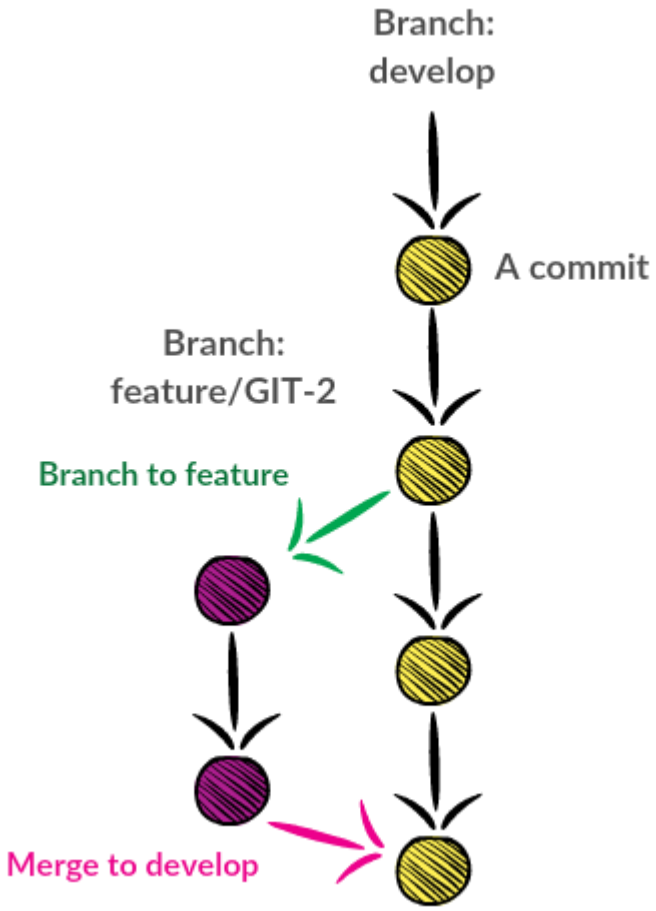
Naming Convention

feature/[Jira number] [Description]

i.e “feature/GIT-2 Help improve accuracy”

Notes

Branch from the “develop” branch, not master.



Completing the Jira

Verify

Be sure that your current working branch is the feature branch!

Add File

Make a file named *[Jira number].txt*

i.e. *GIT-2.txt*

Commit

Create your commit in SourceTree

- Stage the file to the commit
- Add a message and commit

Push the commit to github

Validate

Check out the Jira, it is aware of the new branch

Create a Pull Request

On Github

New pull request

- **Base** - develop
- **Compare** - your feature branch

Add a title

be sure that it starts with the Jira number

i.e. “GIT-2 Added my new file”

Add a comment

Should include with a brief description of the feature

“Create Pull Request”

You’re going to get red – This is a good thing!

Asking for a PR Review

Reach out to someone and ask them to accept your PR

Hipchat is the best way to do this

Include

- Polite words of please and thank you
- The URL to the PR:
  - Like: https://github.com/CIGInsurance/DemoForGithub/pull/15

Accepting a PR

An important responsibility!

- Carefully review the code that is being merged
- Point out any potential issues that may occur if the code is merged
- Review the code, style, and be on the look out for any issues
  - Hard-coded values
  - Logic errors
  - Anti-patterns
- Don’t be afraid to ask questions or justification!

Merging a PR

Click “Merge pull request”

This will merge your feature changes into develop

Delete your feature branch

- Click “Delete branch”
- This makes sure that the repo is kept clean.

Remember, the only two perpetual branches are develop and master

Local Cleanup

In SourceTree

- Checkout develop
- Right-click feature branch -> “delete” -> check “Force delete” -> “Ok”
- Pull develop branch

Notice how we are back to the two perpetual branches, develop and master

Let’s Release

Responsibilities

This will only be performed be the person on the team who is responsible for coordinating releases.

Make the release branch

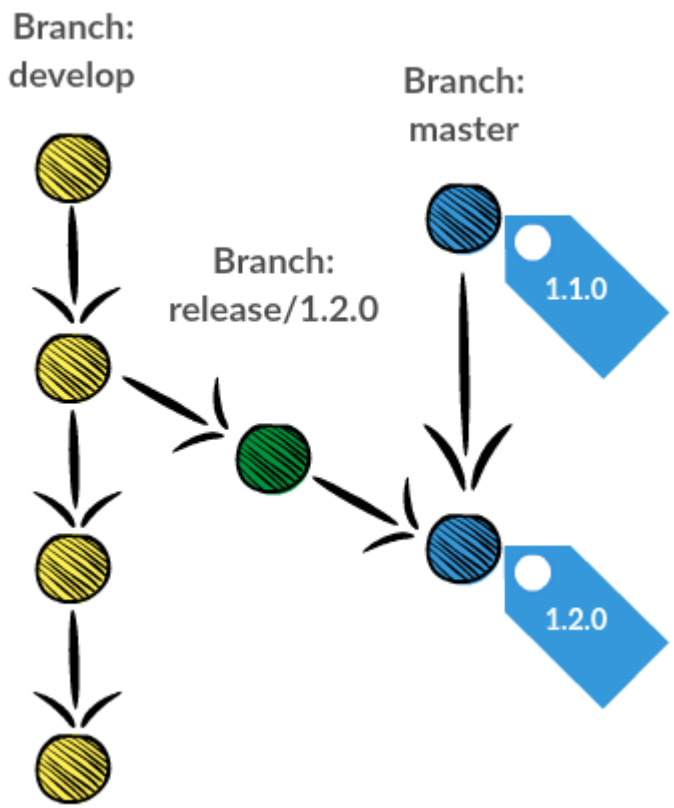
Naming Convention

release/[version number]

i.e “release/1.2.0”

Notes

- Branch from develop
- Push to github
- This branch goes through testing



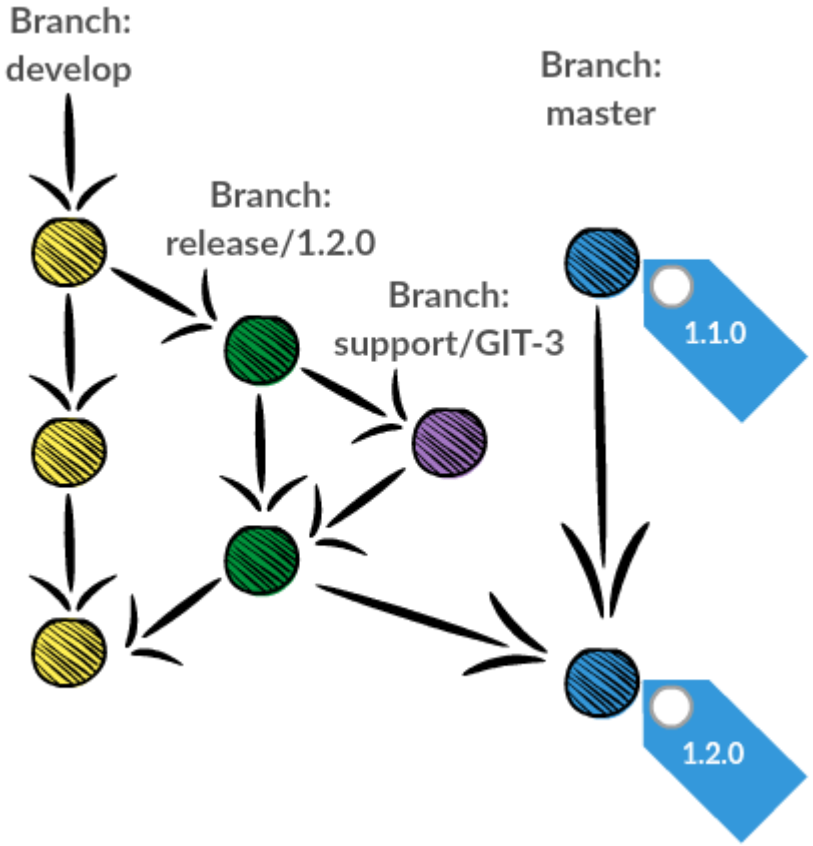
Check out Jenkins!

Release build will deploy to the release environment

Oh no! A bug in the release!

Handling a Release Bug

- Make sure that you are in the release branch
- Create a support branch from the release branch
- Fix the bug
- Push the code to the support branch
- PR from support into release



Release to Production

Create Pull Requests

- Create a PR for release branch -> master
- Create a PR for release branch -> develop

Get PRs approved

These PRs are very important! Remember, we are pushing to production.

Merge

Merge release branch into master and develop

Cleanup

Cleanup the release branch by deleting it on Github

Tag

Tag master as a new release in Github

Hotfixes

Reason To Use

- Occasionally, we will have an issue with a production release that needs to be addressed immediately.
- Normally, we would add the bug fix to our next release from develop, and treat it as a feature branch. However, sometimes this is not a fast enough approach.
- This is the scenario when we use hotfixes

Essentially

Is there a high priority bug in production?

Does it need to be resolved before the next scheduled release?

If you answered yes to both of the above, this is a candidate for a hotfix.

Hotfix

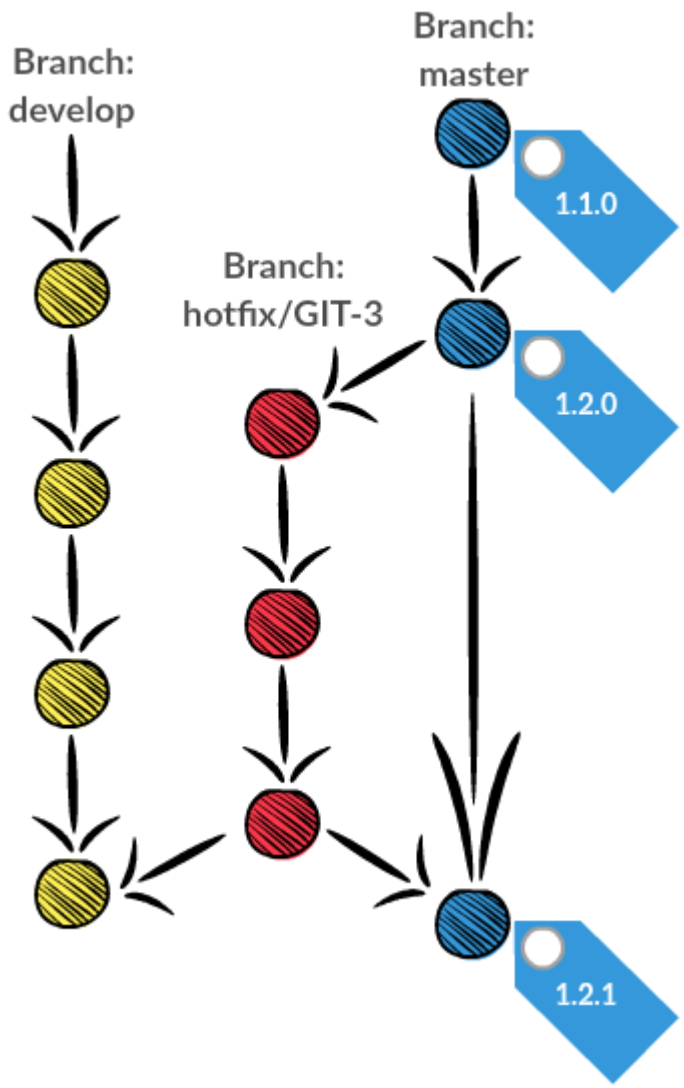
Naming Convention

hotfix/[Jira number]

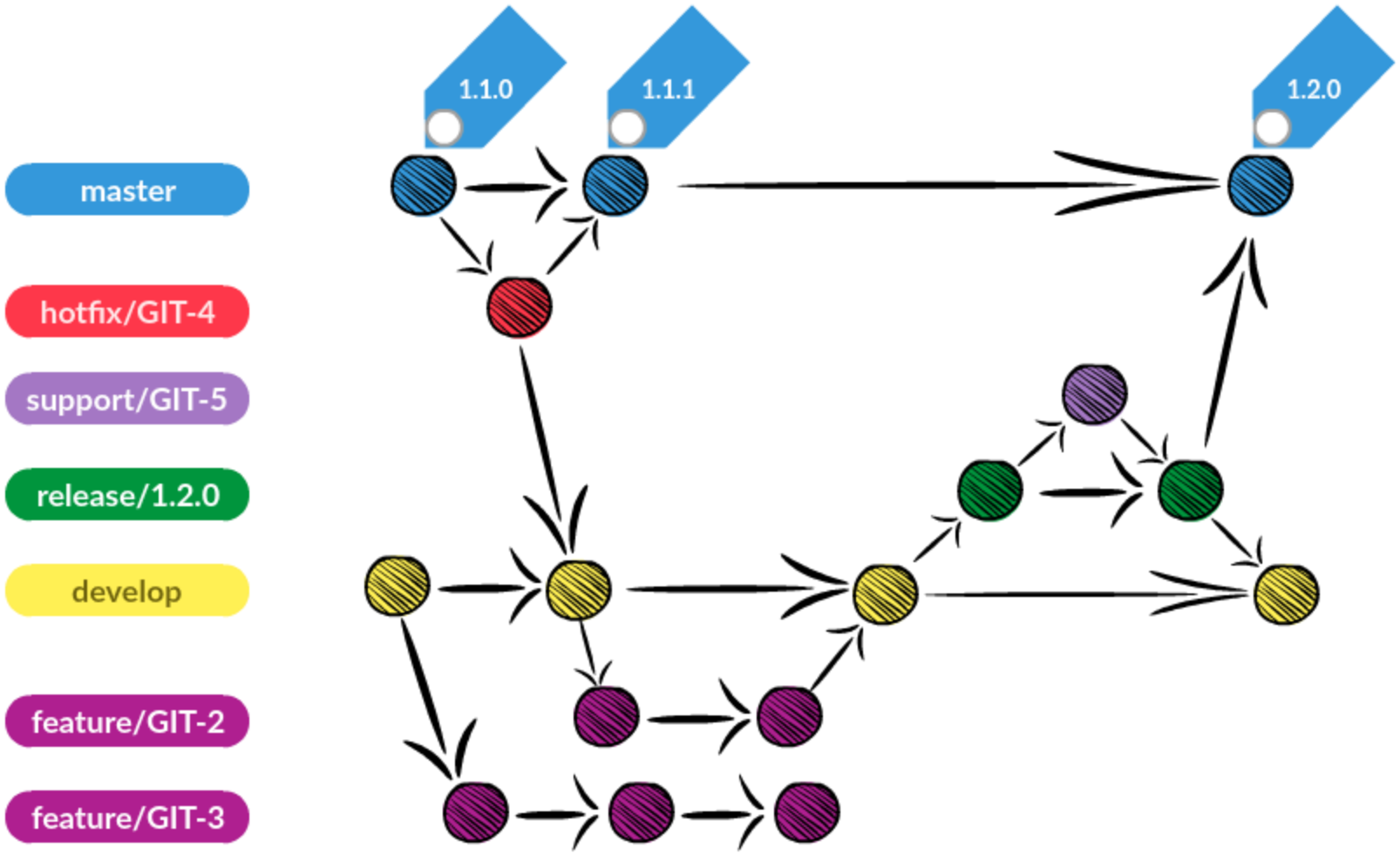
i.e “hotfix/GIT-3”

Notes

- Branch off of master
- Testing should occur at this point
- After testing is approved, create a PR into master and develop
- Merge into master and develop
- Tag master with the appropriate version number



Recap and Q&A


1.1.0
1.1.1
1.2.0
master
hotfix/GIT-4
support/GIT-5
release/1.2.0
develop
feature/GIT-2
feature/GIT-3

Application to new projects

- New projects are a prime candidate for this workflow
- Just follow the practices from the start of the project

Application to existing projects

- The upgrade path for each application is going to be different
  - But should be possible!
- I would strongly encourage at least one person on each team consider what a migration of the existing project would look like.
  - It will probably be a lot easier than you expect
- Some resources
  - Git-tfs for migrating from tfs without losing commit history/branches
    - https://github.com/git-tfs/git-tfs
  - Tfvc, git comparison, if you are wondering what tfvc analogs are in git
    - https://docs.microsoft.com/en-us/vsts/tfvc/comparison-git-tfvc

Resources

- Github getting started guide: https://guides.github.com/activities/hello-world/
- Git Docs: https://git-scm.com/doc
- Git Flow Docs: https://jeffkreeftmeijer.com/git-flow/
- Git Flow Seminal Work: http://nvie.com/posts/a-successful-git-branching-model/
- Demo for this workshop: https://github.com/CIGInsurance/DemoForGithub
- Why use VCS: https://www.git-tower.com/learn/git/ebook/en/desktop-gui/basics/why-use-version-control
- Why use Git: https://www.atlassian.com/git/tutorials/why-git
- Git Cheat Sheet: http://files.zeroturnaround.com/pdf/zt_git_cheat_sheet.pdf
- Semver Standard: https://semver.org/spec/v2.0.0.html


Q
A

thank you!